

Accuracy and Performance of FFT Software Libraries

HIGH PERFORMANCE KERNELS LLC

1 INTRODUCTION

This is an update to the paper by Caprioli and Jenkins [1] to show the latest accuracy and performance data for `hpk::fft` version 0.4.0 on both `x86_64` and `aarch64`. This is a rough draft. The data has been collected and the graphs have been made, but that's about it. Please see our original paper for background and discussion.

2 GENERAL INFORMATION

For comparison on Intel hardware, we use the Intel Math Kernel Library version 2024.0 for single and double precision and the Intel Integrated Performance Primitives version 2021.10 for half precision. These library versions are packaged for download as part of the oneAPI BaseKit version 2024.0.0. At the time of this writing, MKL does not support half precision FFTs at all, and IPP has only limited support. In particular, the functions `ippsDFT{Fwd,Inv}_Direct_CToC_16fc` therein do not support scaling, do not support in-place computations, and are limited to the 58 specific transform lengths listed in Table 1. For single and double precision, we use the 400 lengths listed in Table 2.

We also compare against FFTW3, using version 3.3.10-1 as built and distributed in Debian12 for `amd64` and version 3.3.8-10.amzn2023.0.2 as built by Amazon Web Services (AWS) and distributed in Amazon Linux for `aarch64`. Neither supports half precision.

Results for `x86_64` were obtained on a system having an Intel Xeon w7-2495X processor (formerly Sapphire Rapids) with 24 cores. Only single-threaded performance was measured, with this code path selected by setting the environment variable `OMP_NUM_THREADS=1`. We measured AVX2 performance for MKL by setting the environment variable `MKL_ENABLE_INSTRUCTIONS=AVX2` and for `hpk::fft` by making the factory with configuration `hpk::Architecture::avx2`. The operating system was Debian12.

Results for `aarch64` were obtained on an AWS EC2 compute instance having a Graviton3E processor. Only a single virtual CPU was provisioned. The operating system was Amazon Linux 2023.3.20231218.

3 ACCURACY RESULTS

The following is a summary of `hpk::fft` accuracy given as a ratio to other libraries, where

$$\text{accuracy} = \frac{\text{error}_{\text{other}}}{\text{error}_{\text{hpk::fft}}}$$

Thus, values greater than 1.0 indicate that `hpk::fft` is the more accurate. The values in the table below are the geometric mean of the tested transforms. Note that for the `ipp` comparison only the lengths in Table 1 were used. The `fftw` and `mkl` results for single and double precision use the longer list of sizes in Table 2.

<code>hpk::fft</code>	other	precision	ratio
<code>hpk_avx2</code>	<code>fftw_deb12</code>	fp32	1.105
		fp64	1.118
	<code>mkl_avx2</code>	fp32	1.048
		fp64	1.218
<code>hpk_avx512</code>	<code>ipp_avx512</code>	fp16	1.040
	<code>mkl_avx512</code>	fp32	1.197
		fp64	1.379
<code>hpk_sve256</code>	<code>fftw_aws</code>	fp32	1.023
		fp64	1.054

Next, we show the error for half, single, and double precision on a single graph using a log scale for both axes. It's not possible to see any details at this high level, but it allows everything to fit and gives the overall picture. Only `hpk::fft` data is shown, so all 400 lengths from Table 2 are used for all three precisions.

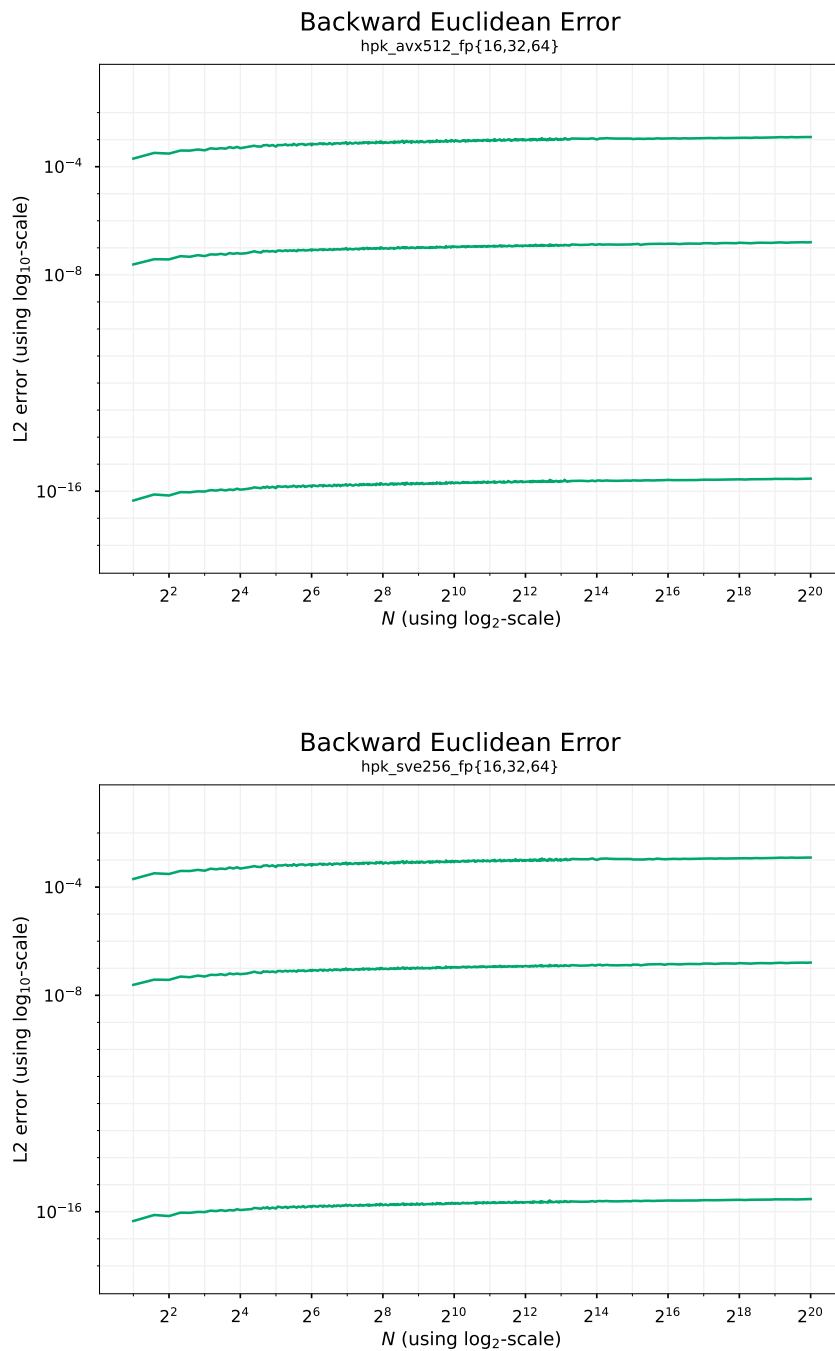


Fig. 1. Error for `hpk::fft` on architectures `x86_64` (top) and `aarch64` (bottom)

In all of the graphs that follow, we compare `hpk::fft` to some other library. The y -axis uses a linear scale for the error, and the x -axis is the transform length in strictly increasing order. The results are spaced equally apart; there is not a uniform horizontal scale. A tic mark is placed at every power-of-two length.

