一种实现任意基 FFT 的快速整序算法

林水生* 黄顺吉
(电子科技大学电子工程系 成都 610054)

【摘要】 提出了一种数据整序快速算法,能对任意基 FFT 变换的数据进行快速整序。该算法对数据进行循环嵌套分组,简化了数据交换的判断条件,并减少了求解数据序号位倒序值的运算量。计算结果表明,当数据规模越大,该算法的数据整序时间较其他算法越少,并使基 2 - FFT 的运算时间较用其他整序算法时减少 1.3%~4%,较用直接整序方法时减少 7%~19%。

关 键 词 快速傅里叶变换; 位倒序; 分组整序; 快速整序算法
中图分类号 TN912.3

FFT 已广泛应用于数字信号处理中,无论是 Cooley - Tukey 算法还是 Sande - Tukey 算法,都需要对变换前的数据或变换后的结果进行整序。整序运算的快慢直接影响 FFT 的运算效率,所以数据的快速整序有利于减少 FFT 的运算时间。

某个数据序号 i 所对应的位倒序值 j,若 $i<j$,则将序号为 i、j 的两个数据相互交换;否则不交换。这种直接整序方法需要求出所有数据序号的位倒序值,运算量大。将数据分组的序号分组算法[1],既可简化数据交换的判断条件,又可减少序号位倒序值的运算量,经适当变换,还可将其中的部分乘法运算简化为加法运算[2]。序号分组算法较直接整序方法减少了运算量,但序号分组算法仍用常规的求解位倒序值的方法分别求解组号和组员的位倒序值,当数据量较大时,数据的整序时间仍然较长。

本文提出了一种快速整序算法,此算法首先对数据按序号进行分组,但数据交换的判断条件及位倒序值的求解不再依赖于组员的位倒序值,而只与组号的位倒序值有关。由于组号是按顺序递增的正整数,其位倒序值仍包含在其中,所以并不需要真正求解组号的位倒序值,而只需将其位置作适当调整,即组号的整序。组号的整序可通过组号的再次分组,分别得到子组号和子组员,使得组号的位倒序值通过子组号的位倒序值来求解。依次类推,子组号进一步分组、求解,直至最后的组号数为 B^2(B 为 FFT 的基),B^2 个正整数的倒序值很容易直接写出,即使采用常规位倒序值的求解方法,运算量也很小。

1 快速整序算法及实现

假设有 $N(N=B^n$,n 为正整数)个数据进行基 $B(B\geqslant 2)$的 FFT 运算,则需对 N 个数据进行整序。若数据序号 i 的 B 进制位倒序值为 j,且 $i<j$,则数据 $x(i)$与 $x(j)$需进行交换,否则,$x(i)$与 $x(j)$不交换。

如果 i 为

$$i=(k_{n-1},\cdots,k_s,\cdots,k_0)_B=k_{n-1}B^{n-1}+\cdots+k_sB^s+\cdots+k_0B^0 \tag{1}$$

1997 年 12 月 17 日收稿
* 男 31 岁 博士生 副教授

式中 $k_s=0,\cdots,B-1;s=n-1,\cdots,0;n=\log_B N$。则

$$j=R(i)_B=(k_0,\cdots,k_s,\cdots,k_{n-1})_B=k_0B^{n-1}+\cdots+k_sB^{n-s-1}+\cdots+k_{n-1}B^0 \tag{2}$$

式中 $R(i)_B$ 表示对 B 进制的数据 i 按位作倒序变换。

当且仅当 $i<j$ 时，需要交换数据，其次数为 $(N-B^{<n/2>})/2$（$<n/2>$ 表示不小于 $n/2$ 的最小正整数）。按交换数据的要求，求解 j 值的个数为 $(N-B^{<n/2>})/2$，然而从交换数据的判断条件来看，需要求解 $N-B^{<n/2>}-1$ 个 j 值[3]。如果有等价的交换数据的判断条件，使之不依赖于 j 值，就可使 j 值的求解个数减少，从而减少整序的运算量。

我们引用并改进文献[1]的定理。首先将 N 个数据按原顺序分成 $B^{<n/2>}$ 组，每组的数据个数为 $B^{[n/2]}$（$[n/2]$表示不大于 $n/2$ 的最大正整数）。若某个数据的序号 $i=(k_{n-1},\cdots,k_s,\cdots,k_0)_B=(gkm)_B$，则称 $(gk)_B$ 为数据的组号，$(m)_B$ 为数据的组员，g 和 m 的 B 进制位数相等，且当 n 为奇数时，$k=0,\cdots,B-1$；n 为偶数时，k 为空。

定理1 如果 $i=(gkm)_B$ 为第0组（即 $(gk)_B=0$）的某个数据序号，且 $(m)_B=(g')_B$，那么当 n 为奇数时，$j=R(i)_B=B^{[n/2]}R(0g')_B$；当 n 为偶数时，$j=R(i)_B=B^{[n/2]}R(g')_B$。

证明 因为 $(gk)_B=0$，那么

$$i=k_{[n/2]-1}B^{[n/2]-1}+\cdots+k_sB^s+\cdots+k_0B^0=(m)_B=(g')_B$$

$$j=R(i)_B=k_0B^{n-1}+\cdots+k_sB^{n-s-1}+\cdots+k_{[n/2]-1}B^{<n/2>}=B^{<n/2>}R(m)_B$$

当 n 为奇数时

$$R(m)_B=R(g')_B=R(0g')_B/B \qquad <n/2>=[n/2]+1$$

则

$$j=R(i)_B=B^{[n/2]}R(0g')_B$$

当 n 为偶数时 k 为空

$$R(m)_B=R(g')_B \qquad <n/2>=[n/2]$$

则

$$j=R(i)_B=B^{[n/2]}R(g')_B$$

定理2 如果 $i''=(g''k''m'')_B$ 为非0组（即 $(g''k'')B\neq 0$）的某个数据序号，且 $(m'')_B=(m)_B$，则 $j''=j+R(g''k'')_B$[1]。

定理3 对任意 $i=(gkm)_B,(m)_B=(g')_B$，如果 $(gk)_B<R(g'0)_B$，则需要交换 $x(i)$ 与 $x(j)=R(i)_B$；否则，不需要交换。

证明 由于存在 $(m)_B=(g')_B\Leftrightarrow(m0)_B=(g'0)_B\Leftrightarrow R(m)_B=R(m0)_B=R(g'0)_B$，则 $(gk)_B<R(g'0)_B$ 与 $(gk)_B<R(m)_B$ 是等价的。

经过改进后的三个定理都只与组号的位倒序值 $R(gk)_B$ 有关，而与组员的位倒序值 $R(m)_B$ 无关。并使得位倒序值的求解只需一次加法或一次乘法即可，因此运算量大大减少。

对于组号的位倒序值的求解，如果采用常规求解位倒序值的方法，当数据量较大时，运算量仍然较大，因此本文提出一种循环嵌套序号分组方法。把 $B^{<n/2>}$ 个组号看作 $B^{<n/2>}$ 个数据，并按原顺序给予编号。仍用 i、j 分别表示 $B^{<n/2>}$ 个数据的序号，$x(i)$、$x(j)$ 分别表示 $B^{<n/2>}$ 个数据，则有 $x(i)=i,x(j)=j$。若 $x(i)$ 的倒序值为 $x(j)$，则 $x(j)$ 的倒序值必为 $x(i)$，要求 $x(i)$、$x(j)$ 的倒序值，只需将它们互换即可。当然 $x(i)$ 与 $x(j)$ 互换时，需要满足条件 $x(i)<x(j)$，否则，两次互换又还原了。由于 $i=x(i),j=x(j)$，所以，当 $i<j$ 时，需要交换 $x(i)$ 与 $x(j)$，否则不需要交换，即求解 $B^{<n/2>}$ 个组号的倒序值的过程就是对 $B^{<n/2>}$ 个组号进行整序的过程。$B^{<n/2>}$ 个组号的整序仍可

用序号分组的方法，将 $B^{<n/2>}$ 个组号按序号分为 $B^{<n/4>}$ 个子组，每个子组的组员数为 $B^{[n/4]}$，则 $B^{<n/2>}$ 个组号的倒序值的求解归结为子组号的倒序值的求解。依次类推，直至最后的组号数为 B^2。B^2 个正整数的 B 进制位倒序值可以直接写出，或用常规求解位倒序值的方法求解。表 1 为实现这种算法的 C 语言程序。程序采用了循环嵌套调用结构，非常紧凑。由于 FFT 运算的数据为浮点，而数据序号为整数，所以子程序的类型转换是必不可少的。

快速整序算法程序如下：

```
#define  n  /* 指数 */    define  B  /* 基 */
int POWERB[n];
void main()      {int i;float x[POWERB[n]];
POWERB[0]=1; for(i=1;i<=n;i++)      POWERB[i]=B*POWERB[i-1];
order(n,x); /*FFT 变换*/ }

void order(int L,float x[])      {int g,m,i,j,a,b,t,y[POWERB[(L>>1)+1]];
m=POWERB[a=(L>>1)]; g=POWERB[b=L-a];
if(b==2)  for(i=0;i<B;i++)  {y[i]=i*B; for(j=1;j<B;j++)  y[i+j*B]=
y[i]+j;}
        else    for(i=0;i<g;i++)    y[i]=i;
if(b>2)    order(b,(float *)y);
for(i=1;i<m;i++)    {a=i; b=m*y[i]; swap(x,a,b);
                     for(j=1;j<y[i];j++)    {a+=m; t=b+y[j];
                     swap(x,a,t);}}}

void swap(float y[],int a,int b)    {float temp; temp=y[b]; y[b]=y[a]; y[a]=temp;}
```

