The Diagonal Multidimensional FFT

Versus the Polynomial transform

PhD Þorgeir Sigurðsson. ERCIM post-doc fellow at NTNU, thorgeir.sigurdsson@ntnu.no, thorgsi@outlook.com

PhD Sven Sigurðsson. Professor emeritus of computational science, University of Iceland.

PhD Stefán Ingi Valdimarsson. Head of Risk Analysis at Arion bank, Iceland

What is the Diagonal transform?

- The Diagonal FFT transform was developed by the authors. Asymptotically, for large transforms, it reduces the multiplication count in an m-dimensional FFT by a factor m, compared to the Row/Column transform [1]. This can, however, be achieved with a Polynomial FFT as well.
- An article on our D-transform [2] was not deemed to demonstrate enough of an improvement over the P-transform by editors of the IEEE DSP journal in 2013.
- The Diagonal transform derives from the Vector transform.

Why the name Diagonal?

The traditonal m dimensional FFT (the Row/Colmn transform) operates on numbers along one dimension at a time. Vector transforms operate on arrays along diagonals.

The Diagonal transform uses a more efficent scheme than older vector transforms and we can show that this scheme results asymtotically in a reduction of complex multiplications by a factor m compared to the Row/Column transform. All multiplications in dimensions above one, are free.

How relevant are the multiplications?

The number of multiplications used to be the only relevant measure of efficiency of an FFT algorithm, but it no longer is. This work demonstrates that other measures, where the D-transform excels, can be of greater importance.

It compares the performance of our C implementation of the Diagonal transform [3] with that of the Polynomial transform, as practically implemented by R. Bernardini [4]. None of our programs is optimized for a given computer architecture.

Testing on four machines

- Machine 1: 2 Cores 8 GB 2,2 GHz
- Machine 2: 4 Cores 8 GB 2,1 GHz
- Machine 3: 4 Cores 16 GB 1,9 GHz
- Machine 4: 10 Cores 128 GB 3,7 GHz

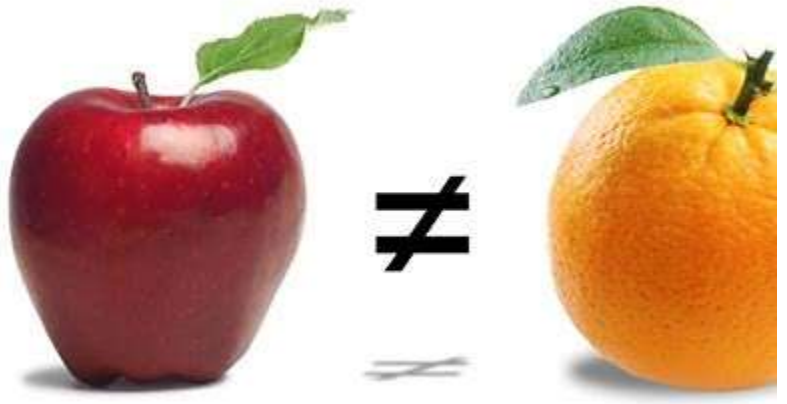
All are 64 bit Windows 10 machines with a x64 type of processor.

Code is produced on each machine by the GCC compiler with the following option flags:

- a) none, b) optimize for speed, c) tailor to native system

Are we comparing apples to oranges?

Yes and No. When comparing the Diagonal transform to Vector transforms, we are comparing apples of different type.



The Diagonal versus Vector transforms

The Diagonal transform outperforms (other) Vector transforms in every respect.

1) In number of multiplications

2) In number of additions

3) In speed on all four machines for all tested transform sizes

See numbers for a 4096 x 4096 transform on following slides.

The percentages are relative to the Row/Column transform.

.

Arithmetic operations:

4096 x 4096 FFT

Real multiplies	Radix 2	Radix 4	Split Radix
Diagonal	259 123 648 59%	191 727 264 58%	173 970 736 58%
Vector	327 204 864 75%	210 788 352 63%	196 175 664 65%
Row/Column	436 273 152 100%	332 791 808 100%	302 022 656 100%

Real additions	Radix 2	Radix 4	Split Radix
Diagonal	1 064 430 016 86%	997 033 632 88%	979 277 104 88%
Vector	1 132 511 232 91%	1 016 094 720 89%	1 001 482 032 90%
Row/Column	1 241 579 520 100%	1 138 098 176 100%	1 107 329 024 100%

Speed:

4096 x 4096 FFT

Machine 4	Radix 2	Radix 4	Split Radix
Diagonal	2,24 sec 40%	2,60 sec 49%	1,87 sec 46%
Vector	2,54 sec 46%	3,34 sec 62%	2,08 sec 51%
Row/Column	5,55 sec 100%	5,35 sec 100%	4,12 sec 100%

The Diagonal transform is always faster than (other) Vector transforms.

The Split Transform is always faster than the R2 and the R4 transform but note that the R2 is faster than the R4 transform on this machine for the Vector transforms, despite fewer complex multiplications.

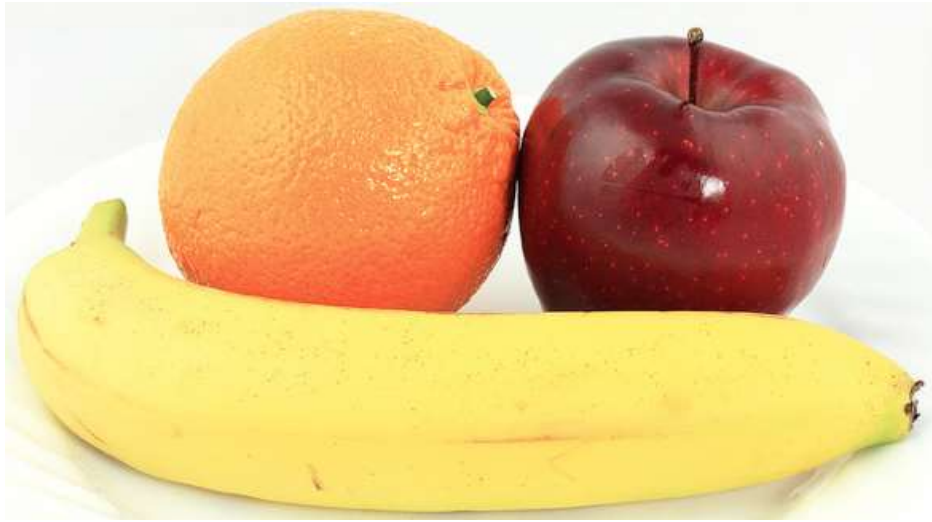
Times are for Machine 4 with GCC with compiler flags O3 and march=native

An orange, an apple, and a banana

The Row/Column transform works through one dimension at a time.

The Diagonal transform, like any Vector transform, works through them in one pass.

The Polynomial transform repeatedly rearranges the data.



The Diagonal versus the Polynomial transform

The Polynomial transform has fewer arithmetic operations

The Diagonal transform has less data transfer

For large transforms, the 2D Diagonal transform is faster than the 2D Polynomial transform, using the GCC compiler, on our four computers.

In the following slides, the percentages are relative to the Row/Column transform

Arithmetic operations:

4096 x 4096 FFT

Real multiplies, number of	Radix 2	Radix 4	Split Radix
Diagonal	259 123 648 59%	191 727 264 58%	173 970 736 58%
Polynomial	234 881 040 54%	178 677 360 54%	162 179 760 54%
Row/Column	436 273 152 100%	332 791 808 100%	302 022 656 100%

Real additions, number of	Radix 2	Radix 4	Split Radix
Diagonal	1 064 430 016 86%	997 033 632 88%	979277104 88%
Polynomial	1 040 187 408 84%	983 983 728 86%	967486128 87%
Row/Column	1 241 579 520 100%	1 138 098 176 100%	1107329024 100%