3.1 Half Precision

In Figure 2, we see the error for the 58 lengths supported by IPP.

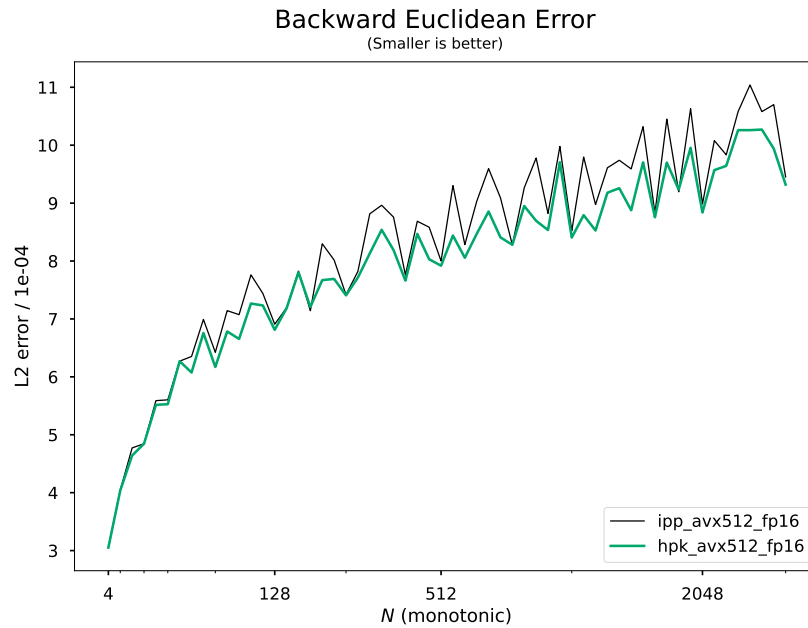


Fig. 2. Error for AVX512 half precision vs. IPP

3.2 Single Precision

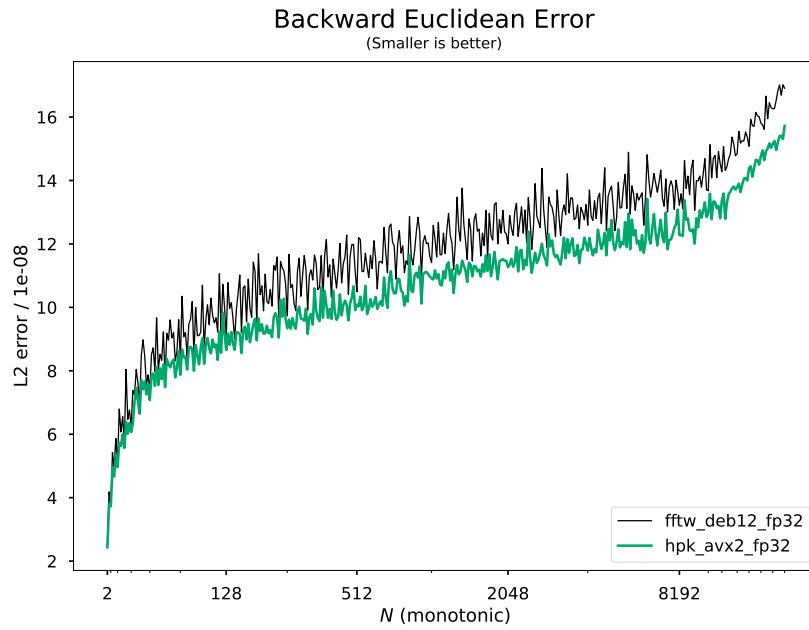


Fig. 3. Error for AVX2 single precision vs. FFTW

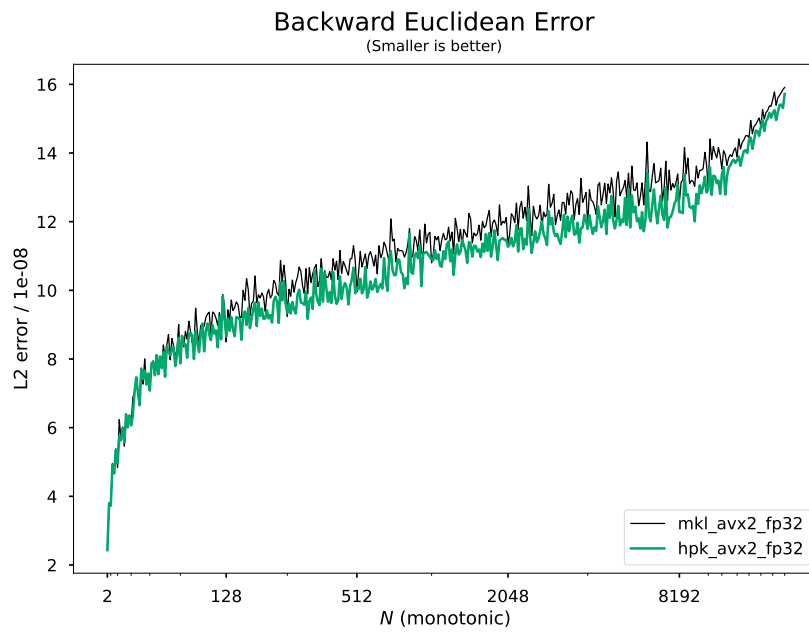


Fig. 4. Error for AVX2 single precision vs. MKL

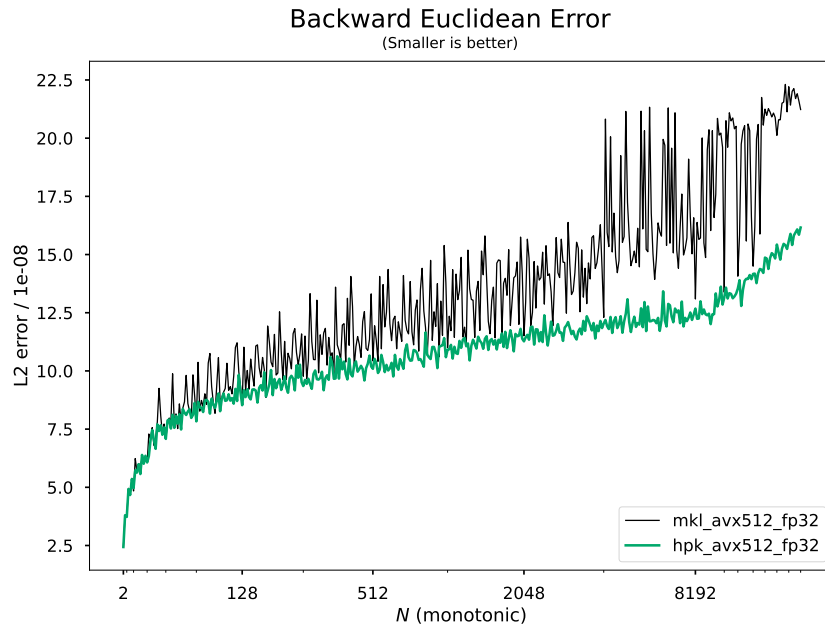


Fig. 5. Error for AVX512 single precision vs. MKL

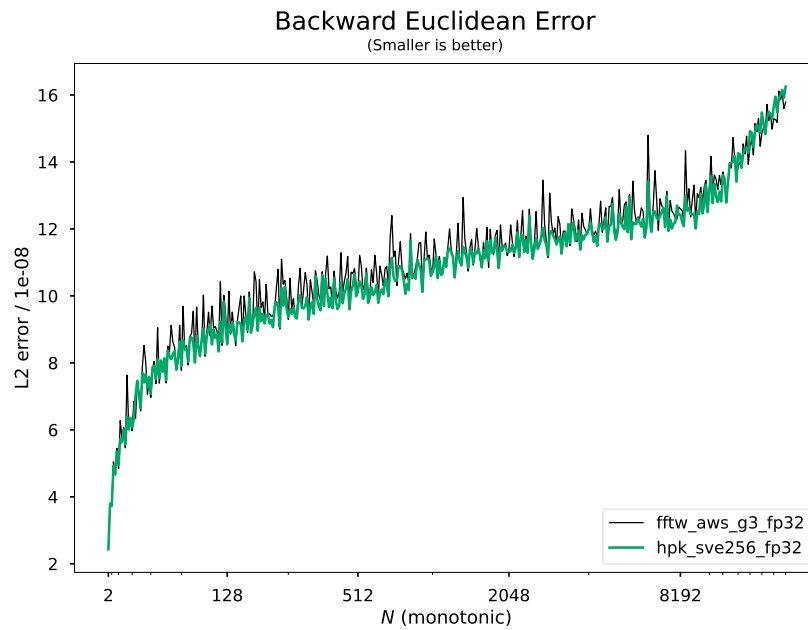


Fig. 6. Error for AArch64 SVE256 single precision vs. FFTW

3.3 Double Precision

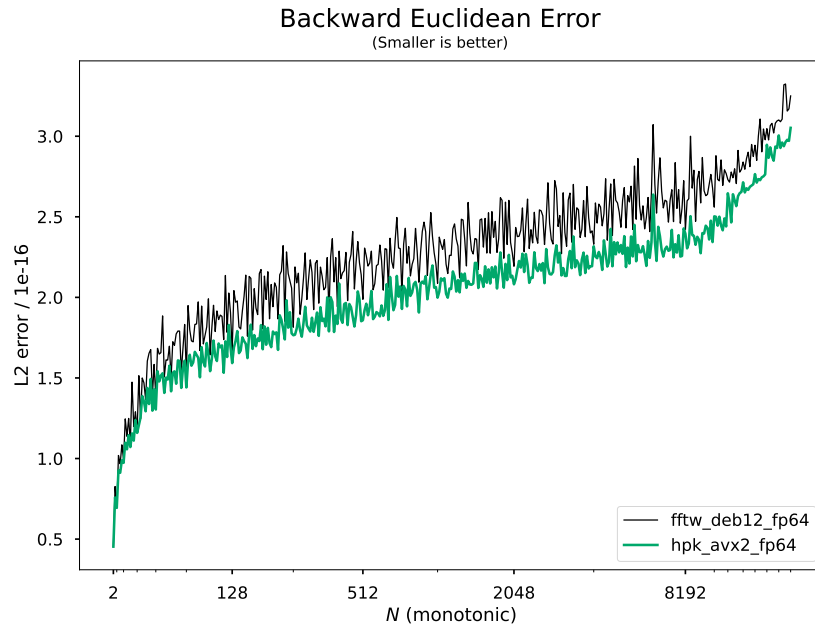


Fig. 7. Error for AVX2 double precision vs. FFTW

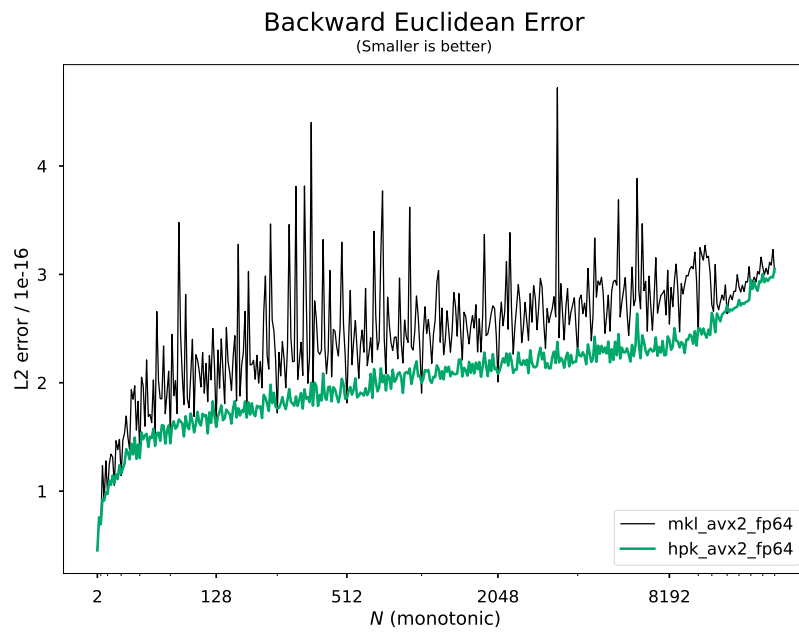


Fig. 8. Error for AVX2 double precision vs. MKL

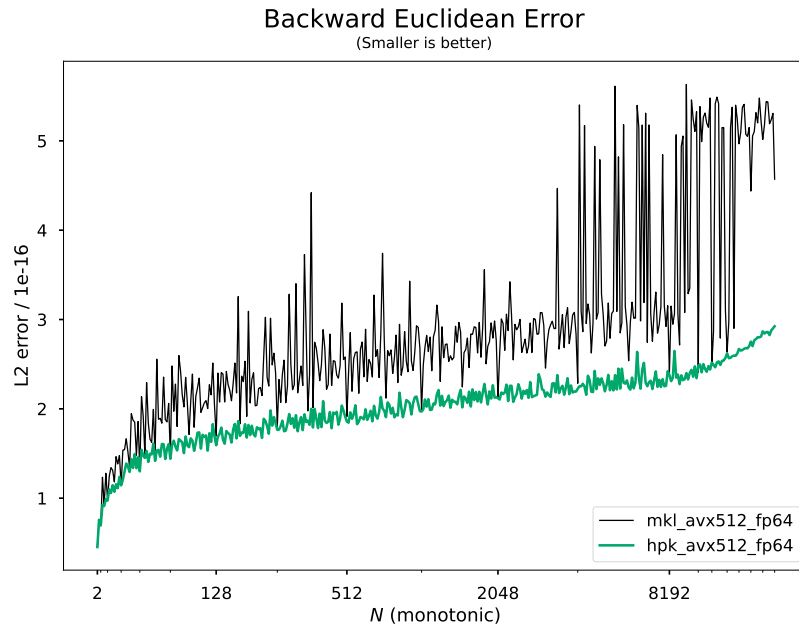


