

1 Mapping FFT computations to processor networks

1.1 Linear arrays

1.2 Two-dimensional meshes

1.3 Three-dimensional meshes

1.4 Binary cubes

The FFT performs butterfly computations on pairs of data elements that differ in a single bit. The computations start from the most significant bit in the index space and proceed monotonically to the least significant bit. In a binary cube, adjacent nodes differ in a single bit in their addresses. Thus, mapping data to nodes by a binary encoding of the indices is suitable for the FFT. We assume such a mapping in the discussion below. In [4] we show that the same communication complexity can indeed be obtained for binary-reflected Gray code data encodings, even though the elements in a butterfly computation are no longer nearest neighbors, but at a distance of two apart. For Gray coded data, a code conversion is performed concurrently with the butterfly network emulation without an increase in the communication cost.

1.4.1 Computing the FFT without explicit permutations

With a binary data encoding, communication is required in a single cube dimension for each butterfly stage, unless it is a local stage. If *all-port* communication is possible, i.e., concurrent communication on all channels, then pipelining of the communication for successive FFT stages can be used. Figure 1 illustrates the idea for a radix-2 FFT on a 3-cube. In the first communication one data element per node is exchanged in the most significant cube dimension for normal order input. After the splitting formulas have been evaluated for these data items, they are ready for the second stage of the FFT. In the second communication, the data in the first memory locations in all nodes are exchanged in the second most significant cube dimension, while the second memory locations are exchanged in the most significant cube dimension. From the third communication stage all communication channels are used in every exchange for the 3-cube, until all local memory locations have been touched, at which point the shut-down of the communications pipeline starts.

Time Step	Memory location	Processor							
		0	1	2	3	4	5	6	7
0	0	2	2	2	2	2	2	2	2
	1	-	-	-	-	-	-	-	-
	2	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
1	0	1	1	1	1	1	1	1	1
	1	2	2	2	2	2	2	2	2
	2	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
2	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
3	0	-	-	-	-	-	-	-	-
	1	0	0	0	0	0	0	0	0
	2	1	1	1	1	1	1	1	1
	3	2	2	2	2	2	2	2	2
	4	-	-	-	-	-	-	-	-

Figure 1: The first four steps of an in-place pipelined radix-2 FFT.

The idea of pipelining the communications for the FFT computations can also be understood by observing that for a consecutive data allocation over 2^n nodes, using a DIT FFT the first n stages can be viewed as $\frac{P}{N}$ distinct FFTs, each with one data point per node. Each such FFT requires communication in the dimensions $n - 1, n - 2, \dots, 1, 0$, one for each stage of the FFT. Hence, when the first stage of the first FFT is computed, dimension $n - 1$ is free to be used for the computations of the next FFT.

The radix-2, pipelined FFT requires $\frac{P}{N} + n - 1$ element transfers in sequence for an axis distributed over n cube dimensions, and with $\frac{P}{N}$ elements per processor. If $\frac{P}{N} \gg n$, then the communication system is fully utilized effectively all the time. It is easily verified that the communication complexity holds for both the consecutive and the cyclic data allocation.

The idea of pipelining the communication and computations for successive FFT stages can be applied to radix- 2^r FFT for $r > 1$, but the pipelined radix-2 FFT offers better overall efficiency [5]

1.4.2 Computing the FFT with explicit permutations

Bi-section

The idea of pipelining can be used in combination with bi-section to fully utilize the communication system. Figure 2 shows the first few exchanges for a pipelined bi-section algorithm.

Time Step	Memory location	Processor							
		0	1	2	3	4	5	6	7
0	0	-	-	-	-	2	2	2	2
	1	2	2	2	2	-	-	-	-
	2	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-
1	0	-	-	1	1	-	-	1	1
	1	1	1	-	-	1	1	-	-
	2	-	-	-	-	2	2	2	2
	3	2	2	2	2	-	-	-	-
	4	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-
2	0	-	0	-	0	-	0	-	0
	1	0	-	0	-	0	-	0	-
	2	-	-	1	1	-	-	1	1
	3	1	1	-	-	1	1	-	-
	4	-	-	-	-	2	2	2	2
	5	2	2	2	2	-	-	-	-
3	0	-	-	-	-	-	-	-	-
	1	-	-	-	-	-	-	-	-
	2	-	0	-	0	-	0	-	0
	3	0	-	0	-	0	-	0	-
	4	-	-	1	1	-	-	1	1
	5	1	1	-	-	1	1	-	-

Figure 2: The first four steps of a pipelined bi-section based radix-2 FFT.

