HPCERC2000-019
October 2000

High Performance Computing
Education & Research Center

HPCERC TECHNICAL REPORT

A Novel Directional Approach for the Scalable, Accurate and Efficient Computation of Two-Dimensional Discrete Fourier Transforms

Authors

Marios S. Pattichis and Ruhai Zhou

The University of New Mexico
Dept. of Electrical & Computer Engineering

The University of New Mexico
High Performance Computing, Education & Research Center
Albuquerque, NM 87131
Phone # 505.277.8249 Fax # 505.277.8235
Web Site: www.hpcerc.unm.edu

The University of New Mexico

Disclaimer

The High Performance Computing, Education & Research Center (HPCERC) provides a focus for high performance computing and communication at the University of New Mexico (UNM). HPCERC is committed to innovative research in computational and computer science with emphasis on both algorithm development and application. As part of this commitment, HPCERC sponsors this technical report series. The technical reports are subject to internal review by the HPCERC. However, the material, as presented, does not necessarily reflect any position of the HPCERC. Further, neither UNM, nor the AHPCC, makes any warranty or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information contained in this report.

Frank L. Gilfeather, Executive Director, HPCERC

Brian T. Smith, Chief Scientist, HPCERC

John S. Sobolewski, Chief Technologist, HPCERC

Ernest D. Herrera, Associate Director, HPCERC

Susan Atlas, Associate Director, AHPCC Computer Systems Research

Bob Ballance, Associate Director, AHPCC Science & Engineering Research

Technical Report AHPCC01

A Novel Directional Approach for the Scalable, Accurate and Efficient Computation of Two Dimensional Discrete Fourier Transforms

Marios S. Pattichis[1] Ruhai Zhou[2]

Albuquerque High Performance Computing Center
1601 Central Avenue, NE
The University of New Mexico
Albuquerque, NM 87131 USA

[1]**pattichis@eece.unm.edu**
[2]**rzhou@ahpcc.unm.edu**

AHPCC Technical Reports

This technical report series allows affiliates of the Albuquerque High Performance Computing Center at The University of New Mexico to publish detailed and recent research results in a timely manner. It is *not* intended that these technical reports duplicate outside publications. However, due to the time lag in publishing results in formal, peer reviewed venues, many of these technical reports will be submitted for review and publication elsewhere. In such cases, it is intended that the technical reports will contain additional details and results that cannot be included elsewhere due to space limitations.

In addition to technical reports pertaining to research conducted within the Center, the technical report series may also be used to publish "pedagogical" results and methods. The University of New Mexico has a strong tradition and committment to high-quality teaching and teaching methods. Many of our faculty are actively engaged in developing new pedagogical techniques, including the use of multi-media and Web-based tools for instructional purposes. We believe that it is equally important to make these results available to the academic and education community.

For more information about this technical report series, please contact Marlene Wallace `wallacem@ahpcc.unm.edu`.

Publication History

A closely related paper titled *Novel Algorithms for Accurate, Efficient, and Parallel Computation of Multidimensional, Regional Discrete Fourier Transforms* has been presented at the 10th Mediterranean Electrotechnical Conference held in May, 2000, in Limassol, Cyprus, and was published in the proceedings of the conference.

Abstract

New algorithms for computing the Discrete Fourier Transform (DFT) spectra along different directions are derived and implemented. For computing the DFT spectrum along any given direction (N DFT frequencies), a new algorithm is presented that requires $N(N-1)$ additions and a single 1-D FFT. As expected, the new unidirectional FFT algorithm is found to be significantly faster than standard 2-D FFT algorithms that compute the entire spectrum (all results are compared against `FFTW` and `FFTPACK`). A scalable extension of the unidirectional algorithm for computing the entire DFT spectrum is also derived and implemented. The three most promising features of the new algorithm are that: (i) computation scales nearly linearly with the number of DFT frequencies computed, (ii) the algorithm uses a reduced number of multiplications (yet uses more additions), and (iii) it is proven to be more accurate.

To achieve scalability, the 2-D DFT spectrum is subsampled into the number of required components. Depending on the application, the algorithm architecture allows very fine control over the number of components, which are computed independently, without the need to share results between the processes (until all computation has been completed). It is expected that this feature of the serial implementation presented here will lead to an efficient, parallel implementation of the algorithm. The fundamental advantage of such scalability, is that a subsampled version of the DFT spectrum should be adequate in numerous applications.

The reduction in the number of multiplications is due to the reduced number of 1-D FFTs used in computing the DFT spectrum. For computing the 2-D DFT spectrum using the new directional approach, $3N/2$ FFTs are required for an $N \times N$ image. In contrast, the standard approach requires $2N$ FFTs. Unfortunately, the new method requires some extra additions. A novel, scalable algorithm is derived that reduces the number of extra additions to be $O(N^2 \log_2 N)$. It is shown that with the new integer addition algorithm, for integer data, the new directional DFT algorithm computes the entire 2-D spectrum in time that is comparable to that of `FFTW` and `FFTPACK`. Nevertheless, it is expected that an ongoing SIMD implementation of the addition algorithm, will allow significant computational improvements for both integer and floating-point data. The improved accuracy of the directional DFT algorithm is due to the fact that each DFT frequency is computed using exact integer additions, followed by a single floating-point FFT. In contrast, the standard 2-D decomposition approach requires $N + 1$ floating-point FFTs to compute each column of frequencies of the DFT spectrum. Hence, for the directional FFT algorithm, the improved accuracy is due to the significant reduction in the number of floating point operations.

Keywords:

Table of Contents

List of Figures

List of Tables

1 Introduction

Multi-dimensional Discrete Fourier Transforms (DFTs) have found many important applications in scientific computing. Usually, whenever used, DFT computation accounts for the most significant portion of the computation time. To help reduce the significant amount of time involved in DFT computation, we consider a scalable computational framework, where the DFT spectrum is computed along independent directions. To motivate this approach, we will first examine some of the limits inherent in deriving algorithms based on the standard tensor decomposition of the algorithm [2], [6].

Let $x(n_1, n_2, \ldots, n_m)$ denote an m-dimensional array. In this paper, we will assume that we have the same number of elements in each array dimension: $0 \leq n_i \leq N-1$. We express the m-dimensional DFT $X(k_1, k_2, \ldots, k_m)$ as:

$$X(k_1, k_2, \ldots, k_m) = \sum_{n_1=0}^{n_1=N-1} \cdots \sum_{n_m=0}^{n_m=N-1} x(n_1, n_2, \ldots, n_m) W_N^{k_1 n_1 + \cdots + k_m n_m}$$

where $W_N = \exp\left[-\sqrt{-1}\frac{2\pi}{N} \right]$. In two-dimensions, the standard tensor method for computing the 2-D DFT is:

Step 1. Compute N 1-D FFTs along each column.
Step 2. Transpose.
Step 3. Compute N 1-D FFTs along each column.
Step 4. Transpose.

Unfortunately, this standard approach exhibits some inherent limitations:

- Limitations on Scalability.
  There is very limited room for scalability in the standard algorithm. The DFT spectrum requires that Step 1 is fully completed before Step 3 can be done. Hence, no modification of the standard scheme can avoid having to compute N FFTs before any DFT results can be obtained.

- Limitations on Parallelism.
  An obvious way to parallelize the approach is to perform each FFT in Steps 1 and 3 on a different processor. A modification of the standard approach can communicate results from Step 1 to Step 3 more efficiently, but the fundamental communication limit is that all the results from Step 1 will still have to be communicated to Step 3.

- Limitations on Accuracy
  For each DFT frequency, $N+1$ floating-point FFTs are used (N from step 1, and 1 from step 3).

The luck of scalability is the most important one. It is thus important to derive a scalable algorithm, which provides meaningful estimates of the DFT spectrum, which in an ideal situation, can be used in lieu of the entire spectrum. We can also use scalability to improve parallelism. We summarize some possibilities:

- DFT spectrum computation along lines
  In many applications, a significant portion of the signal energy can be captured by examining particular directions of the spectrum. To be precise, for the discrete frequency $\mathbf{k} = (\, k_1,\, k_2,\, \ldots,\, k_m\,)$, we associate a direction of the spectrum as given by

  $$n\mathbf{k}, \quad n = 1,\, 2,\, \ldots,\, N-1, \quad \mathbf{k} \neq \mathbf{0}.$$

  In addition, for the recently introduced FM transform [8], it was shown that a large variety of digital signals (including random signals) can be re-arranged via sample permutations so that most of the signal energy can be captured along any desirable direction. A scalable spectrum computation along independent directions can be used to compute the entire spectrum.