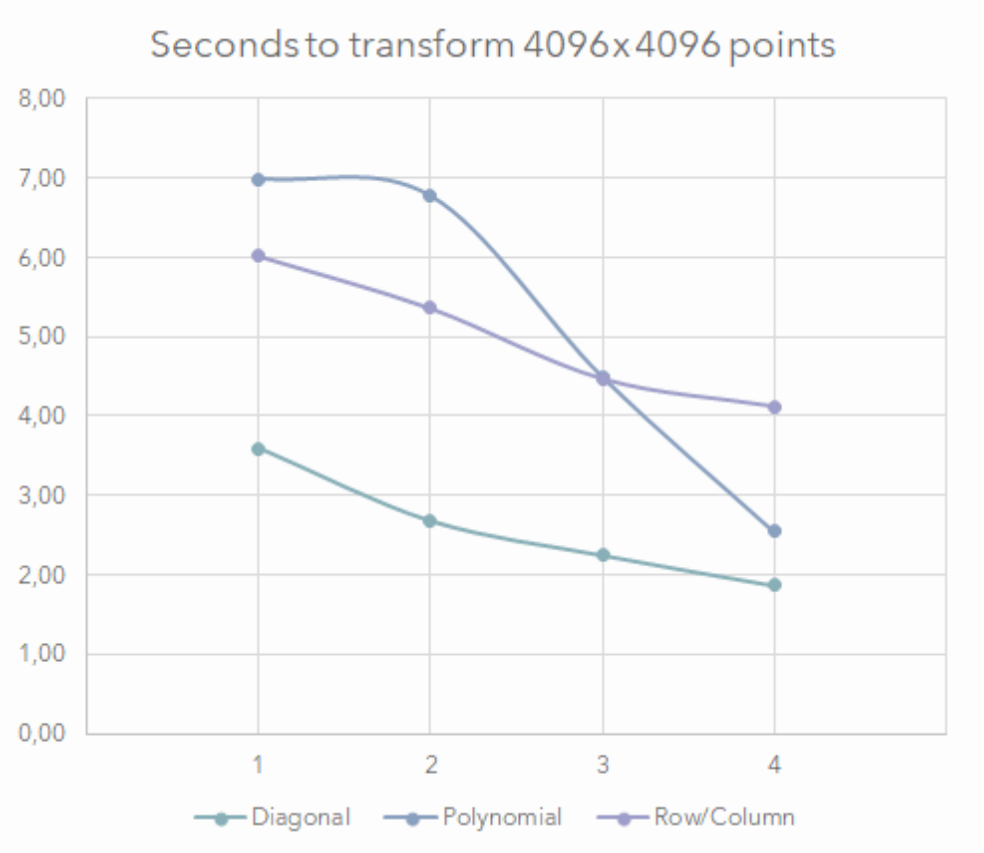
Speed:

4096 x 4096 FFT

Machine 3	Radix 2	Radix 4	Split Radix
Diagonal	2,64 sec 47%	3,01 sec 56%	2,25 sec 50%
Polynomial	5,42 sec 96%	5,60 sec 104%	4,50 sec 101%
Row/Column	5,68 sec 100%	5,38 sec 100%	4,47sec 100%

Here, the Diagonal transform is much faster than the Polynomial transform (the same applies to its related Vector transforms).

Times are for Machine 3 using GCC compiler with flag O3 (optimize for speed)