2 算法分析与测试结果

本文提出的快速整序算法经过 $<\log_2 n>-1$ 次循环嵌套迭代就可完成 B^n 个数据的整序，所需的加法运算量略少于 $N-2B^{<n/2>}$，而乘法运算量略大于 $B^{<n/2>}$。直接求解 B 进制位倒序值的常规整序算法需要 $2BN/(B-1)$次加法运算和$(B+1)N/(B-1)$次乘法运算。当为基 2 - FFT 运算时，快速整序算法的加法运算不足常规算法的 1/4，乘法运算约为常规算法的 $1/(3\times 2^{[n/2]})$。快速整序算法的运算量与其他整序算法的运算量比较如表 1 所示。表 2 为三种整序算法在 SUN SPARC20 工作站上实现基 2 - FFT 运算的整序时间及其占 FFT 总运算时间的百分比。如果采用常规整序算法，基 2 - FFT 整序时间所占的比例高达 23%。运算结果还表明，在相同数据量时，快速整序算法的大基整序时间较小基整序时间略短，当数据为 4 096 时，快速整序算法的基 2、基 4、基 8 和基 16 - FFT 的整序时间分别为 1.704 ms、1.699 ms、1.681 ms 和 1.668 ms，而分组整序算法的整

序时间分别为 2.414 ms、2.388 ms、2.429 ms 和 2.470 ms。

表 1 三种整序算法的运算量比较表

	加法	乘法	数据交换	循环	条件分支
文献[1]	$N-\frac{B-3}{B-1}B^{\langle n/2\rangle}$	$\frac{2B}{B-1}B^{\langle n/2\rangle}$	$\frac{1}{2}N-\frac{1}{2}B^{\langle n/2\rangle}$	$\frac{1}{2}N+\frac{B+3}{2(B-1)}B^{\langle n/2\rangle}$	0
文献[2](基2)	$\frac{3}{2}N+\frac{5}{2}2^{\langle n/2\rangle}$	$2^{\langle n/2\rangle}$	$\frac{1}{2}N+\frac{3}{2}2^{\langle n/2\rangle}$	$\frac{1}{2}N+\frac{3}{2}2^{\langle n/2\rangle}$	$N-2^{\langle n/2\rangle}$
本文	$N-2B^{\langle n/2\rangle}$	$B^{\langle n/2\rangle}$	$N/2$	$N/2$	0

表 2 基 2 - FFT 的整序时间(ms)及百分比

n		6	7	8	9	10	11	12	13	14	15	16	17	18	19
文献[1]	t	0.039	0.077	0.149	0.296	0.582	1.164	2.414	5.279	10.88	22.20	46.48	95.06	376.5	833.8
文献[1]	%	9.6	8.7	7.7	7.0	6.3	5.9	5.4	5.2	5.0	4.8	4.5	4.2	5.4	4.1
文献[2]	t	0.049	0.100	0.191	0.384	0.747	1.521	3.192	6.762	13.62	27.40	59.13	126.2	431.0	901.6
文献[2]	%	11.8	11.0	9.7	8.9	8.0	7.2	7.0	6.6	6.2	5.8	5.6	5.6	6.1	4.3
本文	t	0.032	0.059	0.112	0.223	0.436	0.854	1.704	3.652	7.449	15.04	30.06	62.20	131.8	344.3
本文	%	7.9	6.8	6.0	5.4	4.8	4.2	4.0	3.8	3.6	3.5	3.2	3.1	3.1	2.8

3 结 论

本文提出的快速整序算法能实现任意基 FFT 变换的数据整序，由于采用了循环嵌套序号分组的方法，减少了求解位倒序值的运算量。当数据量较大时，整序时间的减少更为明显，从而更加提高了 FFT 的运算效率。这种快速整序算法也可用于其他需要整序的快速算法中。

参 考 文 献

1 Evans D M W. An improved digit-reversal permutation algorithm for the fast Fourier and Hartley transforms. IEEE Trans on ASSP, 1987, 35(8): 1 120 ~ 1 125

2 刘 彬，谢 平. FFT 或 FHT 的一种改进的整序算法. 电子科学学刊，1997，19(3)：306 ~ 310

3 曹 钧. 提高快速傅里叶变换算法效率的方法. 微电子学与计算机，1984(5)：13 ~ 15

A Fast Digit-reversal Permutation Algorithm for Radix-B FFT

Lin Shuisheng Huang Shunji

(Dept. of Electronic Eng., UEST of China Chengdu 610054)

Abstract A fast digit-reversal permutation algorithm for the radix-B fast Fourier transforms(FFT) is presented in this paper, which decreases the computation of the "digit-reversing" and speeds up the FFT by loop nesting dividing the datas into small groups. According to timing experiments, the fast permutation algorithm significantly hastens the digit-reversal permutation and saves the FFT running time by 1.3% ~ 4%, which is 7% ~ 19% shorter than that of other permutation algorithm.

Key words fast Fourier transforms; digit-reversing; group permutation; fast permutation algorithm

编辑 黄 辛