AN IMPROVED DIGITAL-REVERSED PERMUTATION ALGORITHM FOR THE FAST FOURIER AND HARTLEY TRANSFORMS

Liu Bin  Xie Ping
(Automation Department, Yanshan University, Qinhuangdao 066004)

Abstract A more efficient permutation algorithm which has less computer operation and better structure is presented here for radix-2 FFT(FHT). It can fasten the FFT and FHT efficiently when N becomes large.

Key words Radix-2 FFT and FHT; Permutation algorithm; Bit-reversed order; Upper-limit

I. Introduction

Radix-2 FFT and FHT have been used in a lot of digital signal processing, and more improved efficient algorithms created in recent years[1-3]. The common character of these algorithms is to do computation in situ to save inner memory, so it needs to do bit-reversed operation before or after transformation, and the performance of the permutation algorithm can also affect the efficiency of FFT or FHT greatly.

The most direct permutation algorithm is to do bit-reversing permutation[4], that is to obtain the “digit-reverse” of i as j, then swap the element $X[i]$ and $X[j]$. But it needs to do several arithmetical computations to obtain every j, and in fact it does not required to swap the $X[i]$ and $X[j]$ if $j < i$, and the computation like that is redundant. So the efficiency of this algorithm needs to be improved. A improved algorithm is presented in Ref.[5] which is to get the limit of the i that needs to be swapped, then to calculate the digit-reverse of the “i” in the limit range only, but it still has some redundant computation in the limit range like in Ref.[4]. A permutation algorithm for any radix-x transform is created in Ref.[6], which is to partition the i values(which ranges from 0 to $N - 1$) into equal sized groups at first, then to confirm if the swap of $X[i]$ and $X[j]$ is required or not by the position of “i” before calculate “j”, and to escape the calculation and swap if not required. But this algorithm needs to do more multiplication than others, for the calculation of every “j” of the first group’s “i” needs one multiplication.

Based on the algorithms in Refs.[5,6], the new improved algorithm here has been proved to have two advantages: (1) to confirm the limit range of “i” without computation. (2) to escape the multiplication when calculating the digit-reverse of “i” in the first group.

II. Theory Basis

Consider performing a fast radix-2 transform on the length $N(N = 2^n)$ data sequence stored in array $x[i]$, in which i ranges from 0 to $N - 1$, and j is the “digit-reverse” of i.

Not all $x[i]$ values need to be swapped with $x[j]$: if $i > j$, then the swap is unnecessary. So there is a upper-limit and low-limit of “i”, the swap is required only when the “i” is in
the limited range. The upper-limit \(i_{\text{sup}}\) and low-limit \(i_{\text{inp}}\) was obtained in Ref.\[5\] as

\[
i_{\text{sup}} = N - 1 - N_1, \quad i_{\text{inp}} = 1
\]  

(1)

where \(N_1 = 2^r\), \(r_1 = \text{TRUNC}\{(r + 1)/2\}\), and \(\text{TRUNC}(x)\) is a integer which is the largest but smaller than \(x\).

So the permutation needs to be executed for the element inside the limited range only, and not for the outside element. However, the condition that the \(i > j\) (that is to say the swap is redundant) also exists inside the limited range. So a improved algorithm is created in Ref.\[6\] as: to partition the \(i\) values of the data at first, then confirm if the calculation of \(j\) and the swap of \(X[i]\) and \(X[j]\) are necessary every time before do it. The detail is as follows:

At first, the \(i\) values which range from 0 to \(N - 1\) can be partitioned into \(m\) equal sized groups of contiguous \(i\) values. The size of each group is \(n\), and the zeroth group (the groups are labeled by \(0, 1, 2, \cdots, m - 1\)) is \(i = 0, \cdots, n - 1\), the first group is \(i = n, \cdots, 2n - 1\), and so on. The group label of each \(i\) value is \(f\), the member label in the group of each \(i\) value is \(e\), and the digit-reverse of each \(i\) is \(j\), where

\[
\begin{align*}
n &= \sqrt{N} = m, \quad \text{if } r \text{ is even} \\
 &= \sqrt{N}/2 = m/2, \quad \text{if } r \text{ is odd}
\end{align*}
\]  

(2)

where \(m = N/n\); \(f = 0, 1, \cdots, m - 1\); \(e = 0, 1, \cdots, n - 1\).

Then the \(j\) value of each \(i\) can be calculated by three proved theorems: First, \(j_0\) is named as the digit-reverse value of each member in the zeroth group \((i_0)\), and \(j\) as the other group’s. Then each \(j_0\) can be calculated as follows:

\[
 j_0 = n \times S_m[i_0]
\]  

(3)

where \(S_m[\cdot]\) is the digit-reversing function, which has \(m\) members; and \(i_0 = 0, 1, \cdots, n - 1\). So it needs to do a \(S_m[\cdot]\) operation to get \(S_m[i_0]\) and \(n\) times multiplication to get all \(j_0\) values.

Then to justify if the swap of \(X[i]\) and \(X[j]\) is required before calculating the \(j\) value by

\[
 f_i \leq S_m[e_i]
\]  

(4)

If Eq.(4) is reached, then the swap is unrequired, and the calculation of \(j\) is also avoidable; But if not, it is necessary. The equation is:

\[
 j = j_0 + S_m[f_i]
\]  

(5)

where \(j_0\) is the digit-reverse of \(i_0\) which has the same member label with \(i\), seen in Eq.(3).

III. Deducing of the New Algorithm

In order to get higher efficiency, two improving measures are given in the new algorithm based on the algorithms\[5,6\] we just described: (1) Simpler equations of the \(i_{\text{sup}}\) and \(i_{\text{inp}}\) are given here; (2) A new method to get the \(j_0\) is proposed which can avoid the multiplication in the former algorithm\[6\]. The details are as follows: