

基于轻核阵列机的 FFT 算法并行化研究与实现

常立博^{1,2}, 王时雨²

(1 西安邮电大学 电子工程学院, 陕西 西安 710121; 2 西北工业大学 计算机学院, 陕西 西安 710072)

摘 要: 目前普遍采用基于流水的单路径延时反馈结构, 或基于存储结构实现快速傅里叶变换 (Fast Fourier Transform, FFT). 前一种结构效率高但缺乏灵活性, 而后一种结构通用性较好但性能较差. 首先提出了一种 FFT 并行实现的算法, 然后在并行图形阵列机 (Parallel Array Architecture for Graphics, PAAG) 平台上实现了基 2 时间抽取的 FFT (Decimation-In-Time FFT, DIT-FFT) 算法; 最后将长度为 512 的 DIT-FFT 算法分别映射到 1 个 PE、4 个 PE、8 个 PE 和 16 个 PE 上实现. 分析实验结果显示: 随着 PE 个数的增加, 加速比呈曲线上升的趋势; 但随着 PE 个数的不断增加, 算法执行速度增长开始变得缓慢, 当映射到 8 个 PE 上时, 最大加速比可以达到 5.98.

关键词: 快速傅里叶变换算法; 阵列机; 并行

中图分类号: TP332.1

文献标识码: A

文章编号: 1000-7180(2018)01-0100-06

Research and Implementation of Parallel FFT Algorithm Based on Light Core Array

CHANG Li-bo^{1,2}, WANG Shi-yu²

(1 School of Electronic Engineering, Xi'an University of Posts and Telecommunication, Xi'an 710121, China;

2 School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: At present, the fast Fourier transform is realized based on the flow-based single-path delay feedback or storage structure, the former structure is highly efficient but lacks flexibility, and the latter is more versatile but less robust. In this paper, firstly, an algorithm is implemented in parallel with FFT, and then the Decimation-In-Time FFT (DIT-FFT) algorithm is implemented on the Parallel Array Architecture for Graphics (PAAG) platform. Finally, the DIT-FFT algorithm with length of 512 is mapped to 1 PE, 4 PE, 8 PEs and 16 PEs respectively. The results show that with the increase of the number of PEs, the acceleration rate increases with the increase of the number of PEs. However, as the number of PEs increases, the speed of the implementation of the algorithm becomes slow, and when it is mapped to 8 PEs, The acceleration ratio can reach 5.98.

Key words: the fast Fourier transform algorithm; array processor; parallelization

1 引言

快速傅里叶变换^[1]是数字信号处理的核心技术之一, 广泛应用在频谱分析、匹配滤波、数字通信、语音识别、图像处理、雷达处理、遥感遥测、地质勘探和无线保密通讯等众多领域^[2]. 随着信息处理对实时性需求的不断提高, FFT 处理器的硬件加速性能变得越来越重要. 传统的 FFT 硬件加速器通常采用基

于流水的单路径延时反馈结构^[3]或基于存储结构的设计方法^[4]. 前一种结构使用大量并行的蝶形运算单元实现多级蝶形运算的流水并行, 针对特定点数、固定基的 FFT 算法具有很高的数据吞吐率, 但要耗费大量硬件资源, 缺乏灵活性; 后一种结构针对特定一套蝶形运算单元,

基于同址存储机制, 能有效控制面积和功耗, 但难以获得较高的加速性能. 为了满足应用需求的不

断增长,已有学者提出了针对特定领域更高效的FFT算法^[5],但是所设计的电路仅能完成FFT算法,所以通用性较差.文献[6-7]中采用了多个运算单元(processing element, PE)并行结构以提高运算性能,但是处理单元采用的是相对比较复杂的重核实现,所以面积和功耗相对较大.为了解决专用FFT处理器灵活性较差,而采用通用处理单元实现FFT算法时面积和功耗较大的问题,本文采用基于轻核阵列 PAAG^[8-9]结构实现时间抽取的基2DIT-FFT算法的并行实现.本文设计可在不改变硬件电路结构的情况下,通过修改配置信息实现类似算法(如基4FFT算法,图像滤波算法等).