- Subsampled DFT spectrum computation
  For most signals, either because (i) they exhibit a degree of smoothness, or (ii) they are compactly supported, there is (i) significant DFT spectrum decay, or (ii)large DFT spectrum smoothness that bounds how fast the spectrum can change. Hence, a subsampled DFT spectrum, where a single sample is computed over independent regions would be very indicative of the entire DFT spectrum. For example, we can consider a partition of the DFT spectral plane: $(Z/N)^M = R_1 \cup R_2 \cup R_3 \cup \cdots \cup R_p$, and compute a single sample in each region

  $$X(\mathbf{k_1}),\ \mathbf{k_1} \in R_1,\, X(\mathbf{k_2}),\ \mathbf{k_2} \in R_2, \quad \ldots, \quad X(\mathbf{k_p}),\ \mathbf{k_p} \in R_p.$$

  Subsequent computation will take more samples from each region, until the entire DFT spectrum is computed. A variation of this approach using subsampled directional spectra will be investigated in this paper.

To explain the directional approach, we first note that $W_N^s = \exp\Big[\, -\sqrt{-1}\frac{2\pi s}{N}\,\Big]$, admits exactly N roots of unity. Therefore, setting $s = k_1 n_1 + k_2 n_2 \pmod N$, (assuming that s can take all N possible values), we can express the 2-dimensional DFT using:

$$\begin{aligned} X\,(\,k_1,\,k_2\,) &= \sum_{n_1=0}^{n_1=N-1} \sum_{n_2=0}^{n_2=N-1} x\,(\,n_1,\,n_2\,)\, W_N^{k_1 n_1 + k_2 n_2} \\ &= \sum_{s=0}^{s=N-1} \left(\sum_{k_1 n_1 + k_2 n_2 = s \pmod N} x\,(\,n_1,\,n_2\,) \right) W_N^s \end{aligned}$$

Furthermore, we note that

$$X\,(\,mk_1,\,mk_2\,) = \sum_{s=0}^{s=N-1} \left(\sum_{k_1 n_1 + k_2 n_2 = s \pmod N} x\,(\,n_1,\,n_2\,) \right) W_N^{ms}. \tag{1.1}$$

We rewrite (1.1) as one-dimensional DFT that can be computed using an FFT algorithm:

$$Y(m) = \sum_{s=0}^{s=N-1} y(s) W_N^{sm} \tag{1.2}$$

where the *DFT sums* are computed using

$$y(s) = \sum_{k_1 n_1 + k_2 n_2 = s \pmod N} x\,(\,n_1,\,n_2\,)\,, \tag{1.3}$$

and we map the directional spectrum to the 2-d DFT using:

$$\begin{array}{rcl} Y(0) & \text{is mapped to} & X(0,\,0) \\ Y(1) & \text{is mapped to} & X(k_1,\,k_2) \\ Y(2) & \text{is mapped to} & X(2k_1,\,2k_2) \\ \vdots & \vdots & \vdots \\ Y(N-1) & \text{is mapped to} & X((N-1)k_1,\,(N-1)k_2). \end{array}$$

Based on our discussion, the proposed algorithm can be summarized in 6 basic steps:

Step 1. Transpose 2-D array for computing DFT sums for $(k,\,1)$ directions.
(or distribute input matrix and its transpose among processors).
Step 2. Compute DFT sums for each prime direction $(1,\,k)$.
Step 3. Compute FFTs along each prime direction $(1,\,k)$.
Step 4. Compute DFT sums for each prime direction $(2k,\,1)$.
Step 5. Compute FFTs along each prime direction $(2k,\,1)$.
Step 6. Map directional spectrum to standard 2-D DFT spectrum.

In steps 2 and 5, we note that the prime directions $(1,\,k)$ and $(2k,\,1)$ are chosen so as to cover the entire 2-D spectrum. The fact that these directions are sufficient for covering the DFT spectrum was first published in [8] for rectangular images, and it is also given in Section 2 for square images.

We note that it is clear from the directional spectrum definition in (1.2) that the different directions are independent of each other. This observation leads to a scalable algorithm summarized as:

Step 1. Transpose 2-D array for computing DFT sums for $(k,\ 1)$ directions.
(or distribute input matrix and its transpose among processors).

Step 2. For $i = 0,\ 1,\ 2,\ \ldots,\ 2^q - 1$, execute either

Compute DFT sums for each prime direction $(1,\ 2^q + i)$.
Compute FFTs along each prime direction $(1,\ 2^q + i)$.

or

Compute DFT sums for each prime direction $(2^{q+1} + 2i,\ 1)$.
Compute FFTs along each prime direction $(2^{q+1} + 2i,\ 1)$.

Step 3. Map directional spectrum to standard 2-D DFT spectrum.

The algorithm emphasizes the fact that the scalable algorithm is very easy to implement on a parallel architecture. This is currently under investigation. Furthermore, we note that the directional spectrum computation is inherently very accurate, since a single FFT is used to compute each frequency component.

The rest of the paper is broken into 5 Sections. In Section 2, we compute the independent directions for square image of dimension $N = 2^p$. In Section 3, we derive an algorithm for computing a directional DFT along a single direction. In Section 4, an efficient algorithm for computing the DFT sums is presented. Computational results are presented in Section 5.

2 The Independent DFT Directions for $N = 2^p$

To derive a scalable algorithm, we need to find the independent DFT directions that cover the entire 2-D plane. We will show that for $N = 2^p$, for some positive integer p, there are only $\frac{3}{2}N$ independent directions. In turn, this means that only $\frac{3}{2}N$ one dimensional DFTs are needed to compute the two dimensional DFT spectrum instead of $2N$ in the standard method.

LEMMA **2.1** Let N be an integer of power 2. For any odd integer m and an integer $0 \leq k \leq N - 1$, there is an unique integer $0 \leq n \leq N - 1$ such that the product $m\,n = k(mod\ N)$.

The proof is straight-forward and can be found in elementary texts in number theory.

In the following, note that $N = 2^p$ is a power of 2, and $(k_1,\ k_2)$ is some frequency with $0 \leq k_1, k_2 \leq N - 1$. If $(k_1,\ k_2) = m(a,\ b)$ in the sense that

$$k_1 = ma \pmod N, \qquad k_2 = mb \pmod N,$$

we then say that $(k_1,\ k_2)$ is an element frequency of the direction $(a,\ b)$.

PROPOSITION **2.2** If k_1 is an odd number, then (k_1, k_2) is an element frequency of some direction $(1, k)$, and k is unique.

Proof. : By the last lemma, we choose the unique k such that the product $k\,k_1 = k_2 (mod\, N)$.

Although k is unique in the last proposition, (k_1, k_2) is possibly an element frequency of another direction $(\bar{k}, 1)$ for some $\bar{k}$.

PROPOSITION **2.3** If k_1 is even, k_2 is odd, then (k_1, k_2) is an element frequency of some direction $(k, 1)$ with k an unique even integer. Also, it cannot be element of any other direction $(1, k)$.

Proof. : By the last lemma, we choose the unique k such that the product $k\,k_2 = k_1 (mod\, N)$. Since k_1 is even and k_2 is odd, so k must be even. Since k_1 is even, but 1 is odd, so (k_1, k_2) cannot be an element frequency of $(1, k)$ for any k.

A similar argument implies that, for odd k_1 and even k_2, (k_1, k_2) is and only is an element frequency of some direction $(1, k)$ with k even.

PROPOSITION **2.4** if k_1 is even and k_2 is also even, then (k_1, k_2) is an element frequency of some direction $(1, k)$ or $(\bar{k}, 1)$ with $\bar{k}$ even.

Proof. : Dividing k_1 and k_2 by their largest common denominator (called d here), we get another frequency $(k_1/d, k_2/d)$. For this frequency, either the first component is odd or it is even but the second component is odd. By last two propositions, this new frequency is an element of some direction $(k, 1)$ or $(1, \bar{k})$ with $\bar{k}$ even. Clearly, the original frequency (k_1, k_2) is also an element in such direction.

Any frequency (k_1, k_2) is an element of direction $(1, k)$ or $(\bar{k}, 1)$ with $\bar{k}$ even. Since there are only $N + \frac{1}{2}N$ directions here, we get the following theorem about the two dimensional DFT.

THEOREM **2.5** For two dimensional DFT, at most $\frac{3}{2}N$ one dimensional DFTs are needed to complete the task.

Now we show that, at least $\frac{3}{2}N$ one dimensional DFTs are needed to complete the task.

PROPOSITION **2.6** For odd integer k_1, there is a unique odd integer k_2 such that $(1, k_2)$ and $(k_1, 1)$ are the same direction of N in the sense that they contain the same set of element frequencies.

Proof. By the last Lemma, we just choose the unique integer k_2 such that the product $k_1\,k_2 = 1 (mod\, N)$. This integer must be odd since N is even. It is obvious that $(1, k_2) = k_2\,(k_1, 1)$. Let $\mathbf{k}$ be an element frequency of the direction $(1, k_2)$, then $\mathbf{k} = m\,(1, k_2)$ for some integer m. So $\mathbf{k} = m\,k_2\,(k_1, 1)$. That is, $\mathbf{k}$ is also a element of $(k_1, 1)$. Similar argument shows that each element frequency of $(k_1, 1)$ is also an element frequency of $(1, k_2)$.