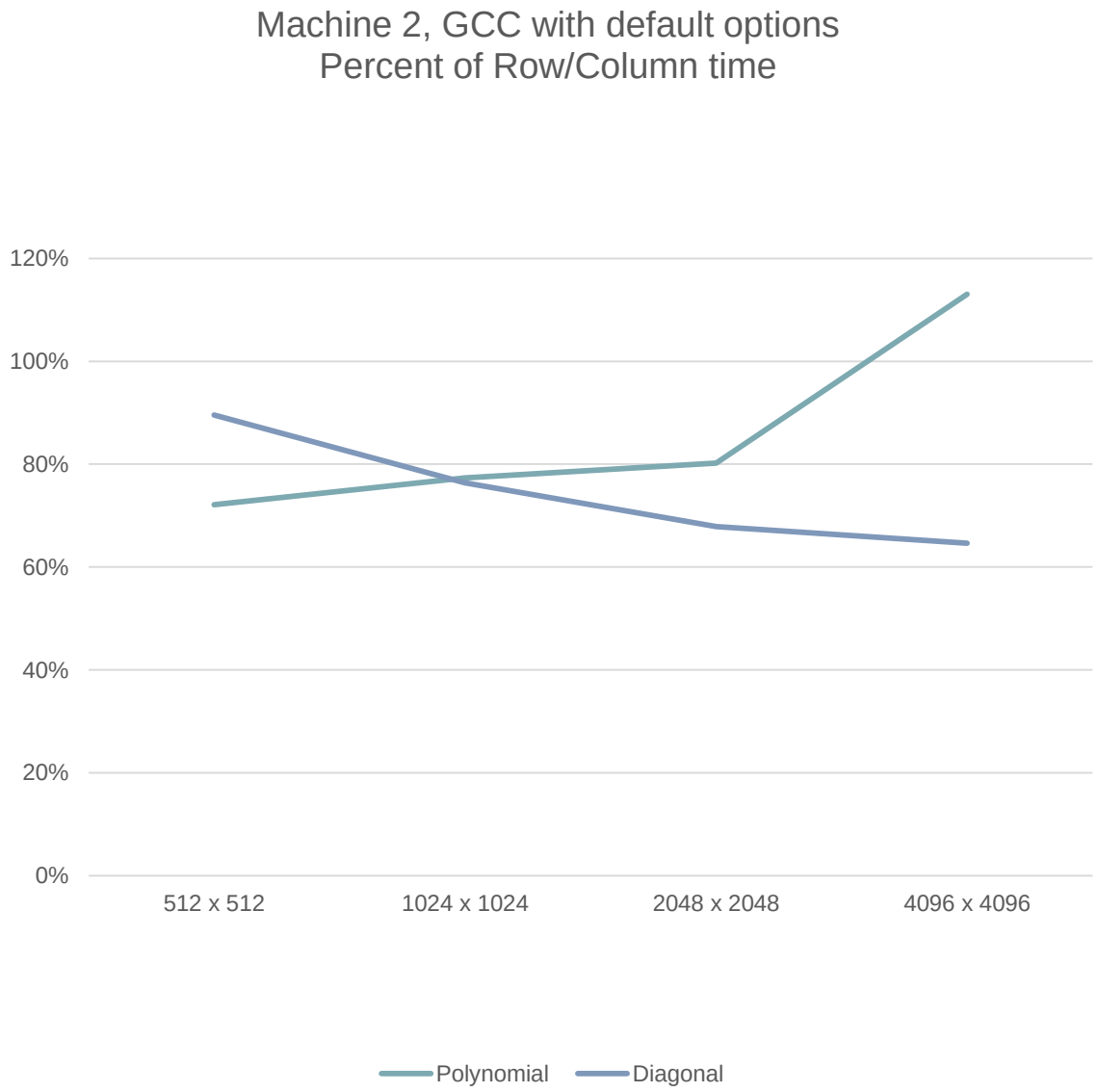


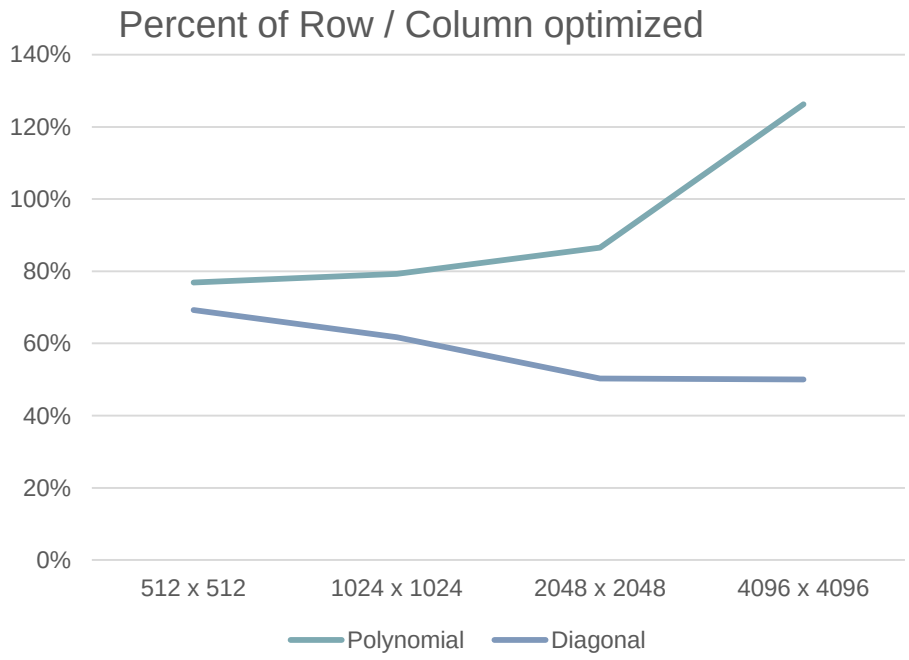
Max speed; four computers:

RAM memory size seems to determine the relative performance of the Polynomial transform.

The sizes are 8 GB, 8 GB, 16 GB and 128 GB.

A large array size punishes the Polynomial transform





The Diagonal transform responds better to compiler optimization

The optimization typically sacrifices memory for speed, which is probably less of an option for the Polynomial transform.

Conclusions

- We demonstrated that the number of arithmetic operations can be of little relevance when comparing FFT algorithms.
- Also, that relative performance of algorithims depends on the system used.
- For our implementations and for common types of Windows computers, the Polynomial transform is not competitive against the Diagonal transform or even the basic Row/Column transform unless much larger RAM is installed than is realistic for most users.

Appendix, the CPUs

- 1: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 2195 Mhz
- 2: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
- 3: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
- 4. Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz, 3696 Mhz

Caveats

- a) FFTW?
- b) Parallelism?
- c) What about the tangent transform?

References