Fig. 9. Error for AVX512 double precision vs. MKL

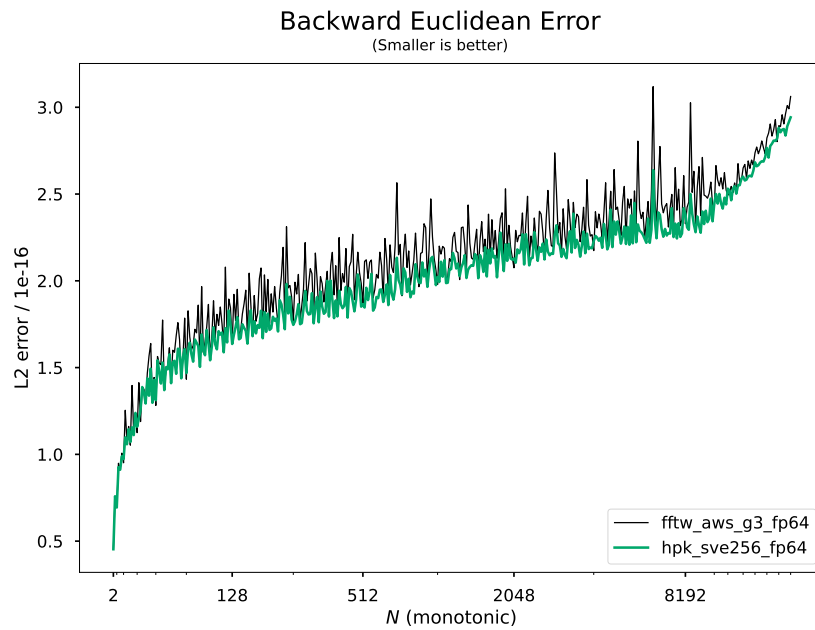


Fig. 10. Error for AArch64 SVE256 double precision vs. FFTW

4 PERFORMANCE RESULTS

For each transform length, four FFTs are computed, and the graphs are created after taking the geometric mean of the four performance ratios. For single and double precisions, the four test cases are the cross product of forward and backward, in-place and out-of-place. For half precision, Intel IPP only supports out-of-place computations, so we measured and compared only that. Unless otherwise noted, the batch size is one, i.e., a single sequence of length N is transformed. The suffix `_b7` indicates that a batch size of seven was used.

The following is a summary of `hpk::fft` performance given as a ratio to other libraries, where