For an integer N of power 2, there are at most $\frac{3}{2}\,N$ different direction, since the direction $(k_1, 1)$ and $(1, k_2)$ for odd k_1, k_2 are pairwisely the same. But Proposition 2.3 then implies

that it has exactly $\frac{3}{2}N$ different directions. Meanwhile Proposition 2.3 and Proposition 2.6 together also means the following theorem.

THEOREM **2.7** For two dimensional DFT, exactly $\frac{3}{2}N$ one dimensional DFTs are needed to complete the task.

Proof. First, $\frac{3}{2}N$ is an upper bound, by Theorem 2.5. On the other hand, it is obvious to check that, for any integer k, $(1,k)$ cannot be an element of $(1,k_2)$ for $k_2 \neq k$, or $(k_1,1)$ for even k_1. So, taking into account the Proposition 2.3 we need all these directions to find the Fourier transform for all frequencies.

2.1 The Two Dimensional DFT Directions for $N = 4$

We consider a simple case to exhibit the basic idea of the directions and the computation of the directional spectrum. For $N = 4$, there are 7 directions with a component set to 1:

$$(1,0),\ (1,1),\ (1,2),\ (1,3),\ (0,1),\ (2,1),\ (3,1).$$

For each such direction, we can generate the element frequencies by multiplying the direction by an integer from 0 to $N-1=3$. This is shown in Table 2.1.

Direction	Element frequencies			
$\mathbf{k_1} = (1, 0)$	(0, 0)	(1, 0)	(2, 0)	(3, 0)
$\mathbf{k_2} = (1, 1)$	(0, 0)	(1, 1)	(2, 2)	(3, 3)
$\mathbf{k_3} = (1, 2)$	(0, 0)	(1, 2)	(2, 0)	(3, 2)
$\mathbf{k_4} = (1, 3)$	(0, 0)	(1, 3)	(2, 2)	(3, 1)
$\mathbf{k_5} = (0, 1)$	(0, 0)	(0, 1)	(0, 2)	(0, 3)
$\mathbf{k_6} = (2, 1)$	(0, 0)	(2, 1)	(0, 2)	(2, 3)
$\mathbf{k_7} = (3, 1)$	(0, 0)	(3, 1)	(2, 2)	(1, 3)

Table 2.1: Directions and Elements for N=4

Consider the direction $\mathbf{k_3} = (1,2)$. To compute the directional DFT, we first get the ordered pairs (n_1,n_2) such that $n_1 + 2\,n_2 = s(mod\,N)$ for $s = 0,1,2,3$. To do this, we may compute $s = n_1 + 2\,n_2(mod\,N)$ for each pair, and then pick those pairs that have same value s. This is summarized in the Table 2.2.

In this table, the first row represents the n_2, the first column represents n_1, and the other values represent s corresponding to the pair (n_1,n_2). From this table, we can find the

	$n_2 = 0$	$n_2 = 1$	$n_2 = 2$	$n_2 = 3$
$n_1 = 0$	**0**	2	**0**	2
$n_1 = 1$	**1**	3	**1**	3
$n_1 = 2$	**2**	**0**	**2**	**0**
$n_1 = 3$	**3**	**1**	**3**	**1**
$n_1 = 0$	0	**2**	0	**2**
$n_1 = 1$	1	**3**	1	**3**
$n_1 = 2$	2	0	2	0
$n_1 = 3$	3	1	3	1

Table 2.2: For direction $(1, 2)$, $s = n_1 + 2n_2 (mod\, 4)$

pairs that have the same value s and then add them up to get $\sum_{n_1+2\, n_2 = s (mod\, N)} x(n_1, n_2)$ for $s = 0, 1, 2, 3$. For example, if $s = 3$, then

$$\sum_{n_1+2n_2=3(mod\, 4)} x(n_1, n_2) = x(3, 0) + x(1, 1) + x(3, 2) + x(1, 3).$$

The two dimensional DFT in the direction $\mathbf{k_2} = (1, 2)$ can be computed using a one-dimensional FFT on the sums for:

$$s = 0,\ s = 1,\ s = 2,\ s = 3.$$

The one-dimensional FFT spectrum will be mapped to the 2-D spectrum using:

$X(0)$ is mapped to $X(0,\ 0)$
$X(1)$ is mapped to $X(1,\ 2)$
$X(2)$ is mapped to $X(2,\ 0)$
$X(3)$ is mapped to $X(3,\ 2)$

For the example above, it is easy to check that for any frequency (k_1, k_2) with $0 \le k_1 \le 3$ and $0 \le k_2 \le 3$, we can find at least one direction such that it is an element in that direction. A more important observation from this table is that each element in the direction $(3, 1)$ is also an element of some other direction. Therefore, this direction and its elements can be dropped from the Table 2.1, and the last conclusion is still true. In other words, we only need one dimensional DFTs in 6 directions to get the whole result of two dimensional DFT. The standard method requires 8 FFTs.

3 Computing a DFT sum for a single direction

Recall that before evaluating the one-dimensional DFT, we need to compute a DFT sum given by:

$$y(s) = \sum_{k_1 n_1 + k_2 n_2 = s \pmod N} x(n_1, n_2), \tag{3.4}$$

We want to derive efficient summation algorithms for the independent directions given by $(1, k)$ and $(k, 1)$.

First, consider $(1, k)$. We want to find all the pairs (m, n) such that $m + k\,n = s\,(mod\,N)$ for fixed s from 0 to $N-1$. Obviously, $(s, 0)$ is such a pair.

PROPOSITION **3.1** Given any fixed s, for each column n, there is only one element that satisfies $m + k\,n = s\,(mod\,N)$. Besides, If (m, n) is a pair such that $m + k\,n = s\,(mod\,N)$, then so is $(m-k, n+1)(mod\,N)$.

Proof. Suppose $m + k\,n = s\,(mod\,N)$ is true for both (m_1, n) and (m_2, n), we immediately get that $m_1 = m_2$, because m_1 and m_2 are between 0 and $N-1$. It is also easy to check that

$$\begin{aligned} & [(m-k) + k\,(n+1)]\,(mod\,N) && (3.5)\\ = \quad & [(m + k\,n)]\,(mod\,N) && (3.6)\\ = \quad & s\,(mod\,N). && (3.7) \end{aligned}$$

This proposition shows that there are exactly N elements for each sum $\sum_{m+kn=s(mod\,N)} x(m, n)$, and these elements distributed in different columns. We can also pick up these elements easily by the formula in the proposition. Especially, we have the formula:

$$\sum_{m+kn=s\,(mod\,N)} x(m, n) = \sum_{j=0}^{N-1} x(s - jk, j), \tag{3.8}$$

where, $s - jk$ should be understood as the integer module N.

In developing an algorithm, we can utilize the vector addition to compute all N sums together for a fixed direction $(1, k)$. Next, we tackle the prime directions given by $(k, 1)$.

PROPOSITION **3.2** Suppose k is an even number, and the integer $0 < p < N$ is the smallest integer satisfying $k\,p = 0\,(mod\,N)$. If (m, n) is a pair such that $k\,m + n = s\,(mod\,N)$, then so is $(m + p, n)(mod\,N)$. And if $(m + q, n)\,(mod\,N)$ is another such pair for some integer q, then q is a multiple of p.

Proof.

$$\begin{aligned} & [k\,(m+p) + n]\,(mod\,N) && (3.9)\\ = \quad & [k\,m + n]\,(mod\,N) + (k\,p)\,(mod\,N) && (3.10)\\ = \quad & s\,(mod\,N). && (3.11) \end{aligned}$$

For the another part, we easily have $k\,(q-p) = 0\,(mod\,N)$. Since p is the smallest integer satisfying $k\,p = 0\,(mod\,N)$, so $q-p$ must be a multiple of p, and so is q.

Note that, since $k\,p = 0\,(mod\,N)$, so N is divisible by p in the case that $k \neq 0$. In other words, p itself is also a power of 2. To see this more clearly, write $k = 2^\alpha\,\beta$ with some odd number β, then we easily see that $p = N/2^\alpha$. Now set $\overline{p} = N/p = 2^\alpha$, then $\overline{p}$ is the smallest integer such that there is an integer q satisfying $k\,q + \overline{p} = 0(mod\,N)$. To see this, suppose $\overline{p} < 2^\alpha$ and $k\,q + \overline{p} = 0, (mod\,N)$ for some q, then $\overline{p}$ must be even, because k is even. So we may write $\overline{p} = 2^\gamma\omega$ with $\gamma < \alpha$ and ω odd , then we would have $2^\gamma(1 + 2^{\alpha-\gamma}\,\beta\,q) = 0\,(mod\,N)$. But the number in the brace is odd and $2^\gamma < N$, so it's impossible to have a q such that the residue of the product is zero. On the other hand, if $\overline{p} = 2^\alpha$, taking q satisfying $\beta\,q = N-1\,(mod\,N)$, then $k\,q + \overline{p} = 2^\alpha(1+\beta\,q) = 0\,(mod\,N)$.