Input order	Algorithm	Consecutive	Cyclic
Normal	No explicit perm.	$\frac{P}{N} + n - 1$	$\frac{P}{N} + n - 1$
	Bi-section	$\frac{P}{N} + n - 1$	$\frac{P}{2N} + n - 1$
	Multi-section	$\frac{P}{N} + (\lceil \frac{n}{r} \rceil - 1)2^{r-1}$	$\frac{P}{2N} + (\lceil \frac{n}{r} \rceil - 1)2^{r-1}$
Bit-reversed	No explicit perm.	$\frac{P}{N} + n - 1$	$\frac{P}{N} + n - 1$
	Bi-section	$\frac{P}{2N} + n - 1$	$\frac{P}{N} + n - 1$
	Multi-section	$\frac{P}{2N} + (\lceil \frac{n}{r} \rceil - 1)2^{r-1}$	$\frac{P}{N} + (\lceil \frac{n}{r} \rceil - 1)2^{r-1}$

Table 1: Communication requirements for unordered transforms with *all-port* communication.

The number of elements exchanged in sequence for the pipelined bi-section algorithm for a cyclic data allocation and normal input order is $\frac{P}{2N} + n - 1$. Thus, the bi-section algorithm yields approximately a factor of two improvement in the data transfer time compared to the algorithm without explicit permutations. However, for a consecutive data allocation and *all-port* communication, the bi-section algorithm no longer improves upon the data transfer time over the algorithm with no explicit permutations. The reason for this fact is that for the consecutive data allocation communication in the most significant dimension must be performed twice. With *all-port* communication, $2\frac{P}{2N} + n - 1$ element transfers in sequence is required for the bi-section algorithm, ignoring a possible overlap between the second exchange in the most significant dimension and the pipeline filling time. Thus, with consecutive data allocation and normal input order, the bi-section algorithm requires the same time for data transfer as the algorithm without explicit permutations.

Multi-section

The communication for the 2^r -sectioning on blocks of r different nodal dimensions can be pipelined. On a binary cube, each 2^r -sectioning requires a time of 2^{r-1} [3]. The number of element transfers in sequence for a pipelined 2^r -sectioning algorithm is $2^{r-1}\lceil \frac{n}{r} \rceil + (\frac{P}{N2^r} - 1)2^{r-1} = \frac{P}{2N} + (\lceil \frac{n}{r} \rceil - 1)2^{r-1}$ for cyclic data allocation. Note that for *all-port* communication, if $p - n \geq n$, then $r = n$ is optimal, just as in the node limited communication model. However, note also that if $p - n < n$, then $r = 2$ yields the smallest data transfer time, i.e., a pipelined 4-section algorithm is optimal. A bi-section algorithm is only slightly inferior. For small values of r the variance in communication efficiency is small, and the choice of r is largely determined by the efficiency in evaluating the splitting formulas.

With a consecutive data allocation the r most significant processor dimensions must be used twice, and a pipelined multi-section algorithm requires $\frac{P}{N} + (\lceil \frac{n}{r} \rceil - 1)2^{r-1}$ element transfers in sequence, essentially the same as for the algorithm with no explicit permutations. The data transfer requirements for the different algorithms are summarized in Table 1.

In conclusion we note that for the *all-port* communication model, the algorithm without explicit permutations yields the same communication efficiency as the bi- and multi-section algorithms for half of the combinations of data allocations and input orderings, while for the other half it yields a factor of two higher data transfer times.

With respect to implementations, it is important to note that the bi- and multi-section algorithms require indirect addressing, which may be slower than direct addressing. If indirect addressing indeed is slower than direct addressing, then the algorithm without explicit permutations is actually faster than bi- or multi-section algorithms for normal input order and consecutive data allocation and bit-reversed input order and cyclic data allocation, and most likely only marginally slower for other cases. For the Connection Machine system CM-2 the algorithm without explicit permutations is always faster than bi- or multi-sectioning [5]. However, on the CM-5, multi-sectioning is faster.

2 Ordered transforms

2.1 Reordering of the output

2.2 Integrated reordering

3 Fourier Transforms of real sequences

Let x_j , $0 \leq j < 2^p$ be a real sequence of length $P = 2^p$. Let its transform be X_k , $0 \leq k < 2^p$. The sequence X_k is *conjugate symmetric*, i.e., $X_k = \overline{X_{N-k}}$. Hence, it suffices to compute X_k , $0 \leq k \leq \frac{N}{2}$. The component X_0 is clearly real, and so is the component $X_{N/2}$. Hence, even though $\frac{N}{2} + 1$ complex points are computed, two of them are real and only N real locations are needed to store the unique part of the transform. The transform of a real sequence can be computed in-place. Also, as we will show next, the number of arithmetic operations for the transform of a real sequence is approximately half of that for a complex sequence. We will consider a few different alternatives for computing transforms of real sequences.

3.1 Transformation of pairs of sequences

A *pair* of real sequences, y_k and z_k , can be transformed using a single complex transform by defining the complex sequence x_k , where $x_k = y_k + iz_k$ and computing its complex transform X_m . Since the Fourier transform is linear,

$$X_m = Y_m + iZ_m$$

where Y_m and Z_m are the Fourier transforms of y_k and z_k , respectively. Since Y_m and Z_m are complex they do not correspond to the real and imaginary parts of X_m . But,

$$X_{N-m} = Y_{N-m} + iZ_{N-m} = \overline{Y_m} + i\overline{Z_m}$$

since, Y_m and Z_m are conjugate symmetric, since y_k and z_k are real. Therefore,

$$\overline{X_{N-m}} = Y_m - iZ_m$$

and,

$$Y_m = \frac{1}{2}(X_m + \overline{X_{N-m}}), \quad m = 0, 1, \dots, N/2$$

$$Z_m = \frac{1}{2i}(X_m - \overline{X_{N-m}}), \quad m = 0, 1, \dots, N/2$$

Therefore, the transform of a pair of real sequences of length N can be computed from a single complex transform of length N . First, form $x_k = y_k + iz_k$, next compute X_m , and finally compute Y_m and Z_m . This technique is particularly useful for multidimensional transforms on real data.

The postprocessing to recover the two desired transforms requires $4N$ additions/subtractions and a scaling by a factor of two. Thus, this method for computing transforms of real sequences requires approximately $\frac{N}{2}(5 \log_2 N + 8)$ arithmetic operations. In addition, it requires an *index reversal* in order that values with index k and $N - k$ be combined. This index reversal may in fact double the communication time in a multi-processor, as we will see later.

Remark: Note that since the Fourier Transform is a linear operation, if the same operation is applied to both transformed sequences, followed by an inverse transform, then explicit computation of Y_m and Z_m is not necessary. There is a large class of applications, including the solution of certain partial differential equations, to which this case applies. For example, an approximate derivative of the sequences y_k and z_k can be computed by multiplying X_m by im and performing the inverse FFT.

3.2 Transformation of a single sequence

3.2.1 Edson's algorithm

Recall the Cooley–Tukey DIT splitting formula

$$X_m = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j} + \omega_N^m \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}, \quad m = 0, 1, 2, \dots, N/2 - 1$$

$$X_{m+N/2} = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j} - \omega_N^m \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}, \quad m = 0, 1, 2, \dots, N/2 - 1$$

But, for x a real sequence X_m is conjugate symmetric, i.e., $X_m = \overline{X_{N-m}}$. The second half can be obtained from the first half with no computation. Thus, the second formula yields

$$\overline{X_{N/2-m}} = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j} - \omega_N^m \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}, \quad m = 0, 1, 2, \dots, N/2 - 1$$

But, it is now apparent that the first and the second formula computes the same set of values of X_m , and we arrive at the following splitting formulas for a real sequence

$$X_m = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j} + \omega_N^m \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}, \quad m = 0, 1, 2, \dots, N/4$$

$$\overline{X_{N/2-m}} = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j} - \omega_N^m \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}, \quad m = 0, 1, 2, \dots, N/4 - 1$$

Let $Y_m = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j}$, $0 \leq m \leq N/4$ and $Z_m = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}$, $0 \leq m \leq N/4$.