$$\text{performance ratio} = \frac{\text{time}_{\text{other}}}{\text{time}_{\text{hpk::fft}}}$$

Therefore, results greater than 1.0 indicate that `hpk::fft` has better performance.

<code>hpk::fft</code>	other	precision	ratio
<code>hpk_avx2</code>	<code>fftw_deb12</code>	fp32	2.228
		fp64	1.514
	<code>mkl_avx2</code>	fp32	1.617
		fp64	1.412
<code>hpk_avx512</code>	<code>ipp_avx512</code>	fp16	1.156
		fp16_b7	1.217
	<code>mkl_avx512</code>	fp32	1.358
		fp64	1.318
<code>hpk_sve256</code>	<code>fftw_aws</code>	fp32	3.882
		fp64	2.180

4.1 Half Precision

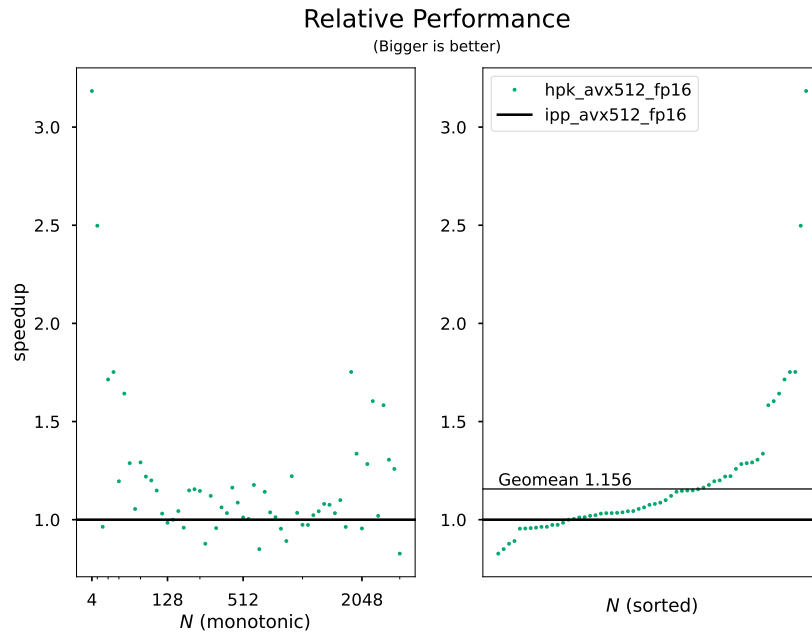


Fig. 11. Performance for AVX512 half precision vs. IPP

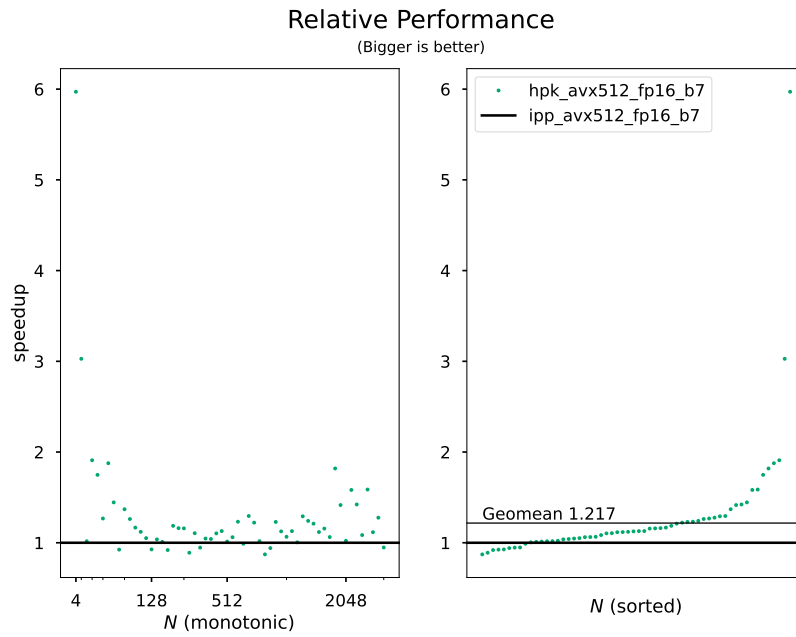


Fig. 12. Performance for AVX512 half precision vs. IPP for a batch of seven transforms