2 算法原理及平台介绍

2.1 基2DIT-FFT算法特点

根据基2-FFT算法原理^[10],DIT-FFT算法的运算特点如下所示.

(1) 原址运算.原址运算是按照时间抽取算法的一个重要的特点.如8点的DIT按时间抽取的FFT算法可分为 $\log_2 N = 3$ ($N = 8$)级运算过程,其中每一级有 $N/2$ 个蝶形运算.对一个蝶形运算单元来说,计算结果只和两个输入数据源有关,且对于同一级中,每个蝶形的输入数据结点在同一条水平线上,因此计算完一个蝶形运算后,所得到的输出数据可以先存入原先的输入数据所在的单元.对于一组 N 点的输入数据,当第一级运算结果数据存到存储器后,以后的每一级运算结果依然存储在以前的同一组存储器中,直到最后一级运算处理完,中间无需再添加其他的存储设备.这种处理方法叫做原址运算.

(2) 旋转因子的变化规律.在 N 点的DIT-FFT运算过程中,每级都有 $N/2$ 个蝶形单元,每个蝶形单元都要乘以旋转因子 W_N^p (p 为旋转因子指数).各级旋转因子和循环方式都有所不同,为了提高运算效率,应先找出旋转因子 W_N^p 与运算级数的关系.用 L 表示从左到右的运算级数($L = 1, 2, \dots, M$),第 L 级共 2^{L-1} 个不同的旋转因子.对于 $N = 2^M$ 的情况,第 L 级的旋转因子如式(1)所示.

$$W_N^p = W_{2^L}^J, \quad J = 0, 1, 2, \dots, 2^{L-1} - 1 \quad (1)$$

因为 $2^L = 2^M \cdot 2^{L-M} = N \cdot 2^{L-M}$,则可得到:

$$W_N^p = W_{N \cdot 2^{L-M}}^J = W_N^{J \cdot 2^{M-L}} \quad (2)$$

(3) 蝶形单元.令 p 为参与本蝶形单元运算的上节点序号, q 为下节点序号,则对于第 m 级运算,则有:

$$x_{m+1}(p) = x_m(p) + W_N^r x_m(q) \quad (3)$$

$$x_{m+1}(q) = x_m(p) - W_N^r x_m(q) \quad (4)$$

由于每一级都包含有 $N/2$ 个蝶形单元,每个蝶形单元需要完成一次复乘和两次复加,因此,完成 $M = \log_2 N$ 级共需要的复数乘法数 M_c 和复数加法数 M_a 分别为

$$M_c = \frac{N}{2} \log_2 N = MN/2 \quad (5)$$

$$M_a = N \log_2 N = MN \quad (6)$$

若让每一级的数据自上而下都按自然顺序排列,则在第 m 级,上下节点 p, q 之间的距离则为 $q - p = 2^m$.

(4) 码位倒置.对 $N = 8$ 的FFT,其输入序列的自然序号是0,1,2,3,4,5,6,7.第一次按奇、偶分开,可以得到两组 $N/2$ 点DIT $x(n)$ 的序号是0,2,4,6;1,3,5,7.对每一组再按奇、偶分开,此时应将每一组仍按自然顺序排,因此抽取后的四组,每组序号是0,4;2,6;1,5;3,7.

2.2 基2DIT-FFT算法并行点分析

由DIT-FFT算法特点以看出,该算法具有高度分治特性,即DIT-FFT算法中存在多个运算并行点.

(1) 初始化中的并行性.根据 N 的点数对PE进行初始化,把 N 个数据平均分配到每个PE上,每个PE的初始化具有独立性,不受其他PE所影响.因此,PE从共享存储中读取初值的过程可以并行操作.