PROPOSITION **3.3** Suppose $k = 2^\alpha\,\beta$ is an even number with β odd, $\overline{p} = 2^\alpha$, and q satisfying $\beta\,q = N-1\,(mod\,N)$. If (m,n) is a pair such that $k\,m + n = s\,(mod\,N)$, then so is $(m+q, n+\overline{p})\,(mod\,N)$.

Proof. We just need to check that

$$\begin{aligned} & [k\,(m+q) + (n+\overline{p})]\,(mod\,N) \\ = \; & [(k\,m+n) + (k\,q+\overline{p})]\,(mod\,N) \\ = \; & [s + 2^\alpha\beta\,q + 2^\alpha]\,(mod\,N) \\ = \; & [s + 2^\alpha(\beta\,q+1)]\,(mod\,N) \\ = \; & s\,(mod\,N). \end{aligned}$$

By Proposition 3.2, 3.3 and the argument between them, for even $k = 2^\alpha\beta$ with β odd,

$$\sum_{k\,m+n = s\,(mod\,N)} x(m,n) = \sum_{j=0}^{p-1}\sum_{i=0}^{\overline{p}-1} x(ip+jq, s+jN/p), \tag{3.12}$$

where, the integers $ip+jq$ and $s+jN/p$ should be understood as the integer module N.

Note, for $k=0$, it does not have the form $k = 2^\alpha\beta$. But we can still choose $p=1$ and $\overline{p} = N$, and the last formula is still true. The choice of q does not matter. Naturally, we choose the smallest one.

4 An Efficient Algorithm for Computing DFT Sums

For computing directional DFTs, we must first evaluate

$$\sum_{m+kn = s(mod\,N)} x(m,n), \quad \text{for } k = 0,1,\cdots,N-1; \quad s = 0,1,\cdots,N-1. \tag{4.13}$$

For each direction of the form $(1, k)$ and each index s, we have to find the summands $x(m, n)$ satisfying $m + k\,n = s \pmod N$. For convenience, associated with each direction $(1, k)$, an index function

$$s_k(m, n) = m + k\,n \pmod N \tag{4.14}$$

is defined for the row index m and the column index n.

Observation 1: $x(m_1, n_1)$ and $x(m_2, n_2)$ are two of the N summands for the direction $(1, k)$ and the index s, if and only if $s_k(m_1, n_1) = s_k(m_2, n_2) = s$.

Therefore, all the necessary summands can be found using this function. This will be done next.

Another problem is the computational work on these additions. Since there are $N - 1$ additions in each sum, so the total works would amount to $N^2(N - 1)$ if it is done naively. Fortunately, we can do it more efficiently under further investigation. The total work will be reduced to $N^2 \log N$.

Before discussing the algorithm in detail, we have another observation. This observation makes it easier to gain a better understanding of the algorithm.

Observation 2: For the direction $(1, k)$ and the n-th column of the input x, different m value produces different s_k value. That is,

$$s_k(m_1, n) \neq s_k(m_2, n) \qquad \text{for } m_1 \neq m_2.$$

This is essentially the Proposition 3.1. We would like to use an example to explain how it works. Take $N = 8$, $k = 3$, and n is chosen as 1 and 6, respectively. The Table 4.3 shows the $s_k(m, n)$ values for different m values. It confirms that, for the same n (1 and 6 in the table), the s_3 values for all the m values from 0 to 7 is just a re-ordering of the integers from 0 to 7. One implication of this observation is that, the summands for any s and any k cannot be in the same column of the input x.

m	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
$s_3(m, 1)$	**3**	**4**	**5**	**6**	**7**	**0**	**1**	**2**	3	4	5	6	7	0	1	2
$s_3(m, 6)$	2	**3**	**4**	**5**	**6**	**7**	**0**	**1**	**2**	3	4	5	6	7	0	1

Table 4.3: The $s_k(m, n)$ values for $s = 3$, $n = 1, 6$, $N = 8$

Another implication of this observation is that, we can add two columns together as vector addition instead of adding them one by one. This has great advantage in modern

high performance computing. To make it clear, we doubled the m and s_3 values in the table 4.3. And we observe that, if the third row of this table is shifted one position to the left, then the second and the third rows are exactly the same (except at both ends where we don't care). Expressing it mathematically,

$$s_3(0:7,1) = s_3(1:8,4). \tag{4.15}$$

Here, $(0:7)$ means the integer vector with elements from 0 to 7, so $s_3(0:7,6)$ is also a vector. Similarly, $(1:8)$ is the integer vector with elements from 1 to 8. The exception is that, if the integer is greater than or equal to 8, then this integer is subtracted by 8. Therefore, $(1:8)$ is in fact the one position left shift of $(0:7)$. Now, by the Observation 1, we can do the addition like

$$x(0:7,1) + x(1:8,6). \tag{4.16}$$

And this produces the partial sums for the direction $(1,3)$ and the frequencies s from 0 to 7.

Now we make a detail explanation of the implementation of the algorithm.

PROPOSITION **4.1**

$$\begin{cases} s_k(m, n+N/2) = s_k(m,n) & \text{for } k \text{ even}, \\ s_k(m+N/2, n+N/2) = s_k(m,n) & \text{for } k \text{ odd}. \end{cases} \tag{4.17}$$

The proof of this proposition is obvious. We will make a general proof in the Theorem 4.3. The meaning of this proposition is that, for the even direction, we can add the n-th column and the $n+N/2$-th column to get the partial sum that is needed in the DFT computation in that direction. Similarly, for the odd direction, we can add the n-th column and the $n+N/2$-th column with a $N/2$ row shifting to get the partial sum that is needed in the DFT computation in that direction. So, we can do

$$B(:,\, 0:N/2-1) = A(:,\, 0:N/2-1) + A(:,\, N/2:N-1) \tag{4.18}$$

and

$$B(:,\, N/2:N-1) = A(:,\, 0:N/2-1) + A(:+N/2,\, N/2:N-1), \tag{4.19}$$

where $:+N/2$ means a row shifting by $N/2$. The first sum is useful for even directions, and the second sum is useful for odd directions. This is the first step and is explained by the Figure 4.1 and Figure 4.2.

In the second step, we redefine B as A, and split A into two groups. The first group contains the first $N/2$ columns, and the another group contains the last $N/2$ columns of A. Remember that the first group applies to even directions, whereas the second group applies to the odd directions.

In the following, we say m is in certain group if the m-th row of A is in that group. p will be always an integer.

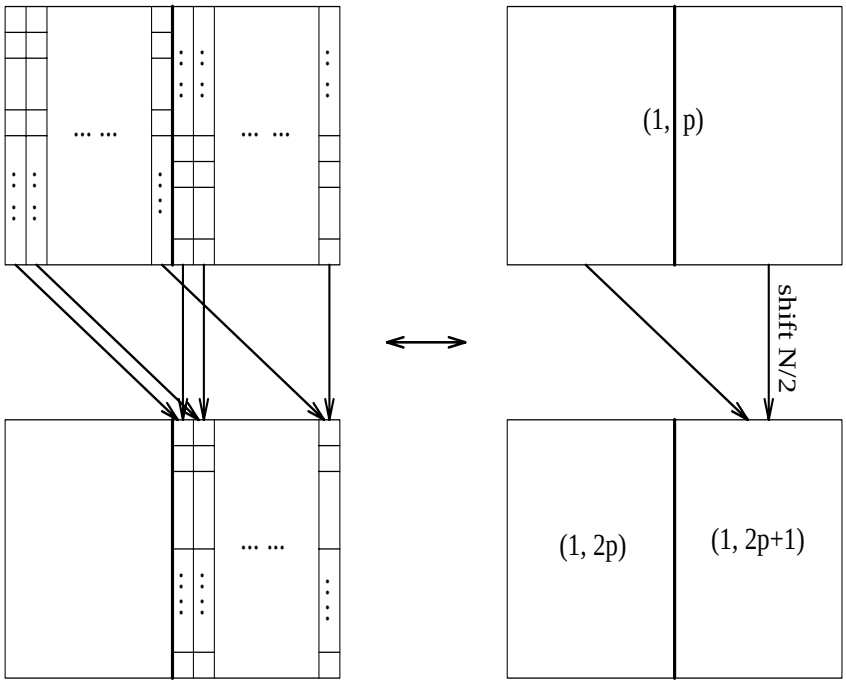

Figure 4.1: Directional sum for $(1, 2p)$. We will use the figure on the right to represent the computation depicted in the left figure. In the left figure, column $i+N/2$ is added to column i for $i = 0, 1, \cdots, N/2 - 1$.