4.2 Single Precision

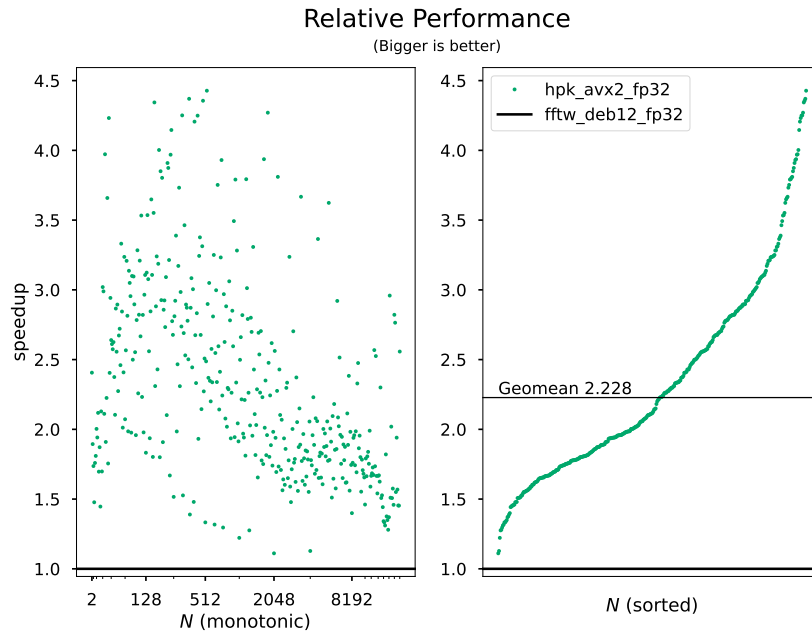


Fig. 13. Performance for AVX2 single precision vs. FFTW

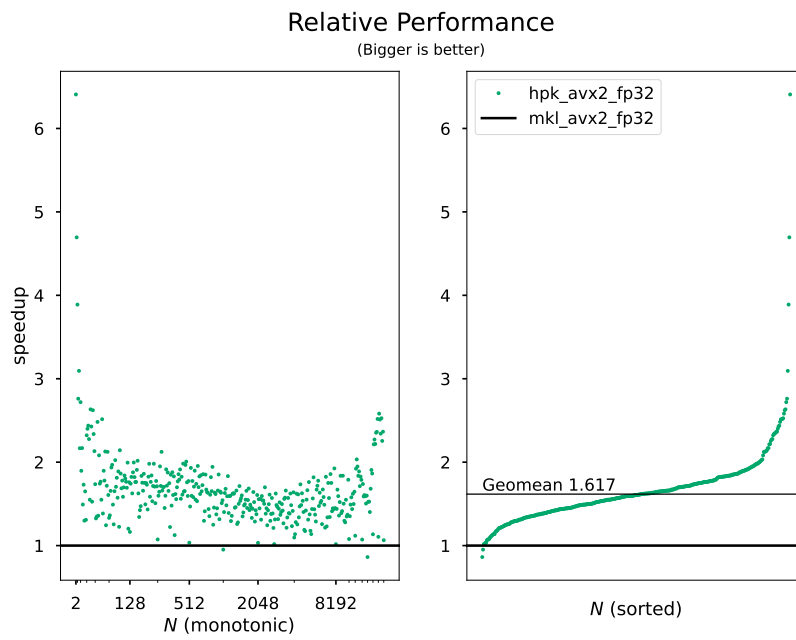


Fig. 14. Performance for AVX2 single precision vs. MKL

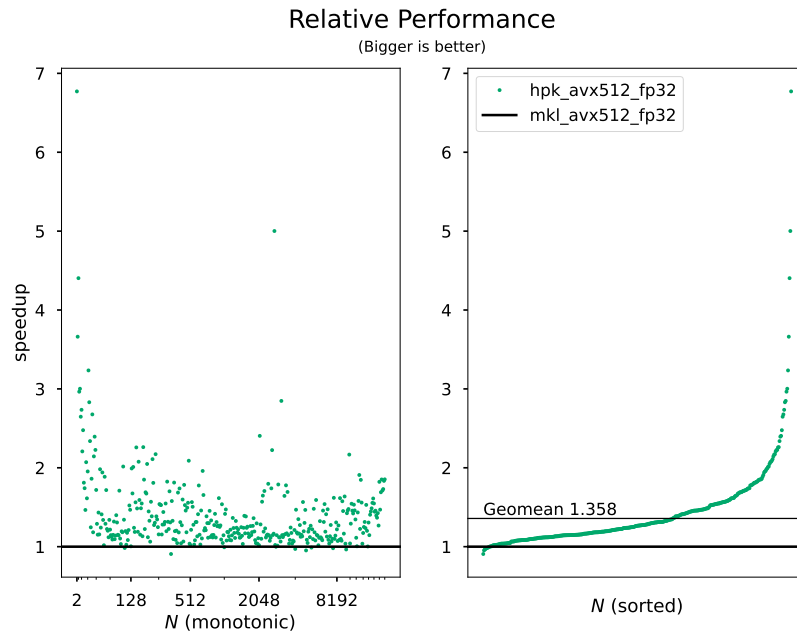


Fig. 15. Performance for AVX512 single precision vs. MKL

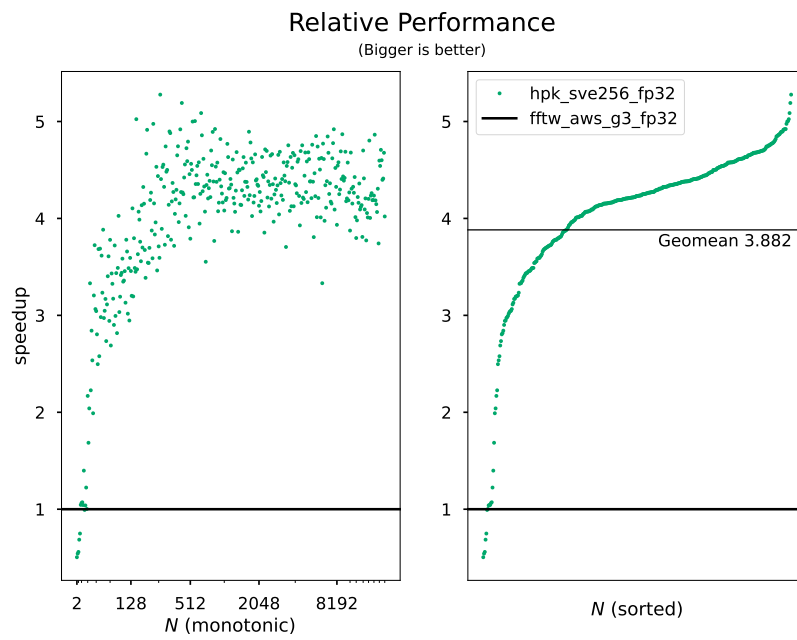


Fig. 16. Performance for AArch64 SVE256 single precision vs. FFTW

4.3 Double Precision

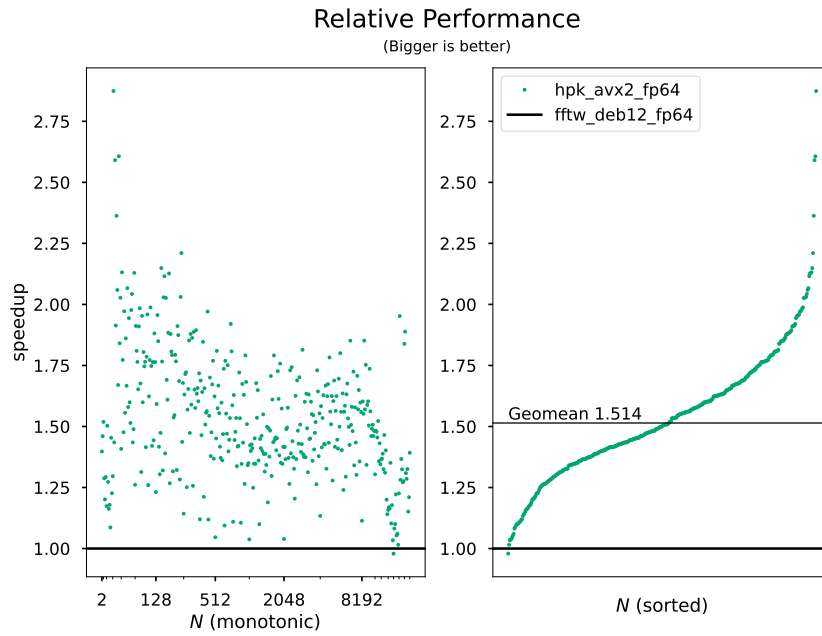


Fig. 17. Performance for AVX2 double precision vs. FFTW

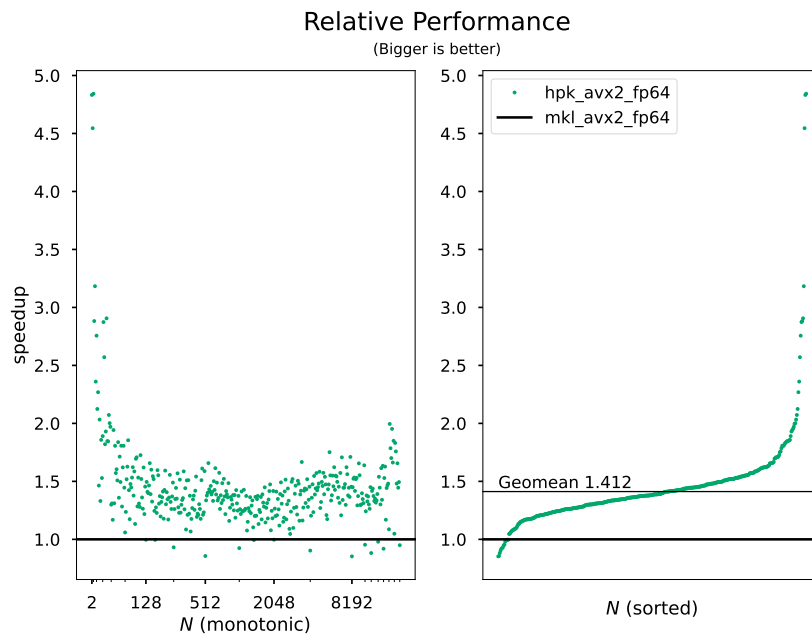


Fig. 18. Performance for AVX2 double precision vs. MKL

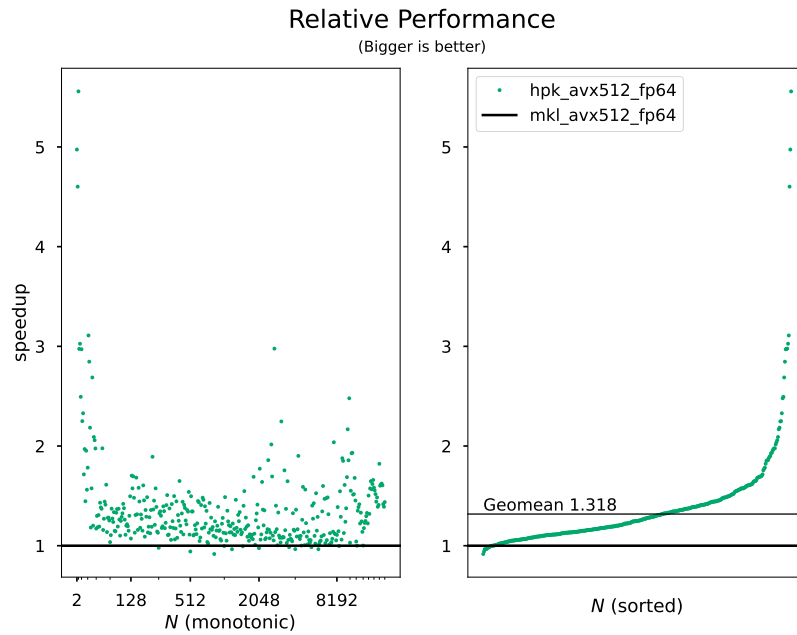


Fig. 19. Performance for AVX512 double precision vs. MKL

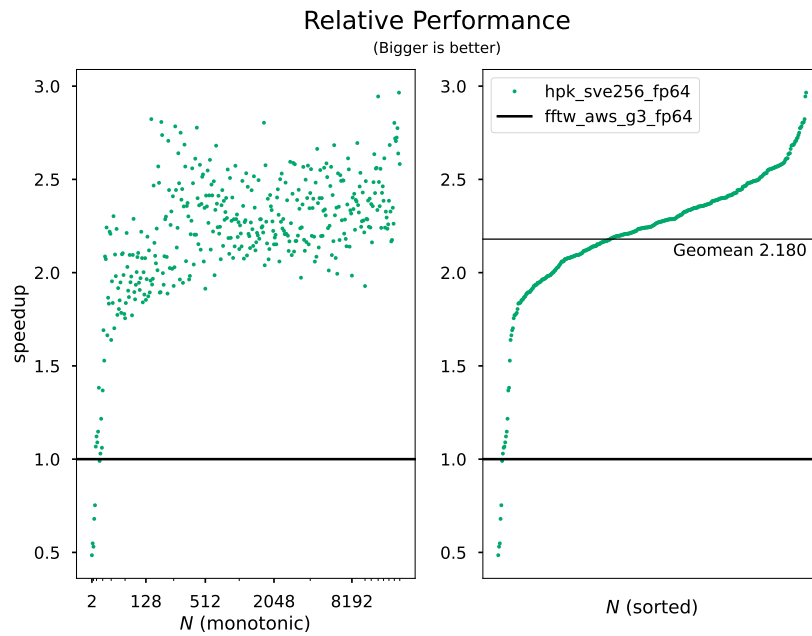


Fig. 20. Performance for AArch64 SVE256 double precision vs. FFTW

5 PYTHON