(2) 蝶形运算中的并行性.FFT算法的核心为蝶形运算,蝶形运算本身具有并行性.在蝶形运算的整个过程中,一个节点会在不同的计算阶段(即在不同的级)与不同的节点作两点离散傅里叶变换运算,并且,在从左往右不同级别的蝶形运算中存在着严格的先后顺序关系;但是,在每一级的蝶形运算中,参加两点离散傅里叶变换运算的节点对是相互独立的,所以在同级的蝶形运算中,多对两点DIT-FFT就可以由多个计算单元并行计算.在每一级蝶形运算处设置一个同步点,当同一级内的两点DIT全部结束运算后,再一起进入下一级的蝶形运算,开始下一轮的并行两点离散傅里叶变换计算.

(3) 数据传递中的并行性.对于每次蝶形运算而言,它们需要做的是两点离散傅里叶变换运算.前文已经提到,处于不同级的蝶形运算之间存在着严格的先后顺序关系;但是,在每一级的蝶形运算中,参加两点离散傅里叶变换运算的节点对却是相互独

立的,因此对于在同一级的蝶形运算而言,参加两点离散傅里叶变换运算的每一对节点与其他任一节点对之间不存在相互影响,同级蝶形运算的每一组节点对之间不需要进行数据通信,因此可以同时进行上下级之间的数据交换,通过访问共享存储读取当前所需要进行蝶形运算的点值,而不会受到任何影响.

2.3 平台介绍

为了满足大规模科学与工程计算应用对并行计算的需求,解决不断增长的应用对高速度和低存储访问延时的需求,阵列计算机应运而生. 本文算法实现平台-PAAG 处理器^[11],即是一种采用 mesh 结构^[12]实现的轻核高效并行处理器.,该处理器通过一系列设计方法来实现高效、灵活的并行处理,使得该处理器在具有较强的性能的同时又具有很好的灵活性.

一个 PAAG 阵列机系统由多个处理器单元簇(cluster)构成. 比较常见的处理器簇结构可以是一个二维阵列(2D array),比如常见的 4×4 的阵列,一个基本簇是包含 16 个处理单元(PE)的簇,邻近的 PE 相互连接. 多个基本簇可以构成一个二层簇,多个二层簇可以构成一个三层簇.

一个处理器簇包括 16 个处理单元,一个行控制器(row controller, CRi),一个列控制器(column controller, CWj)和一个簇控制器. 近邻的处理单元互联组成二维阵列,行、列控制器实现 SIMD 运算,簇控制器负责协调整个簇中的处理单元,每个簇还有一个存储数据跟程序的簇存储(cluster memory),如图 1 所示.

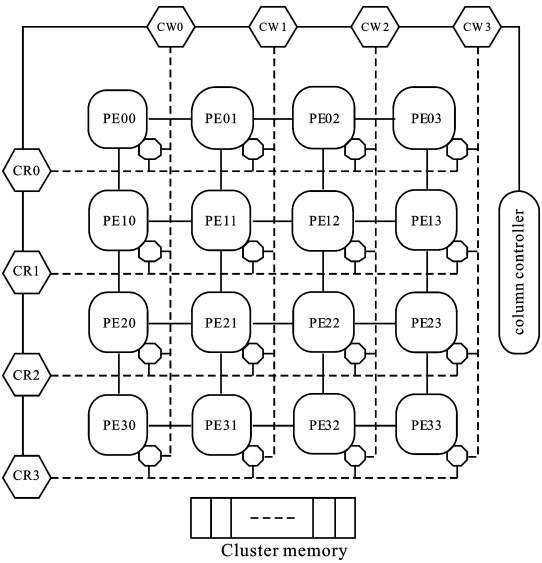


图 1 一个基本簇的结构

每个 PE 单元由多线程控制和微控处理器构成,完成每个线程的数据和指令处理,线程之间没有通信,每个线程都是独立的,微控处理器完成基本处理器的功能. 每行的行控制器和每列的列控制器可以协调实现行列的 SIMD 运算. 整体簇控制器相当于阵列机的控制器,通过行列控制器的辅助可以协调整个阵列机的 PE 单元并发执行. 阵列机有一个簇存储单元,所有处理单元的数据和指令程序段都存储于簇存储单元中,根据行列控制器分配给各个处理单元. 阵列机的控制器同时可以读写阵列机中每个处理单元的线程参数表参数,所以可以提高处理器的利用率、减少功耗,更好地控制阵列机各个处理单元的并发执行.

