第45卷第7期 中国科学技术大学学报 Vol. 45, No. 7
2015年7月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jul. 2015

文章编号:0253-2778(2015)07-0608-06

# 面向高数据并行架构的原位FFT算法

王向前[1],郑启龙[3],王 昊[2],洪 一[1,2],张 磊[1]

(1.合肥工业大学计算机与信息学院,安徽合肥,230009;
.中国电子科技集团公司第三十八研究所,安徽合肥,230088;
3.中国科学技术大学计算机科学与技术学院,安徽合肥,230027)

**摘要**:数字信号处理器的内存较小,而且数字信号处理领域的应用往往是数据密集型,这要求在设计数字信号处理应用算法时既要考虑时间复杂度又要兼顾算法的空间复杂度.为此提出了一种原位的逆序算法;针对数字信号处理器比较高的内存访问并行度,设计了部分逆序的原位高效FFT算法;并在魂芯DSP平台上实现了该算法框架.实验表明,与非原位FFT算法相比,该原位算法的空间复杂度大幅降低而时间效率的损失在可接受范围之内.

**关键词**:逆序;原位;FFT;空间复杂度;时间复杂度

**中图分类号**:TN957 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2015.07.012

**引用格式**:WANG Xiangqian, ZHENG Qilong, WANG Hao, et al. An in-place FFT algorithm for high data parallelism architecture[J]. Journal of University of Science and Technology of China, 2015,45(7):608-613.

王向前,郑启龙,王昊,等. 面向高数据并行架构的原位FFT算法[J]. 中国科学技术大学学报,2015,45(7):608-613.

# An in-place FFT algorithm for high data parallelism architecture

WANG Xiangqian[1], ZHENG Qilong[3], WANG Hao[2], HONG Yi[1,2], ZHANG Lei[1]

(1. *School of Computer and Information, Hefei University of Technoogy, Hefei* 230009, *China*;
2. *No.38 Research Institude, China Electronics Technoogy Group Corporation, Hefei* 230088, *China*;
3. *School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*)

**Abstract**: The on-chip memory in DSP is small, and applications for DSP are often data-intensive, which requires that space complexity as well as time complexity must be considered when algorithms are designed. So a in-place bit reverse algorithm was proposed. Then, to take advantage of memory bandwidth offered by DSP, an effective in-place FFT algorithm with part bit reverse was designed and implemented on BWDSP. Experiment result shows that, compared with the out-of-place FFT algorithm, its space complexity is significantly reduced, while the loss of time efficiency for the proposed in-place FFT algorithm is acceptable.

**Key words**: bit reverse; in-place; FFT; space complexity; time complexity

## 0 引言

数字信号处理器[1-2]的内存一般不大,为了处理数据密集型的应用,在设计算法时必须考虑空间复杂度.FFT算法是数字信号处理领域的一个重要的算法.它分为时域抽取算法和频域抽取算法,计算过

**收稿日期**:2014-10-17;**修回日期**:2015-04-18

**作者简介**:王向前,男,1985年生,博士生. 研究方向:编译优化与系统软件优化. E-mail:forward@mail.ustc.edu.cn

**通讯作者**:郑启龙,博士/副教授. E-mail:qlzheng@ustc.edu.cn

程都涉及逆序计算.传统的非原位的 FFT 算法的空间复杂度为 O(2 * N),单位为复数,其中 N 为 FFT 点数;原位 FFT 的空间复杂度为 O(N),与非原位 FFT 相比,可以节约 N 个复数所占的内存空间.

FFTW[3]是麻省理工学院开发的一种硬件体系结构自适应的 FFT 优化代码生成器,它能自动适应多种硬件体系结构,因而可移植性很强,但针对高数据并行的 DSP 体系结构,它并未做过多考虑;文献[4]针对 DSP 提供的强大的乘累加计算能力等体系结构特征,提出了一种高效的 FFT 算法;文献[5]利用逆序循环针对 FFT 逆序算法展开优化研究,取得了比较好的优化效果,但并未对原位逆序算法以及面向高数据并行能力的逆序算法进行系统的研究;文献[6]研究在 ADI 公司的 TigerSHARC DSP 平台的 FFT 算法的实现,选择基 4 来减少蝴蝶变换的个数,利用该 DSP 提供的数据并行能力,并且提出通过位反序表的方法实现位反序,提高逆序效率;文献[5]针对不同的 GPU 平台提供了一个自动性能调优的 FFT 优化框架 MPFFT[7],可以充分为 GPU 提供多级并行性,较好地利用了 GPU 的体系结构特征以及 FFT 本身所蕴含的并行性.