Results were obtained on x86_64 using Debian12, which provides Python 3.11.2, NumPy 1.24.2, and SciPy 1.10.1.

The SciPy FFT functions return a NumPy array as the result, so for performance we measure only the Hpk out-of-place functions that do the same. Note that Hpk also provides `forwardCopy` and `backwardCopy`, which write results into an existing array to avoid allocating a return array. Furthermore, Hpk provides in-place FFT compute functions, which overwrite the input data with the results. These are faster and are recommended when the input data is not subsequently needed.

5.1 Summary

Hpk is more accurate:

	precision	ratio
scipy	float32	1.124
	float64	1.233

and has better performance:

	precision	ratio
scipy	float32	3.252
	float64	2.497

5.2 Accuracy

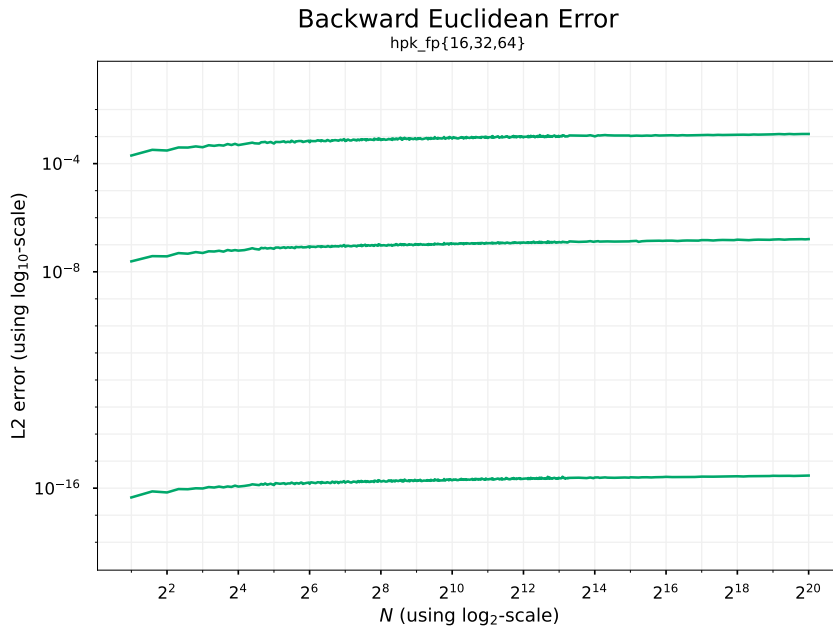


Fig. 21. Error for Hpk on a log-log scale for float16, float32, and float64.