3 基于阵列机的基 2 DIT-FFT 算法实现

通过对基 2 DIT-FFT 算法的并行点分析可以得出,该算法的实现属于数据级的并行,对于本算法映射到的 PAAG 阵列机上的每一个 PE 而言,它们执行的运算任务是相同的,区别仅仅在于所涉及的运算数据不同. 根据 PAAG 阵列处理器的结构特点和 DIT-FFT 算法自身的特点,可以将实现方法分为两个步骤:

- (1) 据 DIT-FFT 需要计算的 N 值,将各个点平均分配到每个 PE 上来进行计算;
- (2) 根据每一次蝶形变换中各个 PE 需要的数据进行数据交换.

图 2 所示为单个 PE 的执行流程图,其中 PE 初始化、进行蝶形运算、判断当前 PE 的本级蝶形运算次数及数据交换均可以实现并行执行.

3.1 PE 初始化

在计算之前,先根据算法介绍中旋转因子的定义求出各个 W_n 的值,并直接将其存入相应的各个 PE 中,这样可以减少每个 PE 在进行蝶形运算时,从共享存储中读取 W_n 的步骤,节省访问共享存储的时间. 在 PAAG 阵列中,一个簇里包含 16 个 PE,而每 4 个 PE 组成一列,每列有列共享存储,总共有 4 个列共享存储. 4 个列可以并行存取列存储中的数据,列中的每个 PE 对列存储的访问是串行的,如图 3 中 PE00、PE04、PE08、PE12 对 column01 的访问. 当 DIT-FFT 算法进行消息初始化时,不同 PE 同时从共享存储中读取各自需要的数据来实现初始化,达到并行初始化的效果. 每一行的 4 个 PE 的初始化是同时进行的,第一行 PE 初始化完毕后,第二行 PE 开始初始化.

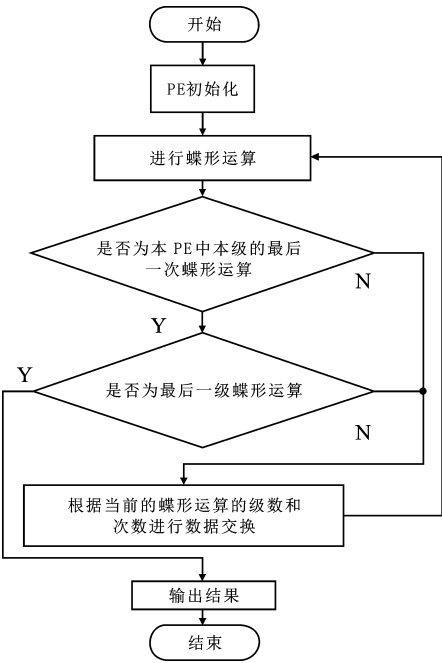


图 2 单个 PE 执行流程图

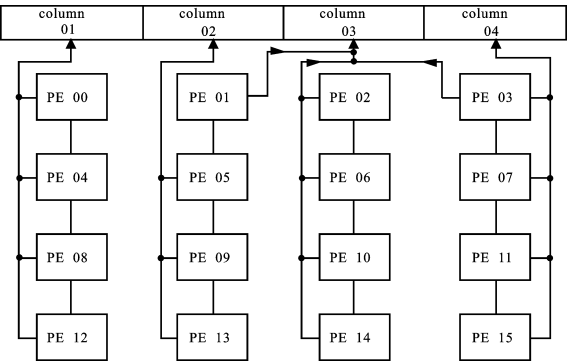


图 3 PE 访问列共享存储

3.2 蝶形运算

如图 4 所示,一次蝶形运算主要包含一次乘法、一次加法和一次减法:

(1)乘法操作根据 FFT 算法,对每个 PE 中的部分初始数据(每个蝶形运算中有一个涉及乘法的初始数据,如图 4 中的初始数据 B)与相应的旋转因子相乘,即图 4 中的 $B \cdot C$;

(2)加法操作将每个蝶形运算的不涉及乘法的初始数据与乘法的计算结果相加并存储,实现数据更新;

(3)减法操作将每个蝶形运算中不涉及乘法的初始数据与乘法的计算结果相减并存储,实现数据更新.

在每一级的蝶形运算中,参加两点离散傅里叶变换运算的节点对相互独立,所以,在同级的蝶形运算中,多对两点 DIT 就可以由多个 PE 同时进行计算.

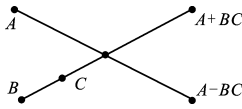


图 4 蝶形运算符号

3.3 步骤判断

判断当前运算执行到的步骤.

(1)判断本级蝶形运算是否已计算完毕.判断当前运算次数是否达到本 PE 中当前所在级蝶形运算的总次数,若未达到,则说明本级蝶形运算还未进行完,继续进行,当运算次数与本级蝶形运算总次数相等时,进行下一步的数据存取交换和下一级蝶形运算.

(2)判断整个 N 点 FFT 算法是否运算结束,即判断各级蝶形运算是否都已计算完毕.若蝶形运算级数未达到 N 点 FFT 所需蝶形运算的总级数,则说明本级蝶形运算还未进行完,当运算次数与本级蝶形运算总次数相等时,进行下一步的数据存取交换和下一级蝶形运算.

3.4 数据交换

数据交换指每个 PE 把当前蝶形运算的结果写入到存储器中,并根据蝶形变换的需要,从存储器中取回其他 PE 写入的结果的过程.通过数据交换,将本 PE 中的数据与下一次蝶形运算所需的其他 PE 中的数据一同存入该 PE 中,进而在 PE 内完成蝶形运算.数据交换可以通过访问共享存储来实现.如图 5 所示,PE a 和 PE b 分别把数据写入共享存储中,再由 PE b 从共享存储中取出 PE a 写入的数据,PE a 取出 PE b 写入的数据,这样就可以达到 PE a 与 PE b 交换数据的效果.

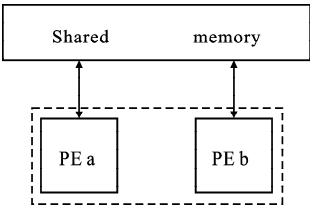


图 5 PE 间通过共享存储实现数据交换

4 实验结果及分析

以 $N=512$ 的基 2 DIT-FFT 算法为例,介绍将该算法分别映射到 1 个 PE、4 个 PE、8 个 PE 种 16 个 PE 上的数据交换实现方法.

(1) PE 个数为 1 时,512 点基 2 DIT-FFT 算法的所有运算都在一个 PE 中完成,不需要进行 PE 间数据交换.

(2) PE 个数为 4 时,基本簇中的每个 PE 需要从共享存储中读取 128 个点的初始数据,在每个 PE 都完成了第一级的 64 次蝶形运算后,将运算得出的 128 个新数据用于进行第二级蝶形运算.以此类推,在每个 PE 内部完成前 7 级蝶形运算.以上过程每个 PE 同时进行.当第 7 级蝶形运算结束后,第 8 级蝶形运算需要与相邻的处理单元 PE 进行数据交换,如图 6 所示,PE00 和 PE01 将运算结果存入共享存储,并从共享存储中读取下一步运算需要的数据.以上过程每两个 PE 之间同时实现,即 PE00 与 PE01 在进行数据交换的同时,PE02 与 PE03 等 PE 对也在进行数据交换,而这些 PE 对在数据交换的过程中彼此互不影响,从而实现数据交换的并行化,后面的几次数据交换也是同理.