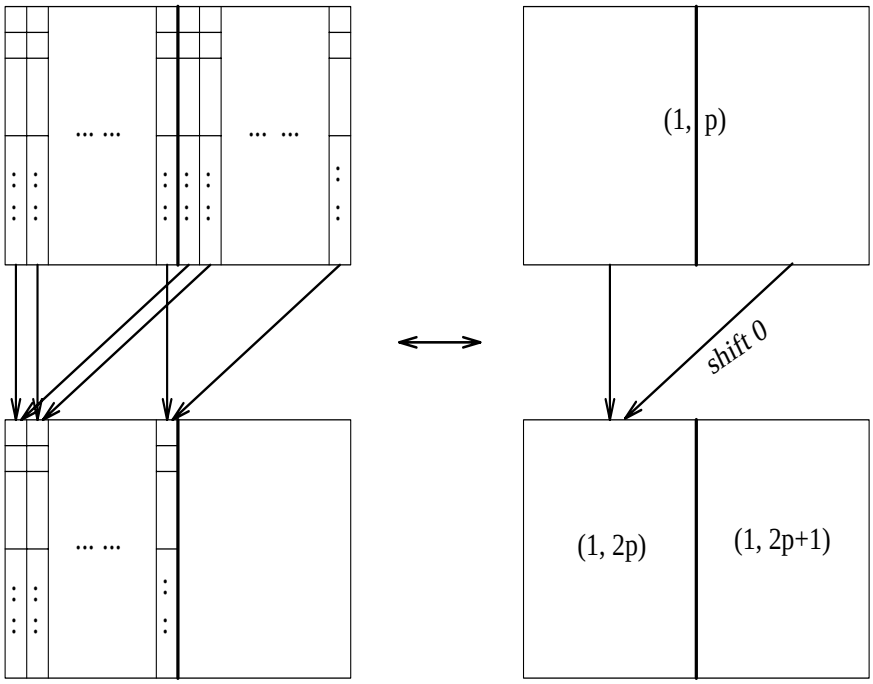

Figure 4.2: Directional sum for $(1, 2p+1)$. As in the Figure 4.1, column $i + N/2$ is added to column i for $i = 0, 1, \cdots, N/2 - 1$. However, in this case, the $(i + N/2)$-th column is circularly shifted down by $N/2$ shifts.

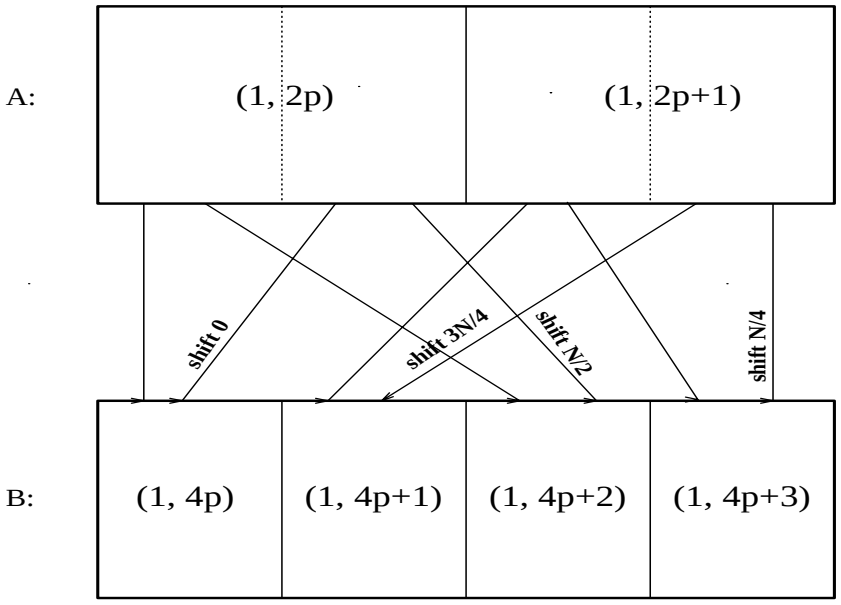

Figure 4.3: The second step

PROPOSITION **4.2**

$$\begin{cases} s_k(m,n) = s_k(m, n+N/4) & \text{for } k = 4p\,, \\ s_k(m,n) = s_k(m+3N/4, n+N/4) & \text{for } k = 4p+1\,, \\ s_k(m,n) = s_k(m+2N/4, n+N/4) & \text{for } k = 4p+2\,, \\ s_k(m,n) = s_k(m+N/4, n+N/4) & \text{for } k = 4p+3\,. \end{cases} \tag{4.20}$$

So, in each group, the first half and the second half of the group are added together twice with suitable row shifts. For the first group of A,

$$B(:,\, 0 : N/4-1) = A(:,\, 0 : N/4-1) + A(:,\, N/4 : N/2-1) \tag{4.21}$$
$$B(:,\, 2N/4 : 3N/4-1) = A(:,\, 0 : N/4-1) + A(:\, +2N/4,\, N/4 : N/2-1) \tag{4.22}$$

For the other group of A,

$$B(:,\, N/4 : 2N/4-1) = A(:,\, 2N/4 : 3N/4-1) + A(:\, +3N/4,\, 3N/4 : N-1) \tag{4.23}$$
$$B(:,\, 3N/4 : N-1) = A(:,\, 2N/4 : 3N/4-1) + A(:\, +N/4,\, 3N/4 : N-1) \tag{4.24}$$

To make life easier, the result is stored so that the first $N/4$ rows of B applies to the directions of the form $(1, 4p)$, the second $N/4$ rows applies to the directions of the form $(1, 4p+1)$, and so on. The detail could be seen from Figure 4.3.

It is easy to know that after this step, we have added four columns of the original input for each direction.

In general step j, we have the following result.

THEOREM **4.3** If $k = C\,p + i$ with $C = 2^j$, then $s_k(m,n) = s_k(m + (C-i)N/C, n + N/C)$.

Proof.

$$\begin{aligned} & s_k(m + (C-i)\frac{N}{C}, n + \frac{N}{C}) \\ = \;& m + (C-i)\frac{N}{C} + k\,(n + \frac{N}{C}) \\ = \;& m + k\,n + N - i\,\frac{N}{C} + (C\,p + i)\frac{N}{C} \\ = \;& m + k\,n + (p+1)\,N \\ = \;& m + k\,n (mod\, N) \end{aligned}$$

Interpreting this theorem, for the original input, we can add the n-th column and the $(n + N/C)$-th column with a row shifting $(C-i)N/C$ to get the partial sum for the specific direction $k = C\,p + i$, where n can be from 0 to some suitable integer. Because of the work in the previous step, we do the following.

The result from the last step is split into $C/2$ groups, each group containing $2N/C$ column partial sums for the direction $C/2 + i$ for some i from 0 to $C/2 - 1$. In each group, we add the first half columns and the other half columns twice with row shifting $(C-i)N/C$ and $(C-i)N/C - N/2$, respectively. After this step, we have added 2C columns of the original input for each direction.

In the last step, $j = \log N$, and $p = 0$ in the above theorem. The previous partial sums are split into $N/2$ group, each containing 2 columns. For each group, we add these 2 columns twice with suitable row shifts. Then the whole additions is done. We get all the sums for all directions.

THEOREM **4.4** The number of additions in the algorithm described above is $N^2 \log N$.

Proof. There are total $\log N$ steps in the algorithm. In the j-th step, there are 2^{j-1} groups, each group containing $N/2^{j-1}$ columns. We add the first half columns with the second half columns of this group twice. Each column has N elements. So the total additions for each group is $N/2^{j-1} \times N = N^2/2^{j-1}$. Sum over the all the groups in this step, the total work is $N^2/2^{j-1} \times 2^{j-1} = N^2$. This means, there are N^2 additions for every step, independent of j. Finally, the total additions for the algorithms is $N^2 \log N$.

The psudo-code for the sums of all directions $(1, k)$ is appended at the end of this report. In this psudo-code, the size of X is $2N$ by N. Initially, both $X(0 : N-1, :)$ and $X(N : 2N-1, :)$ are a copy of the input to be transformed. In practice, it is not necessary to make such copy.

Although we have got all the sums for all directions of the form $(1, k)$, and we also know the sums for the direction $(1, k)$ is in the k-th column. We are still not sure if this column is in correct order. That is, is the sum $\sum_{m+kn}^{=s(mod\,N}\ x(m, n)$ in the s-th position of this column? The answer is yes. We can prove it by induction.

In the first step, there is only one group, that is the whole input. We can think it as the partial sums for all directions. The first column of this group is surely in the correct order, because $n = 0$ so $s_k(m, n) = m + kn = m$.

Now in the j-th step, there are 2^j groups. For each group, we generate 2 groups for the next step. We assume that the first column of each group in the j-th step is in the correct order. And we want to prove the first column of the generated 2 group in the next step is also in correct order. Note that, during the additions at the j-th step, the first half columns of each group does not have any shift, so does the first column of the group. Because the first column in the generated group for the next step is the sum of the first column of the some group in the j-th step with some other column (with or without shifting), so it also has the correct order. At the final step, each group only has one column. This column is of course the first column of the group, and therefore has the correct order.

4.1 DFT sum computation for $N = 8$

For $N = 8$, we describe the DFT sum algorithm. There are $\log 8 = 3$ steps in the algorithm. We first define

$$\begin{aligned} A(0:7, 0:7) &= x(0:7, 0:7) \\ A(8:15, 0:7) &= x(0:7, 0:7)\,. \end{aligned}$$

Note that, we don't need the second copy in the actual implementation. We do it here is just for easier explanation.

At the first step, we add the first 4 columns of A with its last 4 columns:

$$\begin{aligned} 2p: &\quad B(0:7, 0:3) = A(0:7, 0:3) + A(0:7, 4:7) \\ 2p+1: &\quad B(0:7, 4:7) = A(0:7, 0:3) + A(4:11, 4:7)\,. \end{aligned}$$

Here, the symbol $2p$ and $2p+1$ before the equation means the directions that requires the partial sums. So, $B(0:7, 0:3)$ is the partial sum needed for the directions $(1, 0), (1, 2), (1, 4)$ and $(1, 6)$. Now we make another copy of B:

$$B(8:15, 0:7) = B(0:7, 0:7)\,.$$

At the second step, B is divided into two groups. After summation, we have

$$\begin{aligned} 4p: &\quad C(0:7, 0:1) = B(0:7, 0:1) + B(0:7, 2:3) \\ 4p+2: &\quad C(0:7, 4:5) = B(0:7, 0:1) + B(4:11, 2:3) \\ 4p+1: &\quad C(0:7, 2:3) = B(0:7, 4:5) + B(6:13, 6:7) \\ 4p+3: &\quad C(0:7, 6:7) = B(0:7, 4:5) + B(2:9, 6:7)\,. \end{aligned}$$

Now, let's check the result $C(3,4)$ and $C(1,5)$ carefully. $C(3,4)$, being the first colum of $C(0:7,4:5)$, should be the partial sum for the directions $(1,2)$ and $(1,6)$ with frequency $s=3$. In fact,

$$\begin{aligned} C(3,4) &= B(3,0)+B(7,2) \\ &= A(3,0)+A(3,4)+A(7,2)+A(7,6) \\ &= x(3,0)+x(3,4)+x(7,2)+x(7,6)\,. \end{aligned}$$

Since

$$\begin{aligned} s_2(3,0)=s_2(3,4)=s_2(7,2)=s_2(7,6)=3 \\ s_6(3,0)=s_6(3,4)=s_6(7,2)=s_6(7,6)=3\,, \end{aligned}$$

therefore, $C(3,4)$ is indeed the partial sum for the directions $(1,2)$ and $(1,6)$ with frequency $s=3$. Similarly,

$$\begin{aligned} C(1,5) &= B(1,1)+B(5,3) \\ &= A(1,1)+A(1,5)+A(5,3)+A(5,7) \\ &= x(1,1)+x(1,5)+x(5,3)+x(5,7) \end{aligned}$$

is also the partial sum for the directions $(1,2)$ and $(1,6)$. But because it is not in the first column of $C(0:7,4:5)$, so it is not for the frequency $s=1$. Instead, one may check that it is for the frequency $s=3$.

At the last step, after the self copying, C is divided into 4 groups. And we do

$$\begin{aligned} 0: &\quad D(0:7,0)=C(0:7,0)+C(0:7,1) \\ 4: &\quad D(0:7,4)=C(0:7,0)+C(4:11,1) \\ 1: &\quad D(0:7,1)=C(0:7,2)+C(7:14,3) \\ 5: &\quad D(0:7,5)=C(0:7,2)+C(3:10,3) \\ 2: &\quad D(0:7,2)=C(0:7,4)+C(6:13,5) \\ 6: &\quad D(0:7,6)=C(0:7,4)+C(2:9,5) \\ 3: &\quad D(0:7,3)=C(0:7,6)+C(5:12,7) \\ 7: &\quad D(0:7,7)=C(0:7,6)+C(1:8,7)\,. \end{aligned}$$

Let's have a check for $D(3,2)$ which should be the sum for the direction $(1,2)$ with frequency $s=3$. In fact,

$$\begin{aligned} D(3,2) &= C(3,4)+C(9,5) \\ &= C(3,4)+C(1,5) \\ &= x(3,0)+x(3,4)+x(7,2)+x(7,6)+x(1,1)+x(1,5)+x(5,3)+x(5,7)\,. \end{aligned}$$

Since

$$s_2(3,0) = s_2(3,4) = s_2(7,2) = s_2(7,6) = s_2(1,1) = s_2(1,5) = s_2(5,3) = s_2(5,7) = 3\,,$$

so indeed, $D(3,2)$ is the sum expected. Note, by the theory in the Section 2, there are exactly 8 summands for each sum.

The whole step of the algorithm for $N = 8$ is depicted in Figure 4.4.

5 Computational Results

In this section, some computational results of the directional DFT algorithms are presented. The algorithm described in Section 4 is implemented using FORTRAN90 [1]. Because one dimensional DFTs are needed in the directional DFT method, so the performance of our method is dependent on the one dimensional FFT code used. For the one-dimensional FFT code, we used `FFTPACK` and `FFTW` [3], [4], [5], [2], [6], [9]. For the `FFTPACK`-based implementation, single precision arithmetic is used. For the `FFTW`-based implementation, double precision is used. Furthermore, in the following simulations, the input images were assumed to be of integer type. All computation was done at the Blackbear in the Albuquerque High Performance Computing Center. The processor of Blackbear is a 550MHz PentiumIII equipped with 144GB of RAM.

5.1 Accuracy of the Directional DFT Method

First, we tested the correctness of our method. Hence, we also implement the backward directional DFT transform. For the inverse directional DFT, we call one dimensional inverse DFT (instead of forward DFT used for forward DFT).

So suppose x is the original input image. Using the directional forward DFT, we get the Fourier transform of x, and denote it by X. Then, applying the backward directional DFT to X, we get the backward Fourier transform of X, and is denoted by $\tilde{x}$. If there is no error in the computation, $\tilde{x}$ would be x itself. In this case, we recovered the original image. But, because of the error in the numerical computation, $\tilde{x}$ may not exactly the same as x. The error between x and $\tilde{x}$ measures the correctness of the method, as well as the accuracy of the method.

Two kinds of error estimation are used in our computation: (i) the l_∞ norm, and (ii) the l_2 norm. They are defined by

$$e_{l_\infty} = \max_{0\le i,\, j\le N-1} |x_{ij} - \tilde{x}_{ij}|, \tag{5.25}$$

$$e_{l_2} = \Big(\sum_{0\le i,\, j\le N-1} |x_{ij} - \tilde{x}_{ij}|^2 \Big)^{\frac{1}{2}}, \tag{5.26}$$

respectively.

