王书盈,胡勇华,张鑫,等.定点 FFT 的 DSP 向量混洗并行基 4 算法[J].湖南科技大学学报(自然科学版),2024,39(2):75-86.doi:10.13582/j.cnki.1672-9102.2024.02.010

WANG S Y, HU Y H, ZHANG X, et al. Vector Shuffling Based Parallel Radix-4 FFT Algorithm for Fixed Point Data on Vector DSP[J]. Journal of Hunan University of Science and Technology (Natural Science Edition),2024,39(2):75-86.doi:10.13582/j.cnki.1672-9102.2024.02.010

定点 FFT 的 DSP 向量混洗并行基 4 算法

王书盈,胡勇华*,张鑫,陆浩松

(湖南科技大学 计算机科学与工程学院, 湖南 湘潭 411201)

摘 要:基于定点数据的快速傅里叶变换(Fast Fourier Transform, FFT)算法能在合理的精度范围内降低对硬件的要求,计算速度更快.文章面向高性能向量数字信号处理器(Digital Signal Processor, DSP)的硬件特征,构建基 4 复数 FFT 算法的高效指令级并行处理模型.该模型充分考虑基 4 方法下的复数 FFT 计算过程和蝶形组集合的特征,将 SIMD 计算、向量混洗、索引 DMA 等技术与复数 FFT 的基 4 变换过程充分融合,有效控制计算过程中存储器和片内缓存之间的数据块搬移需求,提升 SIMD 计算单元的利用率.在基于自主 YHFT-M7002 处理器的 FT-M7002DSK 平台上进行试验研究,验证算法的有效性.试验结果表明:与 CCS 模拟所得 TI 的相应 TMS320C6678 库函数性能相比,所提优化算法的平均加速比达到 TI 库函数的 4.79倍.

关键词:基 4 复数 FFT; SIMD 技术; 向量 DSP; 向量混洗; 索引 DMA

中图分类号: TP311 **文献标志码**:A **文章编号**:1672-9102(2024)02-0075-12

Vector Shuffling Based Parallel Radix-4 FFT Algorithm for Fixed Point Data on Vector DSP

WANG Shuying, HU Yonghua, ZHANG Xin, LU Haosong

(School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China)

Abstract: The Fast Fourier Transform (FFT) algorithm of fixed-point data can reduce the hardware requirements within a reasonable accuracy range but obtain faster computing speed. Based on the hardware characteristics of high performance vector Digital Signal Processors (DSPs), this paper constructs an efficient instruction level parallel processing algorithm of Radix-4 complex FFT algorithm. This algorithm considers the calculation process of the Radix-4 complex FFT algorithm and the characteristics of butterfly units, and it fully integrates SIMD calculation, vector shuffling, indexing DMA and other techniques with the transformation process of Radix-4 complex FFT. This algorithm effectively controls the data block movement between memory and in-chip cache during the computing process and improves the utilization rate of SIMD processing unit. In this paper, an experimental study is conducted on the FT-M7002DSK platform of the YHFT-M7002 processor, which has independent intellectual property rights. Result shows that, compared with the performance obtained by CCS simulator for the corresponding TMS320C6678 library function, the average performance of our algorithm is 3.79 times faster than the former.

Keywords: Radix-4 complex FFT; SIMD technology; vector DSP; vector shuffling; index DMA

收稿日期:2022-01-30 **修改日期**:2023-11-26

基金项目:湖南省自然科学基金资助项目(2023JJ50019);湖南省教育厅科研项目资助(20B242;19A169)

* **通信作者**,E-mail: huyh@hnust.cn

快速傅里叶变换(Fast Fourier Transform, FFT)是一种利用数字计算机进行功率谱分析和滤波器仿真等信号分析的计算工具[1].自 COOLEY 等[2]提出 FFT 算法(基 2-算法)以来,FFT 算法在科学研究和工程计算等领域得到广泛应用,如频谱分析[3]、通信系统[4]、图像处理[5]、雷达处理[6]、语音识别[7]和结构健康检测[8]等.

随着互联网技术的高速发展,数据信息爆炸式增长,大规模的数据运算在数字信号处理(Digital Signal Processor, DSP)等领域大量存在,SIMD 架构通过同时对寄存器中的多个数据执行相同的操作来实现数据的并行运算,现已成为各类向量处理器的重要组成部分,如龙芯处理器、魂芯处理器[9]、银河飞腾系列处理器[10]和 TI 系列处理器等.同时,该类处理器结合超长指令字(Very Long Instruction Word, VLIW)架构,两者相辅相成,使每条指令都能高效运转,从而得到更高的运算效率[11].因此,越来越多的研究者开始研究 FFT 算法在 DSP 向量处理器上的实现与优化.其中,郭利财等[12-16]在特定的处理器上进行 FFT 算法的并行化研究;陈海燕等[17]基于银河飞腾系列处理器对 FFT 算法的存储结构进行优化,均取得了很好的效果.此外,一些研究人员也在 X86 架构[18-19]和 ARMv8 架构[20-21]相关平台上结合 SIMD 架构进行 FFT 算法的并行优化研究,算法的性能均得到提升.然而,以上研究提出的思路和优化方法都是针对特定的处理器,没有很好的移植性和扩展性.

基于定点数的快速傅里叶变换能在合理的精度范围内降低对硬件的要求,获得更快的计算速度,能应用于图像处理、低精度数值分析等方面.本文针对具有支持向量访问的片内高速缓存向量处理器特征,根据基 4 定点 32 位 FFT 的基本原理,研究一种面向向量 DSP 结构的基 4 定点 32 位 FFT 通用算法.该算法针对向量 DSP 的体系结构和硬件资源特点,在 FFT 计算中采用向量 SIMD 运算单元、混洗和循环展开等方式提升 FFT 的运算效率.

1 FFT 的频率抽取算法基本原理

本文基 4 定点复数 FFT 计算通过使用频率抽取的方式实现.根据 FFT 的计算原理,每一个点的计算公式[22]如式(1)所示.

$$X(k)=\sum_{n=0}^{N/4-1}x(n)w_N^{nk}+\sum_{n=N/4}^{N/2-1}x(n)w_N^{nk}+\sum_{n=N/2}^{3N/4-1}x(n)w_N^{nk}+\sum_{n=3N/4}^{N-1}x(n)w_N^{nk}. \tag{1}$$

式中:$X(k)$为结果数据;$x(n)$为源数据;k 和 n 为整数,其取值都为 0,1,…,N;N 为 4 的整数倍;w_N^{nk} 为旋转因子.