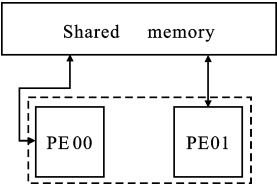


图 6 相邻单个 PE 间数据交换

如图 7 所示,当虚线中的两个处理单元 PE00 和 PE01 联合执行蝶形运算后,即每两个 PE 共同完成了一次 $N=256$ 的 FFT 运算.然后,通过共享存储与 PE04、PE05 进行数据交换,进而完成 512 点 FFT 运算.

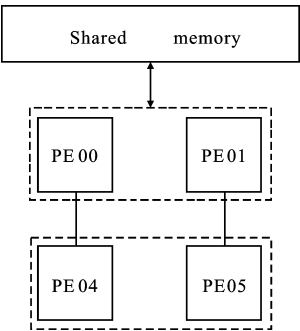


图 7 每两个 PE 作为整体与相邻的两个 PE 间交换数据

(3) PE 个数为 8 时,基本簇中每个 PE 需要从共享存储中读取 64 个点的初始数据,在每个 PE 内部完成前 6 级蝶形运算.从计算第 7 级蝶形运算开始,需要进行 3 次 PE 间数据交换,前两次与 PE 个数为 4 时同理,最后一次数据交换如图 8 所示,虚线括起来的 4 个 PE 进行完蝶形计算后,即每 4 个 PE 完成了一次 $N=128$ 的 FFT 运算.然后,通过共享存储将其数据与其他 PE 进行交换.

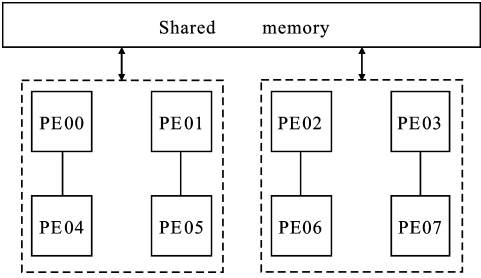


图 8 每四个 PE 作为整体与相邻的四个 PE 间交换数据

(4) PE 个数为 16 时,基本簇中每个 PE 需要从共享存储中读取 32 个点的初始数据,在每个 PE 内部完成前 5 级蝶形运算.从计算第 6 级蝶形运算开始,需要进行 4 次 PE 间数据交换,前两次与 PE 个数为 4 时同理.第三次、第四次 PE 间数据交换如图 8 和图 9 所示.

图 9 中虚线框内的 8 个 PE 计算完毕后,即每 8 个 PE 完成了一次 $N=256$ 的 FFT 运算.然后,通过共享存储将其数据与另外 8 个 PE 进行交换.最后 16 个 PE 共同进行 $N=64$ 的 FFT 蝶形计算,计算的结果分布在 16 个 PE 上.

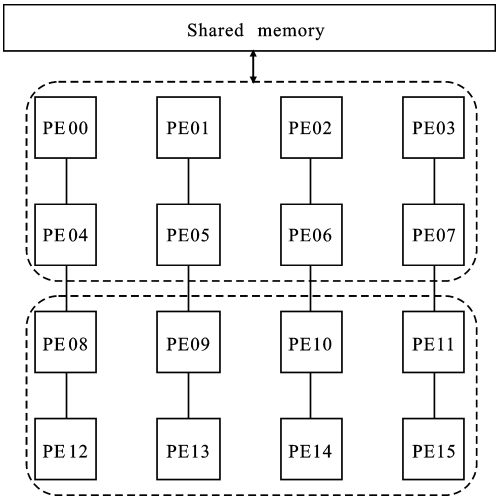


图 9 每八个 PE 作为整体与相邻的八个 PE 间交换数据

下面将 $N=512$ 的基 2 DIT-FFT 算法按照上述步骤映射到 1 个、4 个、8 个和 16 个 PE 上.根据其执行需要的周期数计算出加速比,几种实现机制结果对比如表 1 所示.