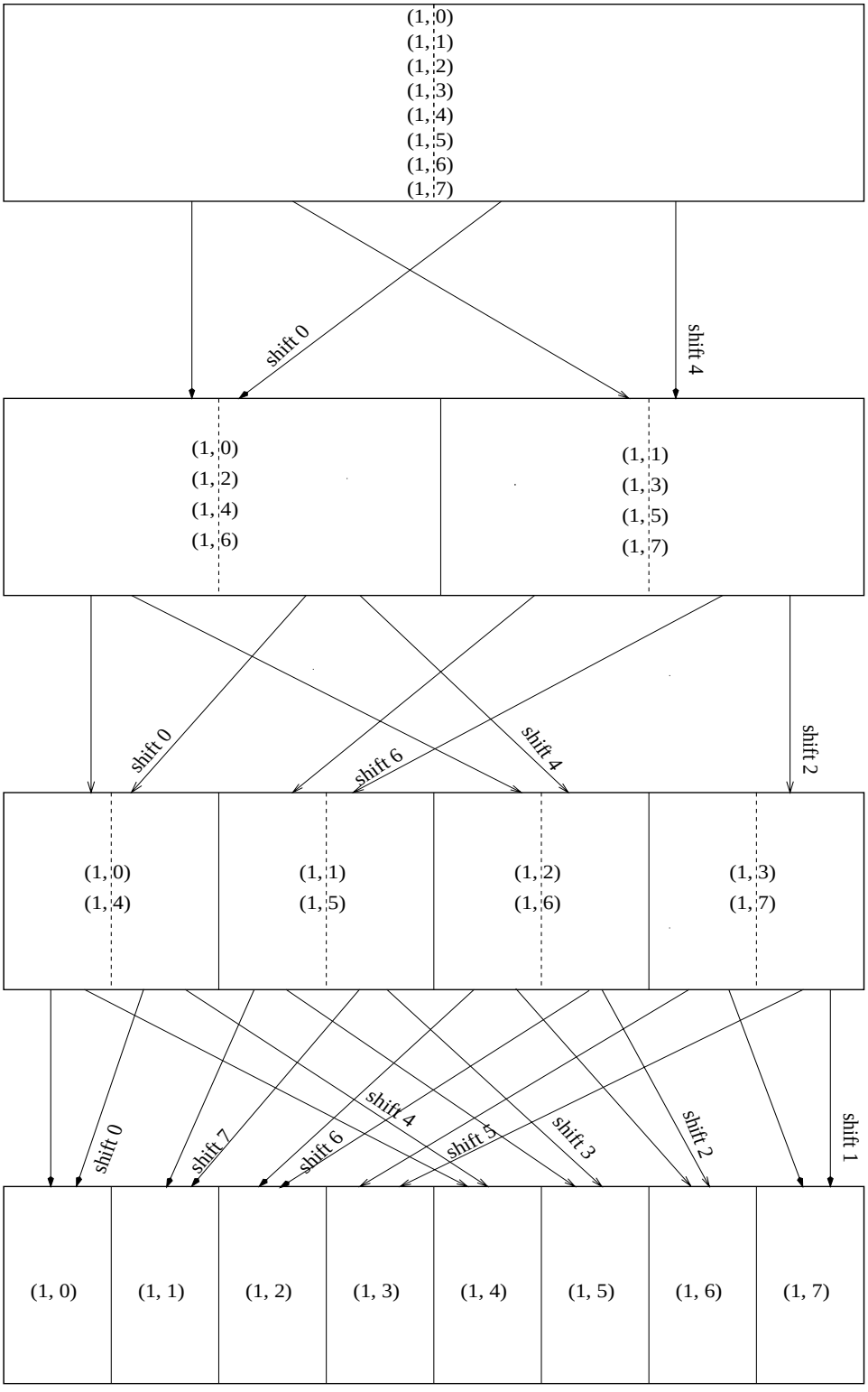

Figure 4.4: The whole step for $N = 8$.

In our examples, $N = 512$. For testing, we first used

$$x(m,n) = \cos((i+j)\frac{2\pi}{N}) + \sin((i+3j)\frac{2\pi}{N})\,. \tag{5.27}$$

Using the directional DFT with one-dimensional method from `FFTPACK`, the l_∞ error and the l_2 norm of the recovery are 8.366×10^{-5} and 2.388×10^{-5}, respectively. Because only single precision is used in the computation, so these errors imply that the method produces the correct result for this example. To avoid accidental confirmation, we also tried many input images by generating them randomly in the interval of [0, 1]. The average of the l_∞ and l_2 norm errors are reported in the Table 5.4. And it also confirm that the directional DFT works correctly. Table 5.5 contains the error comparison using the one dimensional code from FFTW, and using double precision.

	Directional		Standard	
Example	l_∞ error	l_2 error	l_∞ error	l_2 error
(5.27)	8.366×10^{-5}	2.388×10^{-5}	1.132×10^{-4}	3.478×10^{-5}
random	5.047×10^{-5}	1.632×10^{-5}	6.270×10^{-5}	1.947×10^{-5}

Table 5.4: The l_∞ and l_2 norm error of image recovery. The error from the directional method is approximately 20% less than that from the usual method. The 1-D code from FFTPACK are used with single precision.

	Directional		Standard	
Example	l_∞ error	l_2 error	l_∞ error	l_2 error
(5.27)	1.806×10^{-13}	4.968×10^{-14}	7.959×10^{-12}	2.763×10^{-12}
random	1.151×10^{-13}	3.384×10^{-14}	1.459×10^{-13}	3.998×10^{-14}

Table 5.5: The l_∞ and l_2 norm error of image recovery. The error from the directional method is approximately 20% less than that from the usual method. For the first example, the error from the directional method is two order less than that from the usual method. The 1-D FFTs in `FFTW` use double precision.

Note that, the input images are integers. So all the sums computed in the directional method can be exact without computational error if there is no overflow occurs. In this case, we believe the directional method produces more exact result than the existing methods do.

The reason is, directional DFT only need one one-dimensional DFT to get the transform for any frequency, while the existing methods always needs $2N$ one-dimensional DFTs to get the transform (except for the frequencies $(0, k)$ and $(k, 1)$ which may only need $N + 1$ one-dimensional DFTs). The more the one-dimensional DFTs required, the more the accumulated computational error. This is confirmed by the Table 5.4 and Table 5.5. In these table, the 4th and the 5th columns contain the errors computed using the standard method. No matter what image is input, the error from the directional method is always approximately 20% less than that from the usual method.

5.2 Scalability of the Directional Method

An advantage of the directional method is that, it provides a way to compute the transform in some specific directions.

In some applications, we only need to get the Fourier transform in several directions. In this case, using directional DFT is efficient. Because in the directional DFT, only N sums and a one dimensional DFT is need, while in the existing methods, $2N$ one dimensional DFT are needed as mentioned in the above. The computational complexity for a sum is much less than that for a one dimensional DFT. Therefore, the directional DFT method should be much faster than the existing methods in computing frequencies along one direction. We utilize the intrinsic procedure *time_and_date* to record the computational time, and all the time recorded are present in Table 5.6 and Table 5.7. In Table 5.6, the one dimensional DFT method is from FFTPACK, and single precision is used, while in Table 5.7, the one dimensional DFT method is from FFTW and double precision is used. The unit for all the time is millisecond. The factor determines how fast the directional DFT method compared with the usual method. we see that, this factor is increasing as the problem size N increases. For $N \geq 256$, the directional DFT method is at least 11 times faster.

Next the directional method is used to compute the subsampled DFT spectrum. Our first example is for the directions $(1, 2k)$ and $(4k, 1)$ with k integers. The second example is for the directions $(1, 2k + 1)$ and $(4k + 2, 1)$. The standard method has no way to do that, except finding the transforms for all frequencies. For $N = 8$, Table 5.8 and Table 5.9 indicate the sampling of these two examples. Each table samples part of the spectrum. They both together cover the entire spectrum. Table 5.10 contains the average computational time for these two examples.

Finally, we tried to use the directional method to compute the subsampled DFT spectrum determined by the direction $(1, 4k)$ together with $(8k, 1)$ and the directions $(1, 4k+2)$ together with $(8k + 4, 1)$. The timing is reported in the Table 5.11.

5.3 Two Dimensional DFT Computational Results

Now we compute the Fourier transform for all frequencies. In this case, the direction DFT method needs $N + 1$ one dimensional DFTs, while the usual method needs $2N$. For each

N	Directional		Standard	Speed-up
	$(1,k)$	$(k,1)$	$(1,k)$	
128	2	2	12	6
256	5	5	56	11.2
512	23	23	291	12.6
1024	93	95	1252	13.4

Table 5.6: Timings for computing a single direction of the DFT-spectrum. All the algorithms are implemented in FORTRAN 90, and timing are presented in milliseconds (compute using DATE_AND_TIME function). The code was run on the super cluster Blackbear in Albuquerque High Performance Center. The processor of Blackbear is a 550MHz PentiumIII equipped with 144GB of RAM. In this table, the 1-dimensional code provided with FFTPACK is used. Also, the FORTRAN 90 TRANSPOSE function is used for computing the 2-D transpose. For the single column—row results (3rd column), a single transpose is used. For the full-spectrum results (4th column), two transposes were needed.

one dimensional DFT, the computational work is exactly $N \log N$, and most of operation is multiplication. Therefore, the computational complexity in the one dimensional DFT part are $N(N+1)\log N$ and $2N^2 \log N$. For the directional method, there are N^2 sums. They can be computed in $N^2 \log N$ steps, and all the operations are additions. So, roughly speaking, the computational work for the directional DFT method is

$$N(N+1)\log N + N^2 \log N = 2N^2 \log N + N \log N. \tag{5.28}$$

And for the usual method, the computational work is

$$2N^2 \log N. \tag{5.29}$$

But note that, most work in the directional DFT method is addition, while most work in the usual method is multiplication. Multiplication is more expense than addition. So we expect that the time for the computation in directional DFT method is much less than that in usual method. One problem is that, besides these addition and multiplication operations, all methods need to fecth the data from the memory and save the data to the storage. This part of work takes an un-negnected time. It is reasonable that this part is machine dependent. Our experiments show that, for this part of the work, the directional DFT method takes longer time than the usual method. This is the drawback of the new method.

