Software Outlook: FFT Benchmarks for Fortran Codes

H. Sue Thorne

December 5, 2018

1 Introduction

As part of the 2018/19 Software Outlook Work Plan, we will be benchmarking a number of different Fast Four Transform (FFT) libraries with bindings for Fortran. The attributes of the different libraries are given in Section 5. Assuming indexing starts at 1, the discrete 1D Fourier transform of a vector x of length n is defined as

$$z(k) = \sum_{m=1}^{n} x(m) \exp(-2\pi i(k-1)(m-1)/n), \quad l = 1, \ldots, n. \tag{1}$$

1.1 Half-complex format

For input data that is purely real, the discrete Fourier transform satisfies the "Hermitian" redundancy: in 1D, if x is a real array, then z computed via (1) will be a complex array satisfying

$$z(k) = [z(n-k+2)]^*, \quad k = 2, \ldots, n.$$

Also note that the imaginary part of $z(1)$ is always 0; for n even, the imaginary part of $z(n/2+1)$ is also always 0. This special symmetry in z is known as *half-complex* format and means that it can be stored more efficiently using a real array y of length n. The method of storing z in y will vary according to the library being used but one possibility is to define the values of y as

$$\begin{aligned} y(1) &= real(z(1)), \\ y(i) &= real(z(i)), \quad i = 2, \ldots, \lfloor n/2 \rfloor + 1, \\ y(n-i+2) &= imag(z(i)), \quad i = 2, \ldots, \lfloor (n+1)/2 \rfloor. \end{aligned}$$

Half-complex format can be extended to more dimensions. Note that if the input vector x is half-complex format, then z will be a real vector.

2 1D Benchmark

Let A be a 2D array with dimensions $n_1 \times n_2$ and there be q 2D arrays B_i that are the same size as A and satisfy $\sum_i^q [B_i(j,k)]^2 = 1$ for each $j = 1, \ldots, n_1$ and $k = 1, \ldots, n_2$. The general benchmark will take the form of Algorithm 1, where $comp\_mult(A, B_i)$ is defined to be component-wise multiplication of A with B_i; $comp\_div(H_i, B_i)$ is defined to be component-wise division of H_i by B_i; `FFT` is the discrete Fast Fourier Transform and `IFFT` is the discrete inverse Fast Fourier Transform.

3 2D Benchmark

Let A be a 3D array with dimensions $n_1 \times n_2 \times n_3$ and there be q 3D arrays B_i that are the same size as A and satisfy $\sum_i^q [B_i(j,k,l)]^2 = 1$ for each $j = 1, \ldots, n_1$, $k = 1, \ldots, n_2$ and $l = 1, \ldots, n_3$. The benchmark will

Algorithm 1 1D Benchmark

for $i = 1, \ldots, q$ **do**
  $C_i = comp\_mult(A, B_i)$
  for $k = 1, \ldots, n_2$ **do**
    $D_k = C_i(:, k)$
    $F_k = \texttt{FFT}(D_k)$
    if do_inverse **then**
      $G_k = \texttt{IFFT}(F_k)$
      $H(:, k) = G_k$
    end if
  end for
  if do_inverse **then**
    $J_i = comp\_div(H_i, B_i)$
    $abs\_err = \|A - J_i\|_2$
  end if
end for

take the form of Algorithm 3, where $comp\_mult(A, B_i)$ is defined to be component-wise multiplication of A with B_i; $comp\_div(H_i, B_i)$ is defined to be component-wise division of H_i by B_i; `FFT` is the discrete Fast Fourier Transform and `IFFT` is the discrete inverse Fast Fourier Transform. This benchmark is designed to imitate some of the workload done in CCP_PETMR's SIRF code.

Algorithm 2 2D Benchmark

for $i = 1, \ldots, q$ **do**
  $C_i = comp\_mult(A, B_i)$
  for $l = 1, \ldots, n_3$ **do**
    $D_l = C_i(:, :, l)$
    $F_l = \texttt{FFT}(D_l)$
    if do_inverse **then**
      $G_l = \texttt{IFFT}(F_l)$
      $H(:, l) = G_l$
    end if
  end for
  if do_inverse **then**
    $J_i = comp\_div(H_i, B_i)$
    $abs\_err = \|A - J_i\|_2$
  end if
end for

4 3D Benchmark

Let A be a 3D array with dimensions $n_1 \times n_2 \times n_3$ and there be q 3D arrays B_i that are the same size as A and satisfy $\sum_i^q [B_i(j, k, l)]^2 = 1$ for each $j = 1, \ldots, n_1$, $k = 1, \ldots, n_2$ and $l = 1, \ldots, n_3$. The benchmark will take the form of Algorithm 3, where $comp\_mult(A, B_i)$ is defined to be component-wise multiplication of A with B_i; $comp\_div(H_i, B_i)$ is defined to be component-wise division of H_i by B_i; `FFT` is the discrete Fast Fourier Transform

and `IFFT` is the discrete inverse Fast Fourier Transform.

Algorithm 3 2D Benchmark

for $i = 1, \ldots, q$ **do**
  $C_i = comp\_mult(A, B_i)$
  $F_i = \texttt{FFT}(C_i)$
  if do_inverse **then**
    $H_i = \texttt{IFFT}(F_i)$
    $J_i = comp\_div(H_i, B_i)$
    $abs\_err = \|A - J_i\|_2$
  end if
end for

5 FFT Libraries

We will compare the libraries listed in Table 1. The datatypes listed are real (R), complex (C) and half-complex (H). The column "Dimensions" indicates the dimensions for which interfaces are provided. Lower dimensions can be input by calling the interface for higher dimensions and setting the dimension size to 1 for the additional dimensions.

Library	Data types	Dimensions	Valid n	Parallelism	License	Citation
FFTE	R → H C → C H → R	2,3 1,2,3 2,3	$2^a \times 3^b \times 5^c$	OpenMP, MPI, CUDA	Open source	[4]
FFTW	R → H C → C H → R	Any Any Any	Any but optimised for $2^a \times 3^b \times 5^c \times 7^d \times 11^e \times 13^f$ with $e + f = 0$ or 1	Multithreading, MPI	GPL v3	[1]
MKL	R → H C → C H → R	Any Any Any	Any	Multithreading, MPI	Intel Simplified Software License	[2]
P3DFFT	R → H H → R	3 3	Any	OpenMP, MPI	GPL v3	[3]
P3DFFT++	R → H C → C H → R	1,3 1,3 1,3	Any	MPI	GPL v3	[3]

Table 1: Libraries being benchmarked. For "valid n", the values a, b, c, d, e and f are all assumed to the non-negative integers.

References

[1] M. Frigo and S. G. Johnson, *FFTW*. `http://www.fftw.org/`, 2018.

[2] Intel, *MKL: Math Kernel Library*. `https://software.intel.com/en-us/mkl`, 2018.

[3] D. Pekurovsky, *P3DFFT: a framework for parallel computations of Fourier transforms in three dimensions*, SIAM Journal on Scientific Computing, 34 (2012), pp. 192–209. `https://www.p3dfft.net/`.

[4] D. Takahashi, *FFTE: A Fast Fourier Transform Package*. http://www.ffte.jp/, 2014.