将式(1)进一步提取可得

$$X(k)=\sum_{l=0}^{3}w_N^{lk}\sum_{n=0}^{(N/4)-1}x(4n+l)w_N^{nk}. \tag{2}$$

令 r 为整数,其取值为 0, 1, 2, 3, …, $N/4$,式(2)变成如下 4 个公式:

$$X(4r)=\sum_{n=0}^{N/4-1}\left[x(n)+x\left(n+\frac{N}{2}\right)+x\left(n+\frac{N}{4}\right)+x\left(n+\frac{3N}{4}\right)\right]w_{N/4}^{nk}; \tag{3}$$

$$X(4r+1)=\sum_{n=0}^{N/4-1}\left\{x(n)-x\left(n+\frac{N}{2}\right)-j\left[x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right]\right\}w_N^n w_{N/4}^{nk}; \tag{4}$$

$$X(4r+2)=\sum_{n=0}^{N/4-1}\left[x(n)+x\left(n+\frac{N}{2}\right)-x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right]w_N^{2n}w_{N/4}^{nk}; \tag{5}$$

$$X(4r+3)=\sum_{n=0}^{N/4-1}\left\{x(n)-x\left(n+\frac{N}{2}\right)+j\left[x\left(n+\frac{N}{4}\right)-x\left(n+3\frac{N}{4}\right)\right]\right\}w_N^{3n}w_{N/4}^{nk}. \tag{6}$$

式中:$X(4r)$, $X(4r+2)$, $X(4r+1)$和 $X(4r+3)$为计算所得的 4 个结果数据,它们均由源数据 $x(n)$, $x(n+N/2)$, $x(n+N/4)$ 和 $x(n+3N/4)$ 决定.按上述公式,一个 N 点的离散傅里叶变换(Discrete Fourier Transform, DFT)可以分解为 4 个 $N/4$ 个点的 DFT 运算和它们结果的蝶形运算.

上述分解可以继续进行下去,直到参与每个 DFT 运算的源数据只有 4 个为止.每一次分解后都要对全部的数据进行一次基 4 蝶形运算,可以进行这种分解的次数决定了基 4 FFT 运算的级数,这是基 4 FFT 算

法的核心.将第 $L(L=1, 2, \cdots)$ 次分解后的每一组参与 DFT 运算的数据整体作为一个蝶形组,令该组的数据量为 N_L,则该组中的蝶形数量为 $N_L/4$.在这种方案下,从第一次分解开始,每一级蝶形组的数量都是其前一级的 4 倍,一个蝶形组的蝶形数量都是其前一级蝶形组内蝶形数量的 1/4.

如果总共可以分解 $L_{\max}-1$ 次,则在第 L 次分解后有

1)蝶形组的数量 $N_{\text{bfg}}=\text{pow}(4, L)$;

2)一个蝶形组中的复数数据个数 $N_L=\text{pow}(4, L_{\max}-L)$.

本文用 A, B, C, D 分别代表一次蝶形运算依次的 4 个源数据或 4 个结果数据,A, B, C, D 也称为数据的蝶形属性(简称蝶形属性).

2　基于向量 SIMD 混洗的处理模型

本文研究的算法主要基于如下处理器硬件资源特征:

1)具有向量 SIMD 处理单元.该单元支持 SIMD 运算,包括向量寄存器和向量功能单元.如果向量长度为 p,则它可以对由 p 个数据组成的向量进行同步处理.

2)具有片内高速向量数据缓存 VecCache.该单元支持向量化的数据访问,向量 SIMD 处理程序段的源数据和结果数据都在 VecCache 中.此外,VecCache 支持通过 DMA 操作与 DDR 存储器来交换数据.

3)具有向量混洗处理功能.该功能支持在向量变量内进行字粒度的数据重排,这种重排操作是指执行重排操作前所设置的具体混洗模式.

4)具有带间隔 DMA 的高级数据传送功能.该功能支持在 DMA 数据传送中以某种固定的间隔挑选一定数量的源数据,并将它们以另一种固定的间隔存放在目标存储空间中.

在这种向量 SIMD 体系结构下,一次向量化的蝶形运算可以同时对 p 个蝶形进行计算,本文接下来的部分也用前述 A, B, C, D 分别代表连续的 p 个蝶形中相应的 p 个数据.

2.1　源数据的预处理及其片内向量数据缓存中的数据存储模型

令 Re 和 Im 分别代表一个复数的实部和虚部.在一般情况下,DDR 中存储的源复数数组的数据存放格式为 Re_1, Im_1, Re_2, Im_2, …, Re_N, Im_N.本文假定处理器没有专门的针对整型复数的运算指令,因此,为了进行 FFT 过程中的向量化计算处理,需要将源复数数据数组在 VecCache 中修改为如下存放格式:

1)实部:Re_0, Re_1, …, Re_{N-1};

2)虚部:Im_0, Im_1, …, Im_{N-1}.

利用处理器的带间隔 DMA 高级数据传送功能,用 2 次 DMA 传输即可实现上述效果.每次 DMA 传输时,应将源空间的数据间隔设置为 4 字节,目的空间的数据间隔设置为 0.

令变换的输入复数数据个数为 $N_L=4^{L_{\max}}$,其中 $L_{\max}$可以为 2, 3, 4, 5, …,并假设 VecCache 能容纳全部的 N_L 个数据.这样,在计算的初始时刻,VecCache 中的数据的存储格式及数据与蝶形的对应关系如表 1 所示.

表 1　VecCache 中初始的数据存储格式

存储地址	蝶形属性	实部内容组织	虚部内容组织
高地址 ↑ 低地址	D	$\text{Re}_{3N/4}$, …, Re_{N-1}	$\text{Im}_{3N/4}$, …, Im_{N-1}
	C	$\text{Re}_{N/2}$, …, $\text{Re}_{3N/4-1}$	$\text{Im}_{N/2}$, …, $\text{Im}_{3N/4-1}$
	B	$\text{Re}_{N/4}$, …, $\text{Re}_{N/2-1}$	$\text{Im}_{N/4}$, …, $\text{Im}_{N/2-1}$
	A	Re_0, …, $\text{Re}_{N/4-1}$	Im_0, …, $\text{Im}_{N/4-1}$

然而,处理器的 VecCache 的尺寸总是有限的.当要进行 FFT 运行的数据量很大时,VecCache 可能只能存放 N_L 个数据中的一部分.在本文研究中,令 VecCache 能够存放的最多复数个数为 m.考虑向量 SIMD 处理单元每次处理的向量长度为 p,进一步假定 m 为 $4p$ 的整数倍.