Then, the two expressions can be rewritten as

$$X_m = Y_m + \omega_N^m Z_m, \quad m = 0, 1, 2, \dots, N/4$$

$$X_{N/2-m} = \overline{Y_m} - \omega_N^{-m} \overline{Z_m}, \quad m = 0, 1, 2, \dots, N/4 - 1$$

Note that the first expression is evaluated for one more index than the second expression. The butterfly computations for $m = 0$ and $m = N/4$ are special. For $m = 0$, the first equation simply yields $X_0 = Y_0 + Z_0$ with all variables being real. Similarly, the second equation yields $X_{N/2} = Y_0 - Z_0$. Thus, these two values of the transform are simply the sum and the difference of a pair of real numbers. They are often stored as one complex number. For $m = N/4$, only the first equation is used. However, $Y_{N/4}$ and $Z_{N/4}$ are both real numbers. Thus, $X_{N/4} = Y_{N/4} + iZ_{N/4}$ and this value requires no computation once Y_m and Z_m are known. Note, that since Y_m and Z_m both are conjugate symmetric sequence of length $N/4$ each, Y_0 and $Y_{N/4}$ may be stored as one complex number. Similarly, Z_0 and $Z_{N/4}$ may also be stored as one complex number. The butterfly operating on the first pair of complex numbers from the sequences Y_m and Z_m is a special butterfly requiring a total of two real arithmetic operations. The number of real arithmetic operations per splitting formula is $(\frac{N}{4}10 + 2)$. The splitting is repeated $\log_2 N$ times.

This real-to-complex FFT is due to Edson [1]. It requires a special butterfly in each stage, and more importantly with respect to parallel computation, an index reversal in each stage.

3.2.2 A postprocessing algorithm

The idea for this algorithm is to use a standard complex-to-complex FFT with postprocessing. We derive this real-to-complex FFT by starting with the splitting formulas

$$X_m = Y_m + \omega_N^m Z_m, \quad m = 0, 1, 2, \dots, N/4$$

$$X_{N/2-m} = \bar{Y}_m - \omega_N^{-m} \bar{Z}_m, \quad m = 0, 1, 2, \dots, N/4 - 1$$

But, instead of using the formulas recursively we compute Y_m , $0 \leq m \leq N/4$ and Z_m , $0 \leq m \leq N/4$ by using a complex-to-complex FFT as follows.

Let $v_j = x_{2j} + x_{2j+1}$, $0 \leq j < N/2$, and let

$$V_m = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} v_j, \quad m = 0, 1, 2, \dots, N/2 - 1$$

Then,

$$V_m = \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j} + i \sum_{j=0}^{N/2-1} \omega_{N/2}^{jm} x_{2j+1}, \quad m = 0, 1, 2, \dots, N/2 - 1$$

or

$$V_m = Y_m + iZ_m$$

But, the sequences Y_m and Z_m are conjugate symmetric. Hence,

$$\overline{V_{N/2-m}} = Y_m - iZ_m, \quad m = 0, 1, 2, \dots, N/4$$

and

$$Y_m = \frac{V_m + \overline{V_{N/2-m}}}{2}, \quad m = 0, 1, 2, \dots, N/4$$

$$Z_m = \frac{V_m - \overline{V_{N/2-m}}}{2i}, \quad m = 0, 1, 2, \dots, N/4$$

Note that

$$Y_0 = \text{Re}\{V_0\} \quad \text{and} \quad Z_0 = \text{Im}\{V_0\}$$

and

$$Y_{N/4} = \text{Re}\{V_{N/4}\} \quad \text{and} \quad Z_{N/4} = \text{Im}\{V_{N/4}\}$$

Now, to compute X_m we note that

$$X_0 = Y_0 + Z_0 = \text{Re}\{V_0\} + \text{Im}\{V_0\}$$

$$X_{N/2} = \text{Re}\{V_0\} - \text{Im}\{V_0\}$$

and

$$\text{Re}\{X_{N/4}\} = \text{Re}\{V_{N/4}\} \quad \text{Im}\{X_{N/4}\} = \text{Im}\{V_{N/4}\}$$

The remaining values of X_m are obtained by substituting the expressions for Y_m and Z_m into the expressions for X_m . This yields

$$X_m = \frac{1}{2}[(V_m + \bar{V}_{N/2-m}) - i\omega_N^m(V_m - \bar{V}_{N/2-m})], \quad m = 1, 2, \dots, N/4 - 1$$

$$\bar{X}_{N/2-m} = \frac{1}{2}[(V_m + \bar{V}_{N/2-m}) + i\omega_N^m(V_m - \bar{V}_{N/2-m})], \quad m = 1, 2, \dots, N/4 - 1$$

Thus, the real-to-complex Fourier transform can be computed by

- considering pairs of successive real numbers as a complex number,
- computing the Fourier transform on this complex sequence using an FFT for a sequence of length $N/2$,
- performing a postprocessing step to recover the transforms on even and odd index real variables
- performing a final butterfly stage defined by the splitting formulas for real sequences.