## 1　一种原位逆序算法

逆序[8-10](也叫位反序)是指一个数的二进制形式对应位的置换.假如一个数的二进制形式是($x_n$…$x_1$ $x_0$),则它的逆序数为($x_0$ $x_1$…$x_n$).通过数据置换可以实现逆序过程,基于数据置换的逆序算法推导过程如下.

推导步骤一:假定 A 点 FFT 逆序的方法已知,则获得 4 * A 点 FFT 逆序的途径可以通过将 4 * A 点转换成四个子序列块序列的{0A0,0A1,1A0,1A1}的整体逆序;整体逆序完成之后,再对四个序列块的对应 A 点 FFT 用已知方法进行处理.整块逆序是通过图 1 所示的方法,进行整体置换.

推导步骤二:步骤一是可递归的.由 A 点已知的逆序方法既可以知道 4 * A 点的逆序方法,也可以知道 16 * A(即 4 * (4 * A))点的逆序方法.如图 2 所示,将 16 个序列块中首位和末位相同的序列块依次组合成四个序列块{0B0,0B1,1B0,1B1};用推导步骤一的方法对该四个序列块进行整体置换;然后对四个序列块中每个 B 序列块用推导步骤一的方法再进行整体置换;最后再用已知的方法对 A 进行变换.

推导步骤三:当序列为{a0,a1,a2,a3}时,它的逆序序列为{a0,a2,a1,a3};当序列为{a0,a1,a2,a3,a4,a5,a6,a7}时,它的逆序序列为{a0,a4,a2,a6,a1,a5,a3,a7};这两个简单的事实是步骤一和步骤二递归的终止条件;由此设计基于置换的逆序算法.

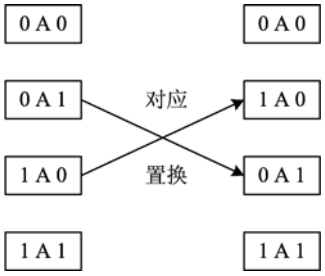


**图 1　整体逆序示意图**

**Fig. 1　Illustration of whole bit reverse**

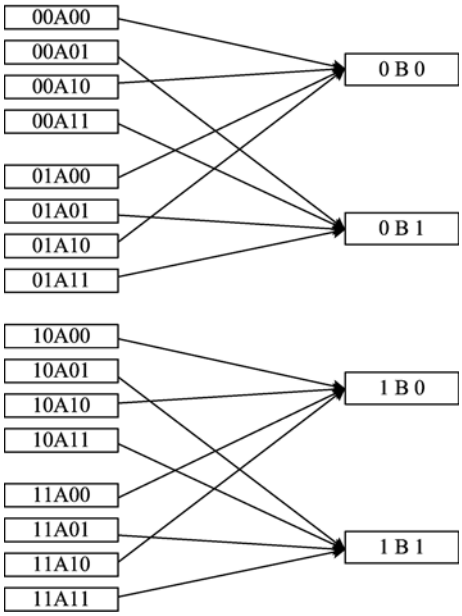


**图 2　数据块组合示意图**

**Fig. 2　Illustration of data block composition**

**算法 1.1**　原位逆序算法

```
Procedure Inplace_BitReverse(Array,R) Array:Complex R:unsigned int
//Array 为待逆序序列,R 为该序列的阶数,即 R=log2 N,其中 N 为序列大小
begin
N,NN,step: unsigned int
N := 1<<R
step := 1
for(r := R; r>=2;r--2)
  NN := N/step
```

```
  for(k := 0;k < N;k+=NN)
    for(i :=0; i<step;i++)
      u1 = i + step + k
      u2 = NN/2 + i + k
      for(j := 0;j < NN/2;j += 2 * step)
        temp1 := Array[u1]
        temp2 := Array[u2]
        Array[u1] := temp2
        Array[u2] := temp1
        u1 := u1 + 2 * step
        u2 := u2 + 2 * step
      od
    od
  od
  step := step * 2
od
end
```

假定最内层循环的一次置换为基本单位，则算法复杂度为 $O((R/2)*(N/4))$次，其中 $R/2$ 为整体置换次数. 当处于 1 024 点时，置换次数为 $5\times256=1\ 280$ 次；与非原位逆序运算相比，该原位逆序运算节约 1 022 个空间，但整体置换次数也多了$((R/2)-1)=4$ 次. 此时，FFT 算法的空间复杂度降低到最小，只需要 2 个单位的临时寄存器，即完成了整个原位逆序过程.

## 2 基于部分逆序的 FFT 算法设计

### 2.1 面向高数据并行的部分逆序算法

与非原位逆序相比，第 1 节的提出的原位逆序算法所需要的空间复杂度很小，而时间复杂度则较高. 考虑到 DSP 处理器提供的连续地址空间的高数据访存并行性，充分利用 DSP 提供的连续若干个地址的数据访问能力，结合 FFT 旋转因子访问的特点，可以考虑部分逆序. 部分逆序是指保持逆序数的高/低若干位不变，其他位数则对应逆序，如图 3 所示.

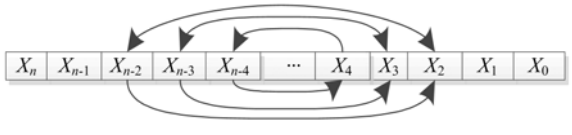


**图 3 高/低 2 位不参与逆序的部分逆序示意图**

**Fig. 3 Illustration of part bit reverse(part = 2)**

假定 DSP 处理器的连续多地址的数据访问能力为 $B$ 个复数字，则部分逆序位数至少为 $\log_2 B$，即保持高低 $\log_2 B$ 位不变，中间其他位数对应逆序. 基于第 1 节中提出的原位逆序算法，设计针对部分逆序的块原位逆序算法.

**算法 2.1** 部分逆序的原位算法

```
Procedure Inplace_PartBitReverse(Array,R,part) Array:Complex
R:unsigned int part:unsigned int
//Array 为待逆序序列,R 为该序列的阶数,即 R=log2 N,其中 N 为序列大小
    //part 为高多少位/低多少位不参与逆序位数
begin
N,NN,innerN,Step,partR: unsigned int
blockLoop,partSize: unsigned int
N := 1<<R
partsize := N>>part
innerN := 1<<(R-2 * part)
for(partLoop := 0;partLoop <(1<<part);partLoop++)
  step := 1
  partR := R-2 * part
  for(r :=partR ; r>=2;r--=2)
    NN := innerN/step
    for(k := 0;k < innerN;k+=NN)
      for(i :=0; i<step;i++)
        u1 = i + step + k
        u2 = NN/2 + i + k
        for(j := 0;j < NN/2;j += 2 * step)
          for(blockLoop := 0;blockLoop <(1<<part);
blockLoop++);
            temp1 = Array[u1 * (1<<part)+
blockLoop+partLoop * partSize]
            temp2 = Array[u2 * (1<<part)+
blockLoop+partLoop * partSize]
            Array[u1 * (1<<part)+blockLoop+
partLoop * partSize] = temp2
            Array[u2 * (1<<part)+blockLoop+
partLoop * partSize] = temp1
          od
        u1 := u1 + 2 * step
        u2 := u2 + 2 * step
      od
    od
  od
  step := step * 2
od
end
```

该算法的思想如下：首先把输入序列按照位序的高 part 位连续划分成为 $2^{part}$ 个子输入序列；针对这 $2^{part}$ 个子输入序列，位序的低 part 位作为一个整体块参与置换；此时，部分逆序转化为求解针对每个子序列($R-2*part$)个中间位的逆序操作，可利用第 1 节提出的原位逆序算法完成该逆序过程. 每个子序列完成该逆序过程，输入序列就完成了部分逆序过程. 完成部分逆序序列位序的高 part 和低 part 位保持不变，而中间其他位则完成了部分逆序. 该算法时间复杂度为 O(((R－2 * part)/2) *)，单位为两个复数的置换操作.

### 2.2　基于部分逆序的 FFT 算法设计

在设计 FFT[11-13] 算法时，针对具体计算过程，有两种情况，一是原序输入，逆序输出，最后完成逆序过程；一是逆序输入，原序输出，是在计算前完成的逆序过程. 而如果是部分逆序输入，在计算完成时，则需要最终完成高/低 part 位的逆序过程.

在计算过程中涉及取数规律问题. 如表 1 所示，以 128 点 FFT 的高/低 2 位部分逆序为例，说明基于部分逆序时各阶运算的寻址步长规律. 128 点 FFT 运算时，假如是原序输入的算法，则参与蝶形运算的复数对对应寻址步长依次为 64,32,16,8,4,2,1；假如是逆序输入的算法，则参与蝶形运算的复数对对应寻址步长依次为 1,2,4,8,16,32,64；假如是部分逆序(part＝2 位)输入，则参与蝶形运算的复数对应的寻址步长为 64,32,4,8,16, 2,1. 可见，部分逆序的寻址步长是有规律的，即高/低 2 阶的寻址步长和原序输入的高/低 2 阶的寻址步长是一致的，中间三阶的寻址步长和逆序输入的中间三阶的寻址步长是一致的. 这个规律和高/低 2 位保持原序而中间位已经完成逆序的事实相符.

**表 1　128 点 FFT 部分逆序的寻址步长表**

**Tab. 1　Address step table in 128-dot FFT for part bit reverse**

| 阶数 | 0 阶 | 1 阶 | 2 阶 | 3 阶 | 4 阶 | 5 阶 | 6 阶 |
|---|---|---|---|---|---|---|---|
| 原序 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 逆序 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 部分逆序 | 64 | 32 | 4 | 8 | 16 | 2 | 1 |

部分逆序后输入序列在内存的位序如图 4 所示. 该图是高/低位为 2 的部分逆序后的序列. 输入序列被等分成为高位分别为(00,01, 10,11)四个连续的子序列，其中 A, B, C, …，为自然位序为 $2^{R-2*part}$ 个输入序列的逆序位序. 可见完成最终的逆序过程只需依次遍历 A, B, C, …，对中间($R-4$)位位序相等的四组数据(00A00, 00A01, 00A10, 00A11)，(01A00, 01A01, 01 A10, 01A11)，(10A00,10A01,10A10,10A11)，(11A00, 11A01, 11A 10,11A11)进行对应的数据置换即可完成最终的逆序过程. 即，第一组的 00A00 的高/低 2 位逆序是其自身，不需要交换；第一组的 00A01 和第三组的 10A00 进行置换，因为两者各为对方的高/低两位逆序数等.

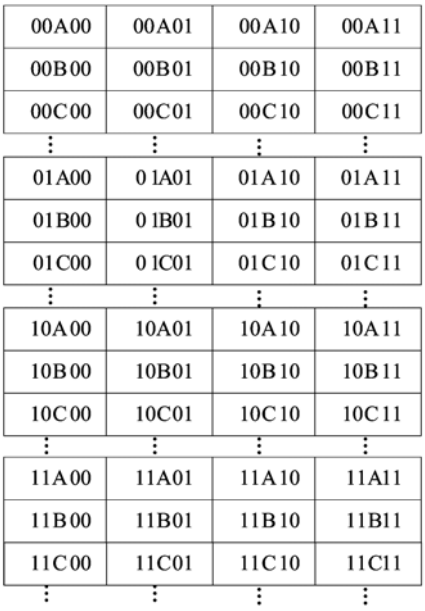


**图 4　部分逆序后的输入序列位序状态**

**Fig. 4　Order status for input data after part bit reverse**

## 3　针对魂芯 DSP 的 FFT 算法实现

魂芯 DSP(BWDSP)是一款高数据并行、分簇结构($x/y/z/t$ 四个簇)、支持 SIMD 的 VLIW 结构的浮点运算信号处理器. 它具有非常强大的浮点运算功能，其单核单周期内能够完成 8 次浮点加减法及 4 次浮点乘法运算；同时也有着较为强大的数据访问能力. 其单核单周期可以读取两个浮点复数，存储一个浮点复数. 由于单个形如$(a+bj)\pm(c+dj)*(Wr+Wij)$的蝶形运算只需要 4 次乘法和 6 次加减法即可完成. 由此可以看出，魂芯 DSP 芯片单核可以在一个周期内实现一次浮点蝶形运算. 为了达到单周期实现单个蝶形运算的目的，要求芯片能够支持单周期内读取三个复数(两个运算数、一个旋转因子)，单周期内存储两个复数(两个运算结果)，而这

已经超出了魂芯DSP的访存能力.如果采取两阶合并策略的话,两个周期内要求芯片的访存能力降低为:读取四个复数(两个运算数、两个旋转因子),存储两个复数(两个运算结果),则可以满足这一要求,所以在设计针对魂芯DSP的高效FFT算法框架时,各处理循环至少为两阶合并以充分发挥芯片性能.

FFT算法的前两级为特殊旋转因子,单个蝶形运算只需要4次加减法.这样前后两级蝶形运算共8次加减法,一个周期即可完成.当再次遇到了数据访存能力不足的问题时,继续采用多阶合并的解决办法.由于魂芯DSP芯片单核具有64个寄存器,合并阶数最高可以达到4阶.考虑到支持32点FFT点数,如果前几阶按4阶合并,则其最后只有独立1阶,无法实现最优,故而采用判断奇偶阶的处理办法.对于偶数阶(最小点数为64点),采用4阶合并;对于奇数阶(最小点数32点),采用3阶合并.

由于支持连续8个地址的字存取[14],所以部分逆序位数为 $\log_2(8/2)=2$,即采取高/低位为2的部分逆序算法.综上所述,设计高效的基于部分逆序的针对的高效FFT算法如下.

**算法3.1** 魂芯DSP基于部分逆序的FFT算法

```
Procedure FFT(Array,R) Array:Complex R:unsigned int
//Array为待逆序序列,R为该序列的阶数,即R=log2N,
                其中N为序列大小
begin
r: unsigned int
//部分逆序
Inplace_PartBitReverse(Array,R,2)
if(R & 1 := 0)
    前三阶合并
    r := r-3
fi
if(R & 1 ! = 0)
    前四阶合并
    r := r-4
fi
//如果不是最后两阶循环,则进行两阶合并的循环
for(;r>2;r--=2)
    两阶合并循环
od
//最后两阶循环
for(i:=0;i<(1<<R-4);i++)
    进行最后两阶FFT运算
    四组数据内部交换完成最终逆序
od
end
```

算法首先对输入序列进行部分逆序;然后判断FFT输入序列是否为奇数阶,若是奇数阶,则进行三阶合并,即从内读取序列到数字信号处理器的寄存器进行前三阶运算之后,写回内存;若是偶数阶,则进行四阶合并,即从内存读取序列到数字信号处理器的寄存器进行前四阶运算之后,写回内存;然后除去最后两阶的两阶合并循环处理,从内存读取序列和旋转因子到数字信号处理器寄存器的寄存器进行两阶运算后,写回内存;最后进行最后两阶的运算以及最终的高/低2位的逆序.

FFT的算法的时间复杂度为 $N/2*\log_2(N)$次蝶形运算.设计的FFT算法经过阶合并后魂芯DSP的单核每周期可以进行一个蝶形运算,而魂芯DSP有四个核心,故不考虑部分逆序的时间复杂度,该算法的时间复杂度为 $O(N/8*\log_2(N))$.由于部分逆序的时间复杂度为 $O(((R-2*\text{part})/2)*2^{R-\text{part}-2})=O(((R-4)/2)*2^{R-4})$,所以该算法的时间复杂度为 $O(N/8*\log_2(N))+O(((R-4)/2)*2^{R-4})$,空间复杂度为 $O(N)$.如果采用非原位的部分逆序的FFT算法,考虑到魂芯DSP有从硬件上支持逆序的指令,在实现非原位的部分逆序过程的时间则可以忽略不计,即非原位的部分逆序的FFT算法的时间复杂度为 $N/8*\log_2(N)$,空间复杂度为 $O(2*N)$.

与非原位的部分逆序的FFT算法相比,基于置换的原位部分逆序的FFT算法的性能降低的百分比为:$O(((R-4)/2)*)/O(N/8*\log_2(N))=(R-4)/4R$(在魂芯DSP平台上 $R$ 的取值范围为[5,17]).

由此可见,随着FFT点数的增大,基于置换的原位部分逆序的FFT算法效率也随之降低,降低的百分比是从 $R=5$(32点FFT)的5%,逐步增大到 $R=17$(131072点FFT)的20%.而空间复杂度则为 $O(N)$.

尤其是在大点数时,$R=17$(131072点FFT)时,虽然时间性能降低了20%,但从空间复杂度讲则节约了131072*8bytes = 1M bytes内存空间.相比于算法空间需求的大幅降低,算法时间效率降低20%是可接受的.考虑到信号处理器较小的内存空间,算法空间复杂度的降低对于无疑是重要的.一方面,节约了信号处理器的内存空间;另一方面,意

味着可以直接支持更大点数[15-16]的 FFT.

两种算法在魂芯 DSP 平台的测试数据如表 2 所示.其中“非原位”是指算法 3.1 中的部分逆序占用 O(2 * N)个空间完成的非原位 FFT 算法.“原位”表示基于块原位部分逆序的 FFT 算法.从表 2 可以看出,实测的性能与理论分析基本相符.