- [1] 2013. Þorgeir Sigurðsson, Stefán Ingi Valdimarsson, Sven Sigurðsson. The Diagonal Algorithm for multidimensional discrete Fourier transforms. Talk by Stefán Ingi at University of Edinburgh joint Analysis and Applied Mathematics Seminar 4th September 2013. available at uni.hi.is/ths185/articles : Edinburgh4sept
- [2] 2013 Þorgeir Sigurðsson, Sven Sigurðsson, Stefán Ingi Valdimarsson. The Diagonal FFT. This is an unpublished paper with a fast method for calculating the multidimensional FFT, written 2013, available at uni.hi.is/ths185/articles The Diagonal FFT October 2013[2590]
- [3] 1979 H. J. Nussbaumer and P. Quandalle, "Fast computation of discrete Fourier-transforms using polynomial transforms," *IEEE Trans. Acoust. Speech Signal Process.,* vol. 27, no. 2, pp. 169-181, 1979.
- [4] 2000 R. Bernardini, "A new multidimensional FFT based on one-dimensional decompositions," *IEEE Trans. Circ. and Syst. II,* vol. 47, no. 10, pp. 1123-1126, Oct 2000.
- [5] 1984. Þorgeir Sigurðsson. A New Method for Smoothing Length Distributions of Fish and Sharpening Peaks due to Different Year Classes. International Council for the Exploration of the sea. C.M.1984/ D:7 Statistics Committee, available at uni.hi.is/ths185/articles. PDF Our implementation uses the complex-conjugate transform originally developed by one of the authors and published as part of new algorithm at a conference on fishery management (equivalent to the Split Radix FFT in structure and number of arithmetic operations).