Input	V_m	Y_m, Z_m	$\overline{Y}_{N/2-m}, \overline{Z}_{N/2-m}$	X_m	m	$N/2 - m$
0	0_r	Y_0		0_r	0	8
1	0_i	Z_0		8_r	0	8
2	4_r	$Y_{N/4}$		4_r	4	4
3	4_i	$Z_{N/4}$		4_i	4	4
4	2_r	\surd		2_r	2	6
5	2_i	\surd		2_i	2	6
6	6_r		\surd	6_r	2	6
7	6_i		\surd	6_i	2	6
8	1_r	\top		1_r	1	7
9	1_i	\top		1_i	1	7
10	5_r		\perp	5_r	3	5
11	5_i		\perp	5_i	3	5
12	3_r	\perp		3_r	3	5
13	3_i	\perp		3_i	3	5
14	7_r		\top	7_r	1	7
15	7_i		\top	7_i	1	7

Table 2: Data combining for a postprocessing real-to-complex FFT on 16 elements.

Only the postprocessing step requires an index reversal in this algorithm.

Table 2 illustrates the computations associated with different memory locations for real input data in normal order. The result of applying a complex-to-complex FFT is showed in the column labeled V_m . The computations of Y_m and Z_m for evaluation of X_m are shown in the subsequent two columns. The symbols show pairs of data that are combined in the index reversal.

Note that with data in normal order and a complex-to-complex FFT producing a bit-reversed order, the postprocessing combines values within subsets of the address space through *bit-inversion*. For instance, the odd values of X_m are computed by pairing data through bit-inversion on all bits but the most significant bit. The next quarter sequence requires bit-inversion on all but the two most significant bits, etc.

4 Transformation of a conjugate symmetric sequence

The transform of a conjugate symmetric sequence X_m is a real sequence x_k . This transform is also known as a complex-to-real FFT. Thus, we wish to compute

$$x_k = \sum_{m=0}^{N-1} w_N^{-mk} X_m, \quad k = 0, 1, \dots, N-1$$

where only the values $X_m, m = 0, 1, 2, \dots, N/2$ are available. For the computation we split the

evaluation of x_k

$$x_k = \sum_{m=0}^{N/2-1} w_N^{-km} X_m + \sum_{m=N/2}^{N-1} w_N^{-km} X_m, \quad k = 0, 1, \dots, N-1$$

or

$$x_k = \sum_{m=0}^{N/2-1} w_N^{-km} X_m + \sum_{m=0}^{N/2-1} w_N^{km} (-1)^k X_{m+N/2}, \quad m = 0, 1, \dots, N-1$$

or

$$x_k = \sum_{m=0}^{N/2-1} w_N^{-km} X_m + \sum_{m=0}^{N/2-1} w_N^{km} (-1)^k \overline{X_{N/2-m}}, \quad m = 0, 1, \dots, N-1$$

In the latter expression only known values of X_m are required. Separating the expressions for even and odd k we have

$$x_{2k} = \sum_{m=0}^{N/2-1} w_{N/2}^{-km} (X_m + \overline{X_{N/2-m}}), \quad m = 0, 1, \dots, N/2-1$$

and

$$x_{2k+1} = \sum_{m=0}^{N/2-1} w_{N/2}^{-km} w_N^{-m} (X_m - \overline{X_{N/2-m}}), \quad k = 0, 1, \dots, N/2-1$$

But, $X_m + \overline{X_{N/2-m}} = 2Y_m$ and $w_N^{-m} (X_m - \overline{X_{N/2-m}}) = 2Z_m$. Ignoring the scaling factor the correctness is clear. It is easily seen that both Y_m and Z_m are conjugate symmetric. Hence, it suffices to evaluate them for the index range $m = 0, 1, \dots, N/4$.

Now, forming the complex sequence $v_k = x_{2k} + ix_{2k+1}$ we get

$$v_k = \sum_{m=0}^{N/2-1} w_{N/2}^{km} [(X_m + \overline{X_{N/2-m}}) + iw_N^m (X_m - \overline{X_{N/2-m}})] \quad k = 0, 1, \dots, N/2-1$$

Let $V_m = (X_m + \overline{X_{N/2-m}}) + iw_N^m (X_m - \overline{X_{N/2-m}})$. Then, the evaluation of V_m can be performed as

$$V_m = (X_m + \overline{X_{N/2-m}}) + iw_N^m (X_m - \overline{X_{N/2-m}}), \quad m = 0, 1, 2, \dots, N/4$$

and

$$\overline{V_{N/2-m}} = (X_m + \overline{X_{N/2-m}}) - iw_N^m (X_m - \overline{X_{N/2-m}}), \quad m = 0, 1, 2, \dots, N/4-1$$

Once the values V_m are computed, then a CCFFT is applied to it, and the sequence v_k obtained.

Hence, the computations for the CRFFT are the inverse of the RCFFT.

5 Communication requirements for transformation of real sequences and conjugate symmetric sequences

The pair of real numbers in a complex number is typically assigned to successive memory locations. For one-dimensional arrays treating a real sequence as a complex sequence requires no data motion, if there is an equivalence instruction between real and complex arrays. Similarly, for multi-dimensional arrays no data motion may be required if the problem axis has stride one.

However, if the stride is higher than one, then the conversion between a real array and a correctly “complexified” array requires data motion in the form of a shuffle.

The index reversal takes several forms depending upon the input and output orderings.

6 Applications of the FFT

6.1 Convolution

The discrete convolution x of two functions h and y is defined as

$$x(j) = \sum_{m=-M_1}^{M_2} h(m)y(j-m), \quad j = \{0, 1, \dots, P-1\}$$

in one dimension. In filtering applications, the array arguments usually represents time, in which case $M_1 = 0$ is common. Negative values of M_1 would include future values of the “input” signal y for the current output signal x . Furthermore, M_2 is usually considerably smaller than P , since the response time of most filters are considerably shorter than the signal duration. The number of arithmetic operations for $P \geq M_2$ and $M_1 = 0$ is $N(2M_2 - 1) - (M_2 - 1)^2$.

From the *Discrete convolution theorem*, the convolution corresponds to a pointwise multiplication in the Fourier domain.

Let $X(k) = \sum_{j=0}^{P-1} \omega_P^{jk} x(j)$. Then,

$$X(k) = \sum_{j=0}^{P-1} \omega_P x^{jk} \sum_{m=0}^{M_2} h(m)y(j-m) = \sum_{m=0}^{M_2} \omega_P^{mk} h(m) \sum_{j=0}^{P-1} \omega_P^{(j-m)k} y(j-m)$$

But, since y is assumed periodic we have

$$X(k) = H(k)Y(k)$$

where $H(k)$ and $Y(k)$ are the Fourier transforms of h and y respectively. Hence, the convolution of two functions can also be computed through a fast Fourier transform of h and y , a

componentwise complex multiplication, followed by an inverse transform of X to find x . Thus, computing the convolution in this way require approximately $15N \log_2 N$ operations. Using the FFT for convolution computations is advantageous with respect to arithmetic operations when $2M_2 > 15 \log_2 N$.

6.2 Fast Poisson solvers

Another very important application of the FFT is in the solution of Poisson’s equation. Consider Poisson’s equation in a rectangle. Consider first the equation

$$\frac{\partial^2 u}{\partial x^2} = f$$

and its discrete analogue with Dirichlet boundary conditions.

$$u_{j-1} - 2u_j + u_{j+1} = h^2 f_j = f'_j, \quad j = \{0, 1, \dots, P-1\}$$

where $u_1 = u_P = 0$. Taking the discrete Fourier Transform of this equation yields

$$\sum_{i=0}^{P-1} \omega_P^{jk} u_{j-1} - 2 \sum_{i=0}^{P-1} \omega_P^{jk} u_j + \sum_{i=0}^{P-1} \omega_P^{jk} u_{j+1} = \sum_{i=0}^{P-1} \omega_P^{jk} f'_j$$

or

$$\omega_P^k U(k) - 2U(k) + \omega_P^{-k} U(k) = F(k)$$

But,

$$\omega_P^k + \omega_P^{-k} = 2 \cos\left(\frac{2\pi k}{P}\right)$$

Thus, in the Fourier domain

$$U(k) = \frac{F(k)}{2(\cos(\frac{2\pi k}{P}) - 1)}$$

Hence, instead of solving the tridiagonal system of equations that result from the discretization we can use the FFT to compute $F(k)$, then obtain $U(k)$ through a componentwise division, followed by an inverse Fourier transform to compute u . Though elegant this procedure is computationally expensive compared to the use of Gaussian elimination or odd–even cyclic reduction for the solution of the tridiagonal system.