考虑 $m \leqslant N_L$ 的完整情况,必然需要各部分数据轮转使用 VecCache.在第一级运算过程中,第 h 次轮转时($h=0, 1, 2, \cdots, \text{ceil}(N/m)$),VecCache 中的数据存储格式如表 2 所示.

表 2 第 h 次轮转时 VecCache 中的数据存储格式

存储地址	蝶形属性	实部内容组织	虚部内容组织
高地址 ↑ 低地址	D	$\mathrm{Re}_{3N/4+h\times m+3m/4}$, …, $\mathrm{Re}_{3N/4+h\times m+m-1}$	$\mathrm{Im}_{3N/4+h\times m+3m/4}$, …, $\mathrm{Im}_{3N/4+h\times m+m-1}$
	C	$\mathrm{Re}_{N/2+h\times m+m/2}$, …, $\mathrm{Re}_{N/2+h\times m+3m/4-1}$	$\mathrm{Im}_{N/2+h\times m+m/2}$, …, $\mathrm{Im}_{N/2+h\times m+3m/4-1}$
	B	$\mathrm{Re}_{N/4+h\times m+m/4}$, …, $\mathrm{Re}_{N/4+h\times m+m/2-1}$	$\mathrm{Im}_{N/4+h\times m+m/4}$, …, $\mathrm{Im}_{N/4+h\times m+m/2-1}$
	A	$\mathrm{Re}_{h\times m}$, …, $\mathrm{Re}_{h\times m+m/4-1}$	$\mathrm{Im}_{h\times m}$, …, $\mathrm{Im}_{h\times m+m/4-1}$

2.2 分解的各级蝶形运算的处理方法

假定待变换数据数组经过 L_k 次分解后,VecCache 可能容纳一个此时的子 DFT 的数据.这时,可以将 VecCache 中的数据持续计算到最后一级蝶形运算,从而避免冗余的 DMA 传输操作.

在向量 SIMD 处理单元的向量长度为 p 的条件下,假定变换数据数组经过 L_x 次分解后(此时进行到 $L_{\max}-\log_4(4p)+1$ 级蝶形运算),子 DFT 的复数数据数组长度正好为 $4p$.在第 L_x 级蝶形计算之前,每一级蝶形组所包含的蝶形数量大于向量长度 p,所以每一级以向量方式进行长度为 p 的蝶形计算后,必须将结果数据存回到 VecCache 中,直到所有的蝶形计算完成之后才能进入下一级的蝶形计算.在进行第 L_x 级蝶形计算时,该级内 1 个蝶形组所包含的蝶形数量正好为 p.此时,像前面各级一样进行处理,则在后续从第 L_x+1级开始各级的向量化计算中,必然存在蝶形组内的蝶形数量与向量长度间的失配问题.

为了解决这种失配问题,一个基本的方法是利用处理器的高级 DMA 传输功能(如 M7002 的索引传输功能),将数据按某一级(如 L_c 级)变换计算所需数据排布的规律重新排序,但是这种方法存在 2 个不足:一是需要用 2 次 DMA 传输才能将数据在 VecCache 中排列好;二是仍然需要重新将数据加载到向量寄存器中.

当处理器提供向量混洗功能时,因为 $4p$ 个数据正好可以由 4 个向量存放,所以,这 4 个向量的内容可以作为一个整体进行向量混洗操作,得到合适的顺序来进行下一级蝶形运算,从而将最后的 $\log_4(4p)$ 级蝶形运算持续进行到最后一级,减少访存操作.

2.3 总体算法

所采用的基于向量 SIMD 混洗的定点 FFT 的总体算法如下:首先,计算蝶形运算总级数、VecCache 中的最大蝶形数、2 个重要的运算级序号、初始蝶形组数量和初始蝶形组内的蝶形数量;然后,分前 L_k 级的蝶形运算、中间 L_x-L_k 级的蝶形运算和最后的 $\log_4(4p)$ 级的蝶形运算这 3 个阶段进行计算.基于向量 SIMD 混洗的定点 FFT 的总体算法如算法 1 所示,它将输入数据数组 cxs 地址与输入数据等尺寸的临时数组 cxsT 地址作为输入,其结果覆盖 cxs.将算法 1 得到的结果再通过串行的方法进行基 4 码位反序操作,即可得到 FFT 的最终结果.

在算法 1 中,涉及的几个子过程的作用如下:

1) PreprocessInput 子过程用于将数据进行实部和虚部分开存放处理.当 FFT 计算过程中 VecCache 能容纳全部输入数据时,其处理结果数据保持在 VecCache 中,否则,数据将被传回到 cxsT 中.

2) ProcessBFGPartByPart 子过程用于当计算过程中 VecCache 不能容纳全部输入数据时,分部分对数据进行第 L_c 级的蝶形计算.

3) ProcessBFG 子过程用于对 VecCache 中的 1 个第 L_c 级的蝶形组进行蝶形计算.当 VecCache 中没有数据,或者 VecCache 中的数据计算完毕且还有数据没有计算时,它要将剩余待计算的第 L_c 级蝶形组数据调入 VecCache.

4) ProcessBFGGetVec 子过程用于将第 L_x 级的 1 个蝶形组 bfg 的 A, B, C, D 各部分的内容分别构成向量 VA, VB, VC, VD.此时,1 个蝶形组 bfg 中有 $4p$ 个数据正好可以由 4 个向量存放.

5) ProcessVecCalc 子过程用于将 VA, VB, VC, VD 进行第 L_x 级的向量 SIMD 基 4 蝶形计算.

6) ProcessByShuffle 子过程用于进行第 L_x+1 级~第 $L_{\max}-1$ 级的蝶形计算.由于计算过程涉及基于硬件向量混洗功能的蝶形计算,下一小节将以向量长度为 16 为例,对其方法进行详细描述.

算法 1 的流程如下:

input: cxs, cxsT, N, m, p, dmaEleNum, bFitSize
output: FFT 的结果
begin:

```
1:    L_max := log4(N)
2:    N_mbf := m/4
3:    L_k := L_max - log4(N_mbf)
4:    L_x := L_max - log4(4p)
5:    N_bfg := 1
6:    N_bf := N/4
7:    PrepareShuffleModes( )
8:    PreprocessInput(cxs, cxsT, cxNum, dmaEleNum, bFitSize)
9:    for L_c = 0 and L_c < L_k
10:       for i_bfg from 0 to N_bfg - 1
11:           ProcessBFGPartByPart(cxsT, cxNum, i_bfg, N_mbf, N_bf, L_c)
12:   end for
13:       N_bfg := N_bfg * 4
14:       N_bf := N_bf/4
15:       L_c := L_c + 1
16:   end for
17:       for L_c = L_k and L_c < L_x
18:       for i_bfg from 0 to N_bfg - 1
19:           ProcessBFG(cxsT, cxNum, i_bfg, N_bf, L_c)
20:   end for
21:       N_bfg := N_bfg * 4
22:       N_bf := N_bf/4
23:       L_c := L_c + 1
24:   end for
25:   for i_bfg from 0 to N_bfg - 1
26:       ProcessBFGGetVec(VA, VB, VC, VD)
27:       ProcessVecCalc(VA, VB, VC, VD, L_x)
28:       ProcessByShuffle(VA, VB, VC, VD, L_x+1, L_max)
29:   end for
end
```

2.4　基于向量混洗技术的蝶形组计算方法

以向量长度为 16 为例,介绍 ProcessByShuffle 的算法.

1)第 L_x+1 级蝶形运算的处理方式

蝶形计算的第 L_x+1 级即第 L_{max}−2 级,1 个蝶形组中包含 4 个蝶形.当第 L_x 级的 1 个蝶形组的 4 个向量 VA, VB, VC, VD 经过该级蝶形计算后,它们所存储的数据在第 L_x+1 级中分别属于连续的 4 个蝶形组,具体的对应关系如表 3 所示,表 3 中的字母和序号分别表示第 L_x+1 级的蝶形组数据部分及蝶形序号(以 4 为周期).

表 3　第 L_x 级的结果向量与第 L_x+1 级的蝶形组内容的对应关系(1 格表示连续的 4 个数据)

向量名称	蝶形组第 1 部分	蝶形组第 2 部分	蝶形组第 3 部分	蝶形组第 4 部分
VA	A_1	B_1	C_1	D_1
VB	A_2	B_2	C_2	D_2
VC	A_3	B_3	C_3	D_3
VD	A_4	B_4	C_4	D_4

由于需要继续以向量方式进行第 L_x+1 级的蝶形计算,直接通过处理器的向量混洗功能将表 3 中的 4 个向量的内容重新排列成如表 4 所示的顺序(表 4 中的 i 为 4 的整数倍).

表 4 经向量混洗后第 L_x+1 级计算前向量中的数据排列规律

向量名称	第 i 个蝶形组	第 i+1 个蝶形组	第 i+2 个蝶形组	第 i+3 个蝶形组
VA	A_1	A_2	A_3	A_4
VB	B_1	B_2	B_3	B_4
VC	C_1	C_2	C_3	C_4
VD	D_1	D_2	D_3	D_4

得到表 4 中的向量后,用相同的算法进行第 L_x+1 级的蝶形计算,得到的结果仍然保存在 VA, VB, VC, VD 这 4 个向量中.

2)第 L_x+2 级蝶形运算的处理方式

在第 L_x+1 级中,1 个蝶形组仅包含 1 个蝶形.在完成第 L_x+1 级的蝶形计算后,VA, VB, VC, VD 这 4 个向量所存储的数据在第 L_x+2 级(即第 $L_{\max}$−1 级)中分别属于连续的 16 个蝶形组,具体的对应关系如表 5 所示,表 5 中的字母和序号分别表示第 L_x+2 级蝶形组内的数据部分和蝶形序号(以 16 为周期).

表 5 第 L_x+1 级的结果向量与第 L_x+2 级的蝶形组内容的对应关系

向量名称	第 1 部分	第 2 部分	第 3 部分	第 4 部分
VA	A_1, B_1, C_1, D_1	A_2, B_2, C_2, D_2	A_3, B_3, C_3, D_3	A_4, B_4, C_4, D_4
VB	A_5, B_5, C_5, D_5	A_6, B_6, C_6, D_6	A_7, B_7, C_7, D_7	A_8, B_8, C_8, D_8
VC	A_9, B_9, C_9, D_9	A_{10}, B_{10}, C_{10}, D_{10}	A_{11}, B_{11}, C_{11}, D_{11}	A_{12}, B_{12}, C_{12}, D_{12}
VD	A_{13}, B_{13}, C_{13}, D_{13}	A_{14}, B_{14}, C_{14}, D_{14}	A_{15}, B_{15}, C_{15}, D_{15}	A_{16}, B_{16}, C_{16}, D_{16}

同样考虑向量方式蝶形计算的需求,直接通过处理器的向量混洗功能将表 5 中的 4 个向量的内容重新排列成如表 6 所示的顺序.

表 6 向量混洗后第 L_x+2 级计算前向量中的数据排列规律

向量名称	数据 1	数据 2	……	数据 16
VA	A_1	A_2	…	A_{16}
VB	B_1	B_2	…	B_{16}
VC	C_1	C_2	…	C_{16}
VD	D_1	D_2	…	D_{16}

通过对表 6 中的向量进行第 L_x+2 级的蝶形计算后获得 64 个结果数据,保存在 VA, VB, VC, VD 中.然后用这 4 个向量覆盖它们在 VecCache 中对应的第 L_x 级原数据,完成对这部分数据的 FFT 计算.

2.5 定点运算精度补偿

对于本文研究的 32 位定点 FFT,由于定点计算时,小数部分使用四舍五入会产生误差,因此,在 FFT 计算的过程中,还需要对蝶形计算的结果值进行补偿.采用公开的源码中的定点 FFT 算法的定点补偿方法来进行定点运算精度补偿[17].具体方法如下:将原有的旋转因子 W(W 的值为−1.0~1.0)放大(乘以整型变量的最大可能取值 INT_MAX)成为一个整型变量.当进行定点 FFT 计算时,先按照常规的蝶形计算流程处理,获得 1 个 64 位的 long long 类型的计算结果,将计算结果加上一个补偿值后将其右移 31 位还原成 4 字节整型值,该整型值即为蝶形计算的最终结果.上述操作可以使用式(7)来进行计算.

$$x_2 = (x_1 + v) >> 31. \tag{7}$$

式中:x_2 为进行定点补偿后的最终结果值;x_1 为蝶形计算得到的结果值;v 为一个补偿值;>>为逻辑右移.

根据文献[17],v 的数值为 1<<30.通过这种补偿机制,可以去除定点 FFT 计算中四舍五入带来的误差值,保证数据的结果与浮点计算结果的整数部分相同.

3 模型实现

对于所提出的计算模型,其数据预处理、向量 SIMD 计算和向量混洗等环节取决于处理器体系结构的

相关具体技术特征.本节介绍在 YHFT-Matrix2 架构的芯片搭建的 FT-M7002DSK 开发平台上对该模型的具体实现方法,该平台的处理器内核型号为 M7002,其 SIMD 处理向量长度 $p=16$.

3.1　源数据处理

FT-M7002DSK 开发平台处理器的内核型号为 M7002.借助 M7002 内核的带索引 DMA 传输功能,可以将实部和虚部相间存放的输入数组快速转换成 2 个分别连续存放实部和虚部的数组.基于索引 DMA 的输入数据预处理算法的伪代码如算法 2 所示.

算法 2 的流程如下:

```
input: cxs, cxsT, N, dmaEleNum, bFitSize
output: 保存在 VecCache 或 cxsT 中的实部数组和虚部数组
begin:
1:      srcAddr := cxs
2:      dstAddr := head of VecCache
3:      num := 0
4:      while num < N do
5:          if num+ daEleNum > N
6:              dmaEleNum:= N−num
7:          DoDmaTransmit(srcAddr, 1, dmaEleNum, dstAddr, 0)
8:          DoDmaTransmit(srcAddr+1, 1, dmaEleNum, dstAddr+ dmaEleNum, 0)
9:          if bFitSize is not TRUE
10:             DoDmaTransmit(dstAddr, 0, dmaEleNum, cxsT, 0)
11:         DoDmaTransmit(dstAddr+ dmaEleNum, 0, dmaEleNum, cxsT+N, 0)
12:         num := num + dmaEleNum
13:     end while
end
```

在算法 2 中,dmaEleNum 为 1 次 DMA 传输对应的复数个数的最大值,bFitSize 表示 VecCache 是否能够在 FFT 计算过程中容纳下全部的输入数据.子过程 DoDmaTransmit 的作用是进行带索引的 DMA 传输,其第 2 个和最后 1 个参数分别是源索引和目标索引,分别表示在源和目标空间中被传送的数据两两之间相隔的数据个数.在第 7 和第 8 行中,它被用来将实部和虚部相间存放的输入数据转换成实部和虚部分开存放的形式(全部实部数据在前,全部虚部数据在后).在第 9 和第 10 行中,它被用来将实部数组和虚部数组传送到 cxsT 这个临时空间中.

3.2　基本的向量 SIMD 蝶形计算方法

在算法 1 中,ProcessBFGPartByPart 和 ProcessBFG 都包含对 VecCache 中的数据进行向量 SIMD 蝶形计算的过程.由于 VecCache 的尺寸一般是有限的,设它从首地址 datas 开始总共能够容纳的复数数据数量为 N_c,这些复数对应的蝶形组数量为 M_{bfg},蝶形组内的蝶形数为 M_{bf}.根据频率抽取方式的 FFT 计算的特点,并结合 M7002 的向量 SIMD 运算处理硬件特征,提出的数据存储模型如表 7 所示,将一次完整的向量 SIMD 运算涉及的相应数据存储在 VecCache 中,表 7 中的省略号表示后面的 $M_{\mathrm{bfg}}-1$ 个蝶形组的内容.

表 7　向量混洗蝶形计算前 VecCache 中的数据存储模型

存储地址(低地址→高地址)									
第 1 个蝶形组的实部				……	第 1 个蝶形组的虚部				……
第 1 部分的 M_{bf} 个数据	第 2 部分的 M_{bf} 个数据	第 3 部分的 M_{bf} 个数据	第 4 部分的 M_{bf} 个数据	……	第 1 部分的 M_{bf} 个数据	第 2 部分的 M_{bf} 个数据	第 3 部分的 M_{bf} 个数据	第 4 部分的 M_{bf} 个数据	……

需要注意的是,当 VecCache 的数据只是 1 个蝶形组全体数据的一部分时,M_{bfg} 也应为 1,此时 M_{bf} 为 N_c 对应的蝶形数,即 $M_{\mathrm{bf}}=N_c/4$.这样,对应的向量 SIMD 蝶形计算方法如算法 3 所示,它适用于对 FFT 的前 L_x-1 级的蝶形计算.

在算法 3 中,外层的 for 迭代过程用于处理 VecCache 中的各个蝶形组,内层的 while 迭代过程用于对 1 个蝶形组或 VecCache 中的全部蝶形组的数据进行蝶形计算.该算法执行完成后,VecCache 中的数据即成为下一级蝶形计算的输入数据.

算法 3 的流程如下:

input: datas, N_c, M_{bfg}, M_{bf}

output: 蝶形计算的结果

begin:

```
1:      for i_Nbfg := 0 to M_bfg-1 do
2:          reAddr := datas + i_Nbfg * M_bf * 4
3:          imAddr := reAddr+Nc
4:          i_Nbf1 := 0
5:          i_Nbf2 := M_bf
6:          i_Nbf3 := 2 * M_bf
7:          i_Nbf4 := 3 * M_bf
8:          while i_Nbf1+p no larger than M_bf, do
9:          LdGetsrcV(reAddr, i_Nbf1,i_Nbf2,i_Nbf3,i_Nbf4,VAre,VBre,VCre,VDre, p)
10:         LdGetsrcV(imAddr, i_Nbf1, i_Nbf2, i_Nbf3, i_Nbf4, VAim , VBim , VCim , VDim, p)
11:         ButterflyCalc( )
12:         StoreResV(reAddr,i_Nbf1,i_Nbf2,i_Nbf3,i_Nbf4,VAre,VBre,VCre,VDre, p)
13:         StoreResV(imAddr, i_Nbf1, i_Nbf2, i_Nbf3 , i_Nbf4,VAim, VBim , VCim , VDim, p)
14:         IndexAdd(i_Nbf1, i_Nbf2, i_Nbf3, i_Nbf4, p)
15:     end while
16: end for
```

end

在算法 3 中,涉及的几个子过程的作用如下:

1)LdGetsrcV 子过程用于将指定地址的数据按照不同的偏移量分别读取到不同的向量寄存器中.其中,第 9 行用于从 reAddr 偏移 i_Nbf1, i_Nbf2, i_Nbf3, i_Nbf4 的位置分别读入 p 个数据到实部向量 VAre, VBre, VCre, VDre;第 10 行用于从 imAddr 偏移 i_Nbf1, i_Nbf2, i_Nbf3, i_Nbf4 的位置分别读入 p 个数据到虚部向量 VAim, VBim, VCim, VDim.

2)ButterflyCalc 子过程用于利用式(3)~式(6)对上述实部和虚部向量进行蝶形计算,结果保存在这些向量中.在此过程中,需要在实部和虚部结果向量的基础上进行定点运算精度补偿.定点运算精度补偿的方法如算法 4 所示(以第 2 个蝶形组的实部计算为例),其中,llvalue 是一个 64 位的整数,vectorGetLow32(src)和 vectorGetHigh32(longint)分别用于得到 64 位的 src 的低 32 位数据和高 32 位数据.

3)StoreResV 子过程用于将上述实部和虚部结果向量写回到 VecCache 中的原位置.其中,第 12 行用于写回实部的数据,第 13 行用于写回虚部的数据.

4)IndexAdd 子过程用于将 i_Nbf1, i_Nbf2, i_Nbf3, i_Nbf4 增加 p.

算法 4 的流程如下:

input: vU12_re, vU12_im, vw12_re, vw12_im

output: 蝶形计算经过定点精度补偿的结果 vd12_reh

begin:

```
1:  coefs2 := 1<<30;
2:  vectorMul (vU12_im, vw12_im, llvalue);
3:  temp1 := vectorGetLow32(llvalue);
4:  temp2 := vectorGetHigh32(llvalue);
5:  vectorMul (vU12_re, vw12_re, llvalue);
6:  temp3 := GetLow32(llvalue);
```

```
7:      temp4 := GetHigh32(llvalue);
8:      vectorSub (temp4, temp2, vd2_reh);
9:      vectorSubUnsigned (temp3, temp1, vd2_rel);
10:     vectorLessThanUnsigned (temp3, vd2_rel, flag1);
11:     vectorSub (vd2_reh, flag1, vd2_reh);
12:     vectorAddUnsigned (vd2_rel, coefs2, temp1);
13:     vectorLessThanUnsigned (temp1, vd2_rel, flag1);
14:     vectorAdd (flag1, vd2_reh, vd2_reh);
15:     vectorlogicleft(1, vd2_reh, vd2_reh);
16:     vectorlogicright(31, temp1, flag1);
17:     vectorAdd (vd2_reh, flag1, vd12_reh);
end
```

3.3　最后若干级的基本向量混洗的蝶形计算

根据算法 1,在准备进行第 L_x 级蝶形计算时,数据在 VecCache 中仍然按表 7 所示的模型来存储,且该级的一个蝶形组的计算结果仍然保存在 VA, VB, VC, VD 这 4 个向量中(注意分实部和虚部 2 组).由于此后的 $L_{max}-L_x$ 级的蝶形计算中,蝶形组中的蝶形数量 N_{bf} 不超过向量长度 p,所以可以不必再将数据写回存储器,避免计算过程中的数据重排处理和访存操作,从而进一步节约 FFT 的时间.

不过,对于 VA, VB, VC, VD 中第 L_x 级的结果数据,不能直接将它们作为第 L_x+1 级蝶形计算的输入,必须先将它们涉及的 $4p$ 个复数按第 L_x+1 级向量 SIMD 计算的要求重新进行排列.在 M7002 中,这个数据重排处理可以通过硬件的向量混洗操作实现.M7002 的向量混洗部件能够根据事先设置好的重排模式,将保存在向量中的内容按指定的顺序进行排列.此外,M7002 提供了较大的片内混洗模式存储区,程序实现时,可以在计算过程开始前就设定好多个不同的混洗模式,在计算过程中提供混洗模式的编号、源向量和目的向量即可完成指令级速度的向量混洗操作.

考虑到 M7002 的向量长度为 16,在满足基 4 条件的数据个数的情况下,第 L_x 级的 4 个结果向量包含第 L_x+1 级中连续 4 个蝶形组的数据,每个结果向量对应第 L_x+1 级的 1 个蝶形组.此外,第 L_x+1 级的4 个结果向量包含第 L_x+2 级中连续 16 个蝶形组的数据,每个结果向量对应第 L_x+2 级的 4 个蝶形组.

为了在第 L_x+1 级用同样的向量 SIMD 方法进行蝶形运算,必须把第 L_x 级的每个向量的内容平均分成 4 个部分,通过向量混洗存放到 4 个不同向量相同位置的片断中,其混洗原理如图 1 所示,图 1 中的箭头为向量混洗前后同一个数据片断在 4 个向量中的位置.由于 M7002 的混洗操作允许 2 个源向量和 1 个目标向量,图 1 展示了两步混洗法:第一步从第 L_x 级的 2 个结果向量进行向量混洗得到 1 个中间向量,4 次混洗就得到 4 个不同的中间向量;第二步从这 4 个中间向量中,选择 2 个向量进行向量混洗得到 1 个第 L_x+1 级的输入向量,4 次混洗就得到 4 个所需的输入向量.

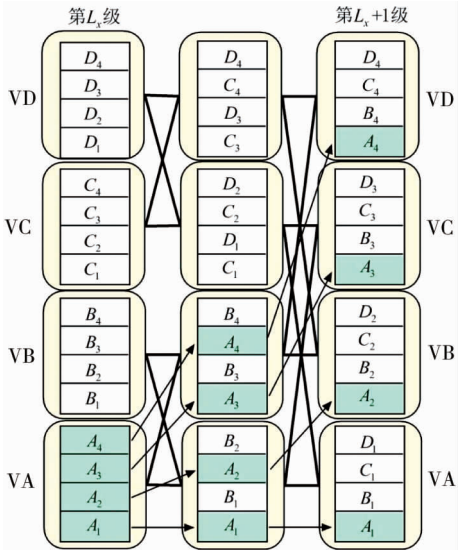

图 1　将第 L_x 级结果向量通过硬件混洗转换成第 L_x+1 级的输入向量的原理(每一格表示连续的 4 个数据)

在算法 1 的 ProcessByShuffle 子过程中的第一阶段,根据图 1 的原理,对第 L_x 级的实部和虚部向量组分别进行硬件混洗操作,即可得到第 L_x+1 级的输入实部向量组和输入虚部向量组.在此基础上,利用式(3)~式(6)对上述实部和虚部向量组进行蝶形计算,即可得到第 L_x+1 级的结果向量组.

对于第 L_x+2 级,同样需要按照上述过程进行硬件混洗操作和蝶形计算.相应的处理在算法 1 的 ProcessByShuffle 子过程中的第二阶段进行,将第 L_x+1 级的结果向量通过硬件混洗转换成第 L_x+2 级的输入向量的原理如图 2 所示,这里同样用到了上述的两步混洗法得到第 L_x+2 级的输入向量.

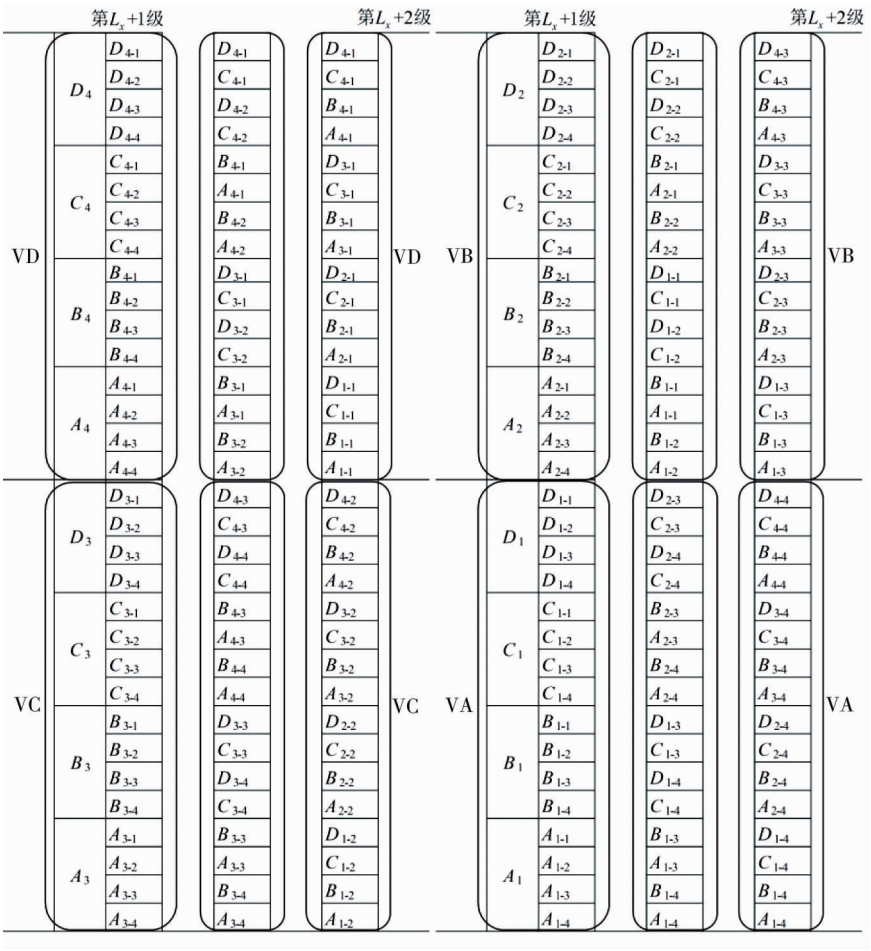

图 2 将第 L_x+1 级结果向量通过硬件混洗转换成第 L_x+2 级的输入向量的原理(每一小格表示 1 个数据)

在具体编程时,根据图 1 和图 2 所示的混洗原理,依照处理器对混洗模式的配置数据规定,即可写出相应的配置数据,在使用本文算法前将这些配置数据写入 M7002 中对应的片内混洗模式存储区即可.

4 试验和讨论

在 YHFT 芯片搭建的 FT-M7002DSK 板卡上对算法性能进行测试.样本点数范围为 64~65 536 个,在各种满足基 4 条件数据量的情况下,测试该算法运行所需的节拍数.此外,相应地在 TI CCS 系统中使用模拟器获得 TMS320C6678 对应库函数运行所需的节拍数,并将两者进行对比(只考虑计算所需时间,不包含数据传输时间),试验结果如图 3 所示,加速比的计算方法为 TI 模拟器获得的节拍数除以本文算法运行所需的节拍数.

由图 3 可知:随着数据量的增多,加速比逐渐增大,性能提升明显.当数据量达到 65 536 时,加速比达到 11.09 倍.此外,只要数据量大于 256,本文算法的性能均要优于 CCS 中相应库函数的性能,平均加速比可以达到 4.23 倍.

由于向量 DSP 体系结构寄存器的数量较多,可以对 FFT 计算使用循环展开技术进行处理,以充分利用寄存器资源,进一步提升算法的性能.考虑 FT-M7002 向量寄存器的数量情况,通过对前 L_x-1 级蝶形计算过程的最内层循环寄存器和执行单元使用情况的分析,发现可以对该内层循环进行循环展开操作,且当循环展开因子为 2 时不会出现寄存器数量不足的情况.因此,对前 L_x-1 级蝶形计算过程的最内层循环进行展开因子为 2 的循环展开操作.图 4 为对本文算法中的 2 个循环进行循环展开操作后与 TI 相应库函数的算法性能对比.由图 4 可知:当数据量大于 256 时,各种数据量下的性能均有所提升,且随着数据量的增大,性能提升越多,加速比最高达到 12.56 倍,平均加速比达到 4.79 倍.与图 3 相比,图 4 的平均性能得到进一步的提升.

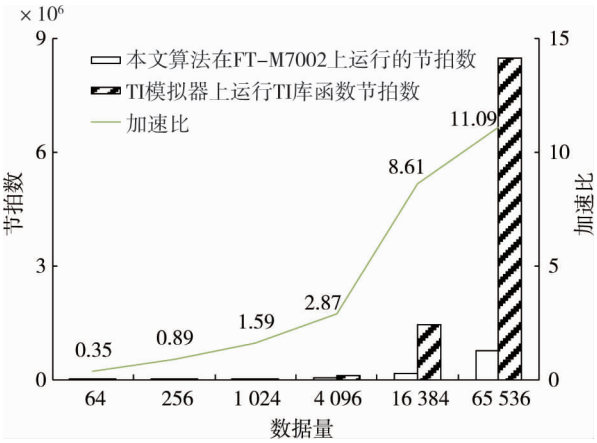

图 3　本文算法与 TI 库函数的性能对比

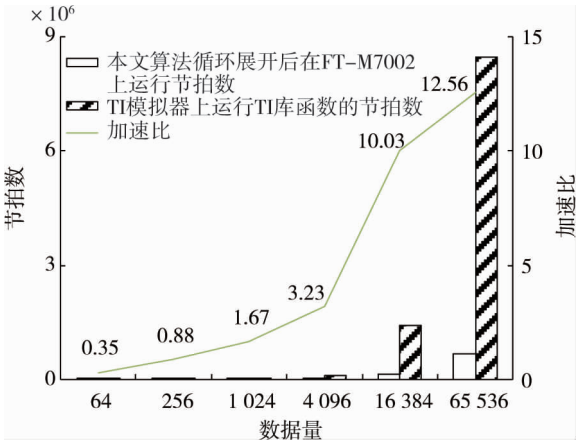

图 4　本文算法使用循环展开后与 TI 库函数的性能对比

除此之外,本文通过对不同类型的相同输入数据进行定点 FFT 和浮点 FFT 运算,并对比两者的运行结果对本文算法的精度进行分析.该对比试验的样本点数范围为 16~16 384 个,在各种满足基 4 条件的数据量的情况下,对各数据量的输入数据进行定点 FFT 和浮点 FFT 运算后的输出结果进行比较,最后求得该数据量下 2 种输出结果的平均误差值,结果如图 5 所示.图 5 的纵坐标即为横坐标对应数据量的输入数据运行定点 FFT 和浮点 FFT 函数后输出结果的平均误差值.

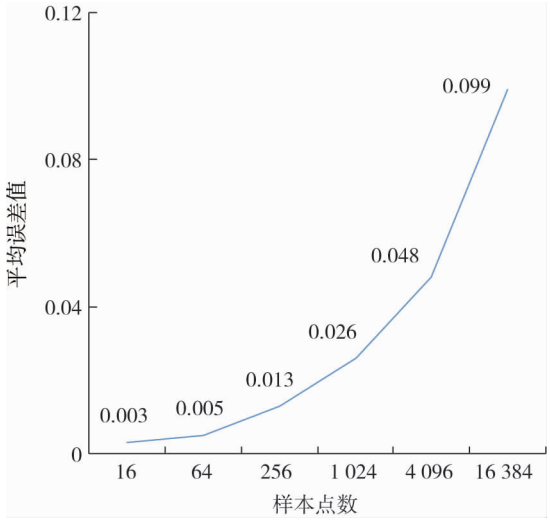

图 5　定点 FFT 与浮点 FFT 中每个样本点数运行结果的平均误差值

由图 5 可知:当样本点数为 16 时,相同输入数据运行定点 FFT 和浮点 FFT 函数后的运行结果的平均误差约为 0.003,样本点数越大,误差越大,当样本点数为 16 384 时,平均误差为 0.099,可以忽略,因此,本文提出的定点 FFT 算法的精度较高.

5　结论

1)所提出的定点 FFT 的 DSP 向量混洗并行基 4 算法通过对内层循环进行循环展开优化,提升了算法指令级的并行性.

2)与 CCS 模拟所得 TI 的 TMS320C6678 库函数性能进行试验对比,本文算法的最高加速比可以达到 TI 库函数的 11.09 倍.

3)对本文算法使用循环展开优化后,平均性能得到进一步提升,与 CCS 模拟所得 TI 的 TMS320C6678 库函数性能相比,本文算法的平均加速比达到 4.79 倍,表明本文算法在发挥向量 DSP 硬件性能方面的效果明显.

参考文献:

[1] COCHRAN W, COOLEY J, FAVIN D, et al. What is the fast Fourier transform? [J]. IEEE Transactions on Audio and Electroacoustics, 1967, 15(2): 45-55.

[2] COOLEY J W, TUKEY J W. An algorithm for the machine calculation of complex Fourier series[J]. Mathematics of Computation, 1965, 19(90): 297-301.

[3] MA Z Q, LU F Y, LIU S Y, et al. An adaptive generalized demodulation method for multimedia spectrum analysis is applied in rolling bearing fault diagnosis[J]. IEEE Access, 2020, 8: 20687-20699.

[4] ADNAN A, LIU Y, CHOW C W, et al. Demonstration of non-hermitian symmetry (NHS) IFFT/FFT size efficient OFDM non-orthogonal multiple access (NOMA) for visible light communication[J]. IEEE Photonics Journal, 2020, 12(3): 7201405.

[5] SAHNOUN K, BENABADJI N. Satellite image compression algorithm based on the FFT[J]. The International Journal of Multimedia & Its Applications, 2014, 6(1): 77-83.

[6] KIM B S, JIN Y, LEE J, et al. High-efficiency super-resolution FMCW radar algorithm based on FFT estimation[J]. Sensors, 2021, 21(12): 4018.

[7] LIU W Q, LIAO Q C, QIAO F, et al. Approximate designs for fast Fourier transform (FFT) with application to speech recognition[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2019, 66(12): 4727-4739.

[8] ZHANG F L, KIM C W, GOI Y. Efficient Bayesian FFT method for damage detection using ambient vibration data with consideration of uncertainty[J]. Structural Control and Health Monitoring, 2021, 28(2): e2659.

[9] 王向前,洪一,王昊,等.魂芯 DSP 的编译器设计与优化[J].电子学报,2015,43(8):1656-1661.

[10] 陈书明,刘胜,万江华,等.协同多核 DSPYHFT-QMBase:体系结构及实现[J].中国科学:信息科学,2015,45(4):560-573.

[11] 潘辑智.DSP 内核—VLIW 与 SIMD 珠联璧合[J].电子产品世界,2004,11(9):45.

[12] 郭利财,刘燕君.龙芯 3A 处理器上 FFT 的高效实现[J].小型微型计算机系统,2012,33(3):594-597.

[13] 张杰,顾乃杰,张明.龙芯 3B 处理器上 FFT 算法向量化研究[J].小型微型计算机系统,2015,36(7):1639-1643.

[14] 杨振浩,郑启龙,邓文齐,等.魂芯 DSP 高效访存并行 FFT 算法研究[J].小型微型计算机系统,2018,39(7):1377-1380.

[15] BAHTAT M, BELKOUCH S, ELLEAUME P, et al. Instruction scheduling heuristic for an efficient FFT in VLIW processors with balanced resource usage[J]. EURASIP Journal on Advances in Signal Processing, 2016, 2016(1): 38.

[16] 罗有才.基于申威平台的混合基 FFT 的实现与优化[D].郑州:郑州大学,2020.

[17] 陈海燕,杨超,刘胜,等.一种高效的面向基 2FFT 算法的 SIMD 并行存储结构[J].电子学报,2016,44(2):241-246.

[18] 龚彤艳,张广婷,贾海鹏,等.一种偶数基 Cooley-Tukey FFT 高性能实现方法[J].计算机科学,2020,47(1):31-39.

[19] 操庐宁.基于 X86-64 计算平台的 FFT 实现与并行优化[D].长春:吉林大学,2019.

[20] 陈暾,李志豪,贾海鹏,等.基于 ARMv8 平台的多维 FFT 实现与优化研究[J].计算机学报,2019,42(11):2384-2402.

[21] 李凤娇,顾乃杰,齐东升,等.基于 ARM SVE 的 FFT 算法向量化研究[J].小型微型计算机系统,2022,43(10):2017-2021.

[22] 刘学梅,孙志坚.按频率抽取的基 4 FFT 算法在 FPGA 中实现[J].现代雷达,2005,27(1):50-51,70.