Table 5.14 and Table 5.15 show the time records for the new methods. Table 5.14 uses the one dimensional method from FFTPACK with single precision, and Table 5.15 uses the

N	Directional		Standard	Factor
	$(1,k)$	$(k,1)$	$(1,k)$	
128	2	2	16	8
256	7	8	83	10.4
512	27	28	348	12.5
1024	107	110	1626	14.7

Table 5.7: Time comparison for one directional frequencies using FFTW

method from FFTW with double precision. We carefully record the time for different part of the work. The time for additions is greater than the time for one dimensional DFTs due to fetching and saving data. The time used in traditional methods computing the two dimensional DFT is listed in Table 5.12, including all the work, if FFTPACK is used. And the time is listed in Table 5.13 if FFTW is used. We do not expect the new method defeats FFTPACK or FFTW right now. What we want to show is that, the new method is another way to do FFTs. It is comparable to the existing methods. And it may produce faster tools to do FFTs under further investigation and using other advanced techniques.

6 Future Research

Ongoing and future research will focus on two distinct areas: (i) a more efficient implementation of the directional and standard DFT for the Pentium III architecture, and (ii) parallel extensions of the directional DFT in multiple dimensions. We next describe each research area in more detail.

For all the algorithms presented in this report, none of the SIMD integer and SIMD floating point extensions of the Pentium III have been exploited. In addition, none of the cache memory control extensions of the Pentium III have been exploited either. This resulted in severe underutilization of the CPU capabilities.

Currently, Balaji Raman with Marios Pattichis are developing new algorithms that will take advantage of these capabilities. For this purpose, a SIMD FFT algorithm developed by the Intel Architecture labs is used. Using the SIMD extensions, it is possible to add 4 floating point (or integer) numbers at a time, and hence a significant speedup in the FFT sum computation time is expected. Using the cache extensions, the final step of mapping the directional spectrum to the standard DFT spectrum is also expected to improve substantially. Overall, it is expected that the new implementation will allow the developed directional DFT algorithms to become significantly faster than the standard 2-D FFTs.

	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
m=0	X				X			
m=1		X		X		X		X
m=2		X	X	X		X	X	X
m=3		X		X		X		X
m=4			X		X		X	
m=5		X		X		X		X
m=6		X	X	X		X	X	X
m=7		X		X		X		X

Table 5.8: Subsampled DFT spectrum by directions $(1, 2k+1)$ and $(4k+2, 1)$ when N=8. X's indicate the spectrum generated by these directions.

The authors will investigate a 2-D extension for rectangular images $N_1 \times N_2$, and a 3-D extension for images of general size $N_1 \times N_2 \times N_3$. The new extensions will generalize the approach, and also provide important frameworks for parallel implementations. The basic scheme for developing parallel implementations has already been presented in this report.

7 Acknowledgements

The authors would like to acknowledge the support of AHPCC. We would also like to express our gratitude to Dr. Susan Atlas and Dr. Brian Smith for many discussions on this research.

	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
m=0	X	X	X	X	X	X	X	X
m=1	X		X		X		X	
m=2	X				X			
m=3	X		X		X		X	
m=4	X	X		X		X		X
m=5	X		X		X		X	
m=6	X				X			
m=7	X		X		X		X	

Table 5.9: Subsampled DFT spectrum by directions $(1, 2k)$ and $(4k, 1)$ when N=8 X's indicate the spectrum generated by these directions.

	$(1, 2k)$ and $(4k, 1)$	$(1, 2k+1)$ and $(4k+2, 1)$
additions for $(1, k)$	70	72
1D FFTs for $(1, k)$	27	26
additions for $(k, 1)$	37	36
1D FFTs for $(k, 1)$	14	14
total	148	148

Table 5.10: Computational time (milliseconds) for subsampling with $N = 512$ (I).

	$(1, 4k)$ and $(8k, 1)$	$(1, 4k+2)$ and $(8k+4, 1)$
additions for $(1, k)$	44	43
1D FFTs for $(1, k)$	16	15
additions for $(k, 1)$	29	28
1D FFTs for $(k, 1)$	8	8
total	97	94

Table 5.11: Computational time (milliseconds) for subsampling with $N = 512$ (II).

N=1024	N=512	N=256	N=128
1618	359	61	12

Table 5.12: Time of the standard method using FFTPACK. All work are counted, including $2N$ 1D DFTs and two transposes.

N=1024	N=512	N=256	N=128
1677	378	87	15

Table 5.13: Time of the standard method using FFTW. All work are counted, including $2N$ 1D DFTs and two transposes.

step	work	N =1024	N = 512	N =256	N=128
step 1	additions for $(1,k)$ directions	616	138	26	5
step 2	1D FFTs for $(1,k)$ directions	260	64	13	4
step 3	additions for $(k,1)$ directions	302	66	13	2
step 4	1D FFTs for $(k,1)$ directions	133	33	6	2
step 5	Mapping for all directions	458	33	22	5
step 1, 2, 3, 4	time for directional spectrum	1311	301	58	13
step 1, 2, 3, 4, 5	time for standard spectrum	1769	404	80	18

Table 5.14: Timing for computing both the directional and the standard DFT spectrum for square N by N matrices. All the algorithms were implemented in FORTRAN 90, and timings are presented in milliseconds. The 1-D codes from FFTPACK are used.

step	work	N=1024	N=512	N=256	N=128
step 1	additions for $(1,k)$	616	140	26	6
step 2	1D FFTs for $(1,k)$	489	105	26	6
step 3	additions for $(k,1)$	303	66	13	2
step 4	1D FFTs for $(k,1)$	245	52	12	2
step 5	1D FFTs for $(k,1)$	628	148	34	7
step 1, 2, 3, 4	time for directional spectrum	1653	363	77	16
step 1, 2, 3, 4, 5	time for standard spectrum	2281	511	111	23

Table 5.15: Time of the directional method using FFTW

8 Appendix: Fortran 90 Code for Computing DFT Sums

```
GROUPS = 1
DO WHILE(GROUPS<=N/2)

   VIEG = N/GROUPS
   HV = VIEG/2

   OFFSET1 = 0
   OFFSET2 = 0
   SHIFT1  = N

   DO K = 1, GROUPS

      SHIFT2 = SHIFT1 - N/2

      XX(0:N-1, OFFSET2:OFFSET2+HV-1) = &
             X(0:N-1, OFFSET1:OFFSET1+HV-1) + &
             X(SHIFT1:SHIFT1+N-1, OFFSET1+HV:OFFSET1+VIEG-1)

      XX(0:N-1, OFFSET2+N/2:OFFSET2+N/2+HV-1) = &
             X(0:N-1, OFFSET1:OFFSET1+HV-1) + &
             X(SHIFT2:SHIFT2+N-1, OFFSET1+HV:OFFSET1+VIEG-1)

      OFFSET1 = OFFSET1 + VIEG
      OFFSET2 = OFFSET2 + HV
      SHIFT1  = SHIFT1  - HV

   END DO

   XX(N:N2-1,:) = XX(0:N-1,:)

   X = XX

   GROUPS = GROUPS *2

END DO
```

References

[1] J.C. Adams, W.S. Brainerd, J.T. Martin, B.T. Smith & J.L. Wagener *FORTRAN 90 handbook, Complete ANSI/ISO Reference* Intertext Publications/McGraw-Hill Book Company, 1992.

[2] D.E. Dudgeon & R.M. Mersereau *Multidimensional Digital Signal Processing* Prentice-Hall, New Jersey, 1984

[3] M. Frigo *A Fast Fourier Transform Compiler* Proceedinds of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Atlanda, 1999

[4] M. Frigo & S.G. Johnson *The Fastest Fourier Transform in the West* MIT-LCS-TR-728, Massachusetts Institute of Technology, 1997

[5] M. Frigo & S.G. Johnson *FFTW: An Adaptive Software Architecture for the FFT* ICASSP conference proceedings, vol. 3, 1998

[6] C. Van Loan *Computational Frameworks for the Fast Fourier Transform* Frontiers in Applied Mathematics, SIAM, 1992.

[7] M.S. Pattichis *Novel Algorithms for Accurate, Efficient, and Parallel Computation of Multidimensional, Regional Discrete Fourier Transforms*, Proceedings of the 10th Mediterranean Electrotechnical Conference, Limassol, Cyprus, May 29-31, 2000.

[8] M.S. Pattichis, A.C. Bovik, J.W. Havlicek, and N.D. Sidiropoulos, *Multidimensional Orthogonal FM Transforms*, to appear in the IEEE Transactions on Image Processing.

[9] P.N. Swarztrauber *Vectorizing the FFTs, in Parallel Computations* (G. Rodrigue, ed.), Academic Press, pp. 51-83, 1982, http://www.netlib.org/fftpack