Note that Poisson’s equation in one dimension can be viewed as a convolution operation. For our second order centered difference stencil, the stencil, or convolution kernel only has three points. Hence it should not be a surprise that direct evaluation of the convolution is faster than using the FFT. However, as the order of the difference approximation increases, the FFT approach eventually becomes competitive.

In two and three dimensions the observation that taking the Fourier transform of the set of equations result in pointwise operations in the Fourier domain is advantageous with respect to the number of arithmetic operations required. Consider

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f$$

discretized by second order centered differences as

$$u_{j,m-1} + u_{j,m+1} + u_{j-1,m} + u_{j+1,m} - 4u_{j,m} = h^2 f_{j,m}, \quad j = \{0, 1, \dots, P-1\}, m = \{0, 1, \dots, R-1\}$$

with Dirichlet boundary conditions. Taking the Fourier transform in both dimensions yields

$$\left(2 \cos\left(\frac{2\pi k}{P}\right) + 2 \cos\left(\frac{2\pi l}{R}\right) - 4\right)U(k, l) = F(k, l)$$

or

$$U(k, l) = \frac{F(k, l)}{2\left(\cos\left(\frac{2\pi k}{P}\right) + \cos\left(\frac{2\pi l}{R}\right) - 2\right)}$$

Now, the operation count of using the FFT, $\sim 10PR \log_2(PR)$, compares favorably with using a direct solver like Gaussian elimination, since the bandwidth of the matrix in two dimensions is P or R depending upon the ordering of the grid points. With a bandwidth of P , the arithmetic count for a direct method is $O(P^3R)$. In three dimensions, the FFT compares even more favorably with direct methods.

The method can still be improved. If instead of a Fourier transform along both axis of the two-dimensional domain, only a single axis transform is taken, then a system of independent tridiagonal systems of equations results. For instance, if a transform is made on the P -axis with index j , then the result is

$$U_{k,m-1} + U_{k,m+1} + 2\left(\cos\left(\frac{2\pi k}{P}\right) - 2\right)U_{k,m} = F_{k,m}, \quad k = \{0, 1, \dots, P-1\}, m = \{0, 1, \dots, R-1\}$$

For a given k , this equation represent a tridiagonal system of equations with constant coefficients. The size of each such system is R and the number of such systems is P . Hence, the Fourier transform along one of the axis can be replaced by the solution of a number of independent tridiagonal systems. A savings of about $10PR \log_2 R$ is possible.

A further reduction in the operation count is possible by using a few cyclic reduction steps prior to the use of Fourier transformation and tridiagonal system solution. This block cyclic reduction algorithm yields an operation count of $O(PR \log_2 \log_2 P)$. Details for a Dirichlet, Neumann and mixed boundary conditions can be found in [2, 6, 7].

References

- [1] G.D. Bergland. A fast Fourier transform for real-valued series. *Communications of the ACM*, 11:703–710, 1968.
- [2] Billy L. Buzbee, Gene H. Golub, and C W. Nielson. On direct methods for solving Poisson’s equations. *SIAM J. Numer. Anal.*, 7(4):627–656, December 1970.
- [3] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.
- [4] S. Lennart Johnsson and Ching-Tien Ho. Boolean cube emulation of butterfly networks encoded by Gray code. *Journal of Parallel and Distributed Computing*, 20(3):261–279, 1994. Department of Computer Science, Yale University, Technical Report, YALEU/DCS/RR-764, February, 1990.
- [5] S. Lennart Johnsson, Michel Jacquemin, and Robert L. Krawitz. Communication efficient multi-processor FFT. *Journal of Computational Physics*, 102(2):381–397, October 1992.
- [6] Paul N. Swarztrauber. The methods of cyclic reduction, Fourier analysis, and the FACR algorithm for the discrete solution of Poisson’s equation on a rectangle. *SIAM Review*, 19:490–501, 1977.
- [7] Clive Temperton. On the FACR(1) algorithm for the discrete Poisson equation. *J. of Computational Physics*, 34:314–329, 1980.