**表 2　魂芯 DSP 平台上基于部分逆序的 FFT 算法性能(单位:周期数)**

**Tbl. 2　The Performances for FFT Algorithm based part bit reverse on BWDSP(unit:cycles)**

| FFT 点数 | 1024 点 | 2048 点 | 4096 点 | 8192 点 |
|---|---|---|---|---|
| 非原位 | 1176 | 2688 | 5532 | 12430 |
| 原位 | 1432 | 3247 | 6497 | 14886 |
| FFT 点数 | 16384 点 | 32768 点 | 65536 点 | 131072 点 |
| 非原位 | 25480 | 57502 | 117912 | 264780 |
| 原位 | 30182 | 69927 | 147352 | 326751 |

## 4　结论

本文提出一种通过置换实现的原位逆序算法,并结合 DSP 信号处理器连续数据的高数据并行能力,提出了基于置换的部分逆序算法;在此基础上,针对魂芯 DSP 的大寄存器文件特征和高数据并行能力,提出了通过阶合并和部分逆序的方法实现该原位 FFT 算法.经过理论分析和实际性能的测试,该算法在大幅优化空间需求的前提下,性能降低在可接受范围之内,表明该算法是一种高效的折衷算法.

**参考文献(References)**


[1]胡广书. 数字信号处理:理论,算法与实现[M]. 北京:清华大学出版社,2003.

[2]吴强,任琳,张杰,等. 快速归一化互相关算法及 DSP 优化实现[J]. 电子测量与仪器学报,2011,25(6):495-499.
WU Q,REN L,ZHANG J,et al. Fast algorithm of normalized cross correlation and optimized implementation on DSP[J]. Journal of Electronic Measurement and Instrument,2011,25(6):495-499.

[3]Frigo M,Johnson S G. FFTW:An adaptive software architecture for the FFT[C]//Proceedings of the International Conference on Acoustics,Speech and Signal Processing. Seattle,USA:IEEE Press,1998,3:1381-1384.

[4]Mikami N,Matsuoka M. A new DSP-oriented FFT algorithm[J]. Electronics & Communications in Japan,1989,72(5):102-108.

[5]方志红,张长耀,俞根苗. 利用逆序循环实现 FFT 运算中倒序算法的优化[J]. 信号处理,2004,20(5):533-535.

[6]Chen Z W. Realization of Radix-4 FFT algorithm based on TigerSHARC DSP[J]. Lecture Notes in Electrical Engineering,2012,128:41-46.

[7]Li Y,Zhang Y Q,Liu Y Q,et al. MPFFT:An auto-tuning FFT library for OpenCL GPUs[J]. Journal of Computer Science and Technology,2013,28(1):90-105.

[8]Drouiche K. A new efficient computational algorithm for bit reversal mapping[J]. IEEE Transactions on Signal Processing,2001,49(1):251-254.

[9]Prado J. A new fast bit-reversal permutation algorithm based on a symmetry[J]. Signal Processing Letters,2004,11(12):933-936.

[10]Huang J S,Chen S,Chen C X,et al. A novel approach based on recursive indexing for FFT data reordering[C]// IET International Conference on Information and Communications Technologies. Beijing,China:IEEE Press,2013:394-398.

[11]Ayinala M,Lao Y J,Parhi K K. An in-place FFT architecture for real-valued signals[J]. IEEE Transactions on Circuits and Systems,2013,60(10):652-656.

[12]吴争,李辉. 高速并行 FFT 的 FPGA 实现[J]. 无线电工程,2013,43(10):16-18.

[13]李晓雯,崔翔. GPU 矩阵乘法和 FFT 算法的性能优化[J]. 现代电子技术,2013,36(4):80-84.

[14]Yu F,Ge R F,Wang Z K. Efficient Utilization Of Vector Registers To Improve FFT performance on SIMD microprocessors[J]. IEICE Transactions on Fundamentals of Electronics,Communications and Computer Sciences,2013,96(7):1637-1641.

[15]马潇,高立宁,刘腾飞,等. 基于 Cache 优化的大点数 FFT 在 TS201 上的实现[J]. 电子与信息学报,2013,35(7):1774-1778.

[16]高立宁,马潇,刘腾飞,等. 基于超大点数 FFT 优化算法的研究与实现[J]. 电子与信息学报,2014,36(4):998-1002.