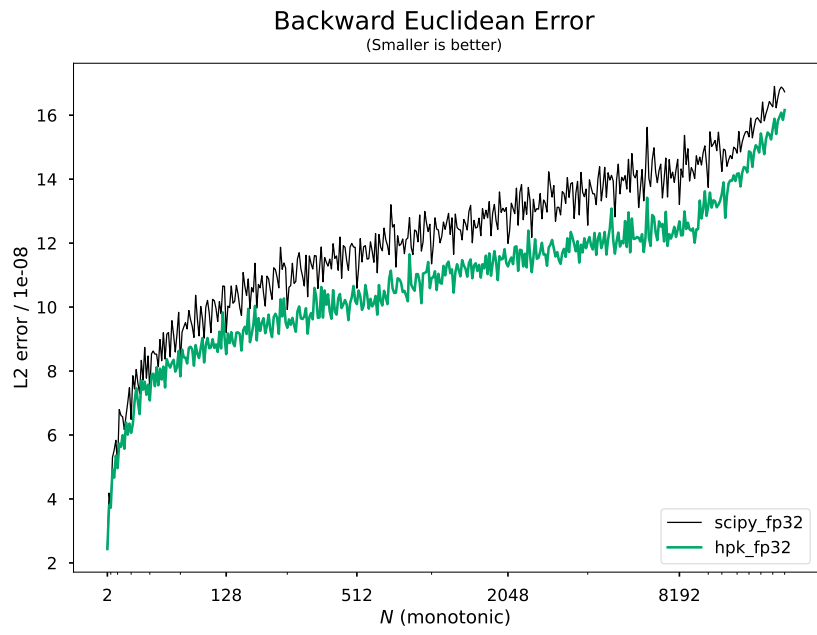


Fig. 22. Error for single precision vs. SciPy

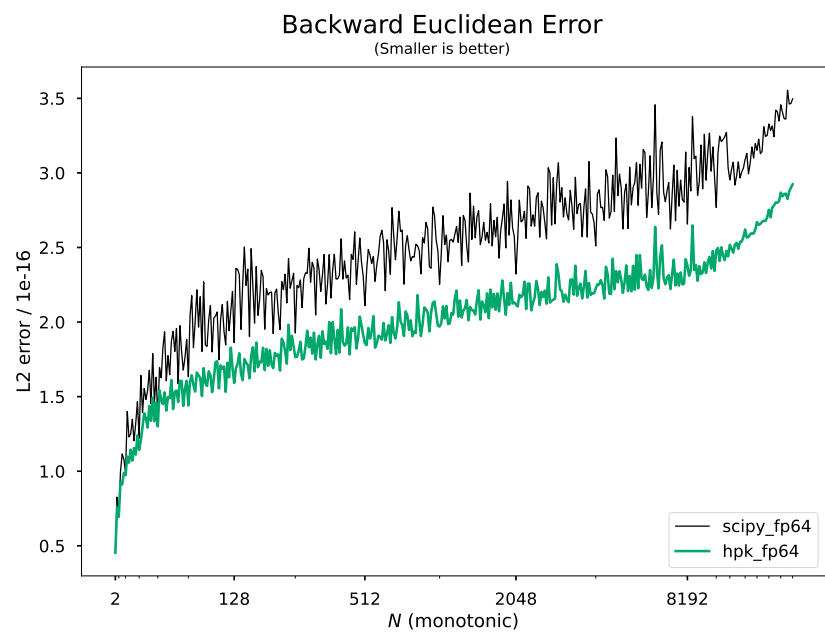


Fig. 23. Error for double precision vs. SciPy

5.3 Performance

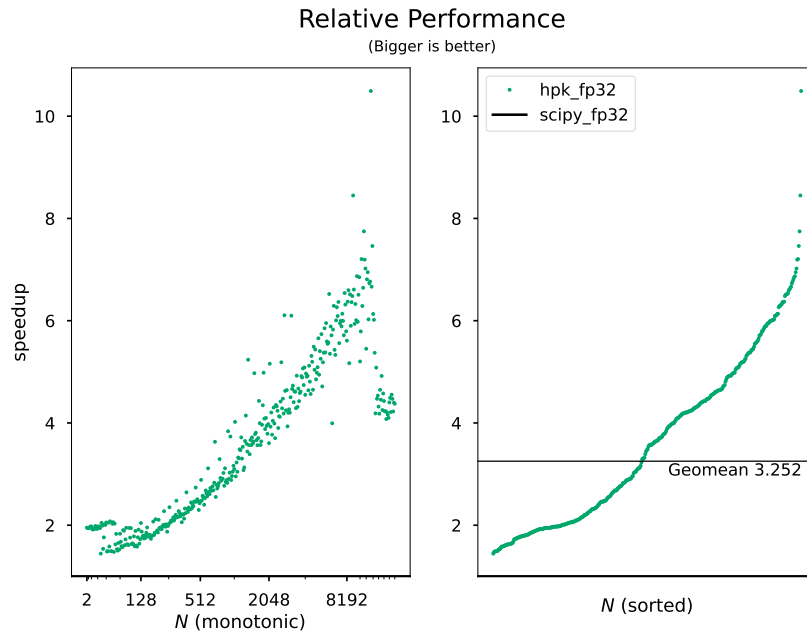


Fig. 24. Performance of single precision vs. SciPy

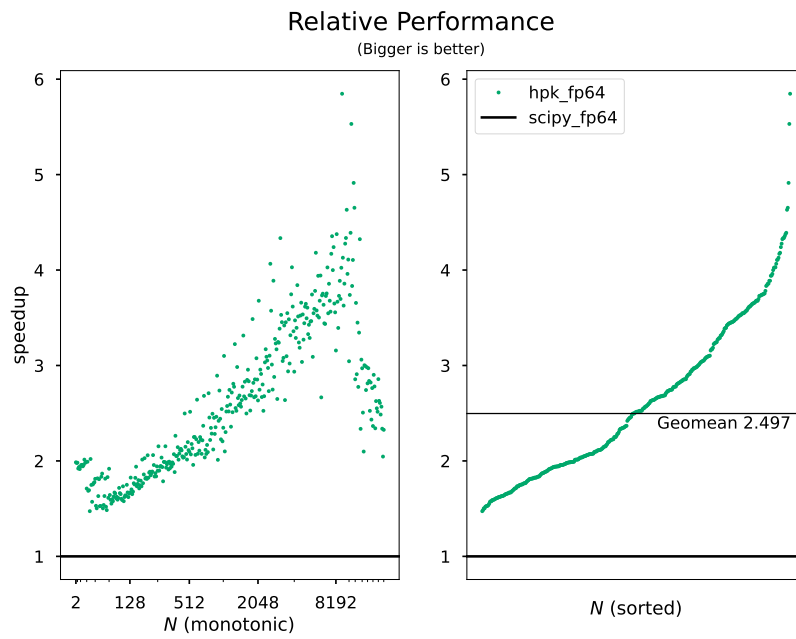


Fig. 25. Performance of double precision vs. SciPy

6 DOWNLOADING

High Performance Kernels for FFT, along with documentation, is available at <https://hpkfft.com>

REFERENCES

- [1] Paul Caprioli and Robby Jenkins. 2023. High Performance Kernels for FFT via Modern C++. <https://doi.org/10.5281/zenodo.8253863>

Table 1. FFT lengths for half precision experimental results

4	24	60	108	180	256	360	512	648	900	1080	1440	1920	2400	3240
8	32	64	120	192	288	384	540	720	960	1152	1500	1944	2700	4096
12	36	72	128	216	300	432	576	768	972	1200	1536	2048	2916	
16	48	96	144	240	324	480	600	864	1024	1296	1620	2160	3000	

Table 2. FFT lengths for single and double precision results

2	32	78	144	234	360	528	800	1210	1764	2560	3584	5184	7000	12000	98304
3	33	80	150	240	375	540	840	1225	1792	2592	3600	5292	7056	14000	114688
4	35	81	154	243	384	550	864	1232	1800	2600	3780	5376	7168	15000	122880
5	36	84	156	245	385	560	875	1260	1815	2640	3840	5400	7200	16000	131072
6	39	88	160	250	390	576	880	1280	1872	2688	3888	5488	7500	16384	147456
7	40	90	162	252	392	588	896	1296	1920	2700	3920	5500	7680	17500	163840
8	42	91	165	256	396	600	900	1300	1944	2704	4000	5600	7776	18000	180224
9	44	96	168	260	400	624	924	1320	1960	2744	4032	5625	7840	20000	196608
10	45	98	175	264	405	630	936	1344	1980	2800	4096	5760	7920	21000	229376
11	48	99	176	270	420	640	945	1352	2000	2880	4116	5832	8000	24000	245760
12	49	100	180	280	432	648	960	1372	2016	2916	4200	5880	8064	29400	262144
13	50	104	182	288	440	660	968	1400	2048	2970	4320	6000	8100	32000	294912
14	52	105	189	294	441	672	972	1440	2080	3000	4400	6048	8192	32768	327680
15	54	108	192	297	448	675	990	1500	2100	3024	4480	6144	8400	36864	360448
16	55	110	195	300	450	700	1000	1512	2160	3072	4500	6272	8640	38400	393216
18	56	112	196	308	462	720	1008	1536	2200	3120	4536	6300	8748	40960	458752
20	60	117	198	312	468	728	1024	1540	2205	3136	4608	6400	9000	45056	491520
21	63	120	200	315	480	729	1040	1560	2240	3200	4704	6480	9072	49152	524288
22	64	125	208	320	490	735	1050	1568	2250	3240	4725	6561	9216	53248	589824
24	65	126	210	324	495	756	1080	1584	2268	3300	4800	6600	9240	57344	655360
25	66	128	216	330	500	768	1100	1600	2304	3360	4860	6720	9360	61440	720896
26	70	130	220	336	504	770	1120	1620	2352	3375	4900	6750	9600	65536	786432
27	72	132	224	343	512	780	1152	1680	2400	3456	5000	6804	9720	73728	917504
28	75	135	225	350	520	784	1176	1694	2500	3500	5040	6860	9800	81920	983040
30	77	140	231	352	525	792	1200	1728	2520	3528	5120	6912	10000	90112	1048576