表 1 不同 PE 个的加速比

PE 数量	执行周期数	加速比
1	366 340	1
4	214 112	1.71
8	61 060	5.98
16	67 139	5.46

通过表 1 可以看出,将基 2 DIT-FFT 算法并行化映射到 4 个 PE 上时,加速比提升的不是很明显,这是由于 PE 的个数和访问存储、PE 间通信所消耗的时间没有达到一个理想的平衡点. 当映射到 8 个 PE 上时,加速比有了明显的提升,这说明 PE 的个数和访问存储、PE 间通信所消耗的时间基本达到了较理想的平衡点. 但是从 8 个 PE 增加到 16 个 PE 时,执行周期数的减少也开始减缓,因为随着使用 PE 个数的增多,访问存储和 PE 之间通信的时间也在加大,这会影响到最终的执行速度. 因此,就目前实验结果来看,使用 8 个 PE 是较理想的结果,这样可以在加快执行速度的同时尽量降低其他工作所消耗的时间.

5 结束语

本文首先对 FFT 算法的基本原理、基 2 DIT-FFT 算法的运算特点、及其并行点进行了分析和总结;其次提出了一种基于 PAAG 阵列机的基 2DIT-FFT 并行实现算法,最后该算法分别映射到 1 个 PE、4 个 PE、8 个 PE 和 16 个 PE 进行加速实现,并对实验结果进行了分析. 实验结果表明,随着点数的增加会使 PE 与 PE 之间、PE 与共享存储之间的访问也会增加,从而导致加速比下降,所以提高此类算法访存效率将是今后工作的重点.

参考文献:

[1] 丁玉美,高西全. 数字信号处理[M]. 2 版. 西安:西安电子科技大学出版社,2000.

[2] 兰海洋,林晓焕. 快速傅里叶变换的硬件实现[J]. 通信技术,2012, 45(8): 115-117.

[3] Chang Y N, Parhi K K. An efficient pipelined FFT architecture[J]. IEEE Transactions on Circuits & Systems II Analog & Digital Signal Processing, 2003, 50 (6):322-325.

[4] Baas B M. A low-power, high-performance, 1024-Point FFT Processor[J]. IEEE Journal of Solid-State Circuits ,1999, 34(3):380-387.

[5] 方维,孙广中,吴超,等. 一种三维快速傅里叶变换并行算法[J]. 计算机研究与发展, 2011, 48 (3): 440-446.

[6] Shin M, Lee H. A high-speed four-parallel radix-2 4, FFT/IFFT processor for UWB applications [C] // IEEE International Symposium on Circuits and Systems, IEEE, 2008:960-963.

[7] Huang S J, Chen S G. A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802. 15. 3c systems[J]. Circuits & Systems I Regular Papers IEEE Transactions on, 2012, 59(8):1752-1765.

[8] 李涛,杨婷,易学渊,等. 萤火虫 2:一种多态并行机的硬件体系结构[J]. 计算机工程与科学, 2014, 36(2): 191-200.

[9] 李涛,肖灵芝. 面向图形和图像处理的轻核阵列机结构[J]. 西安邮电大学学报, 2012, 17(3):41-47.

[10] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series[J]. Mathematics of Computation, 1965, 19(19):297-301.

[11] LI T. PAAG: A polymorphic array architecture for graphics and image processing [C] // PAAP'2012, IEEE Computer Society CPS. IEEE,2012:242-249.

[12] Marowka A, Ramat Gan. Back to thin-core massively parallel processors[J]. IEEE Computer, 2011,44(12): 49-54.

作者简介:

常立博 男,(1985-),博士研究生,助教. 研究方向为计算机视觉、计算机系统结构. E_mail:chang_libo@163. com.

王时雨 男,(1984-),博士研究生. 研究方向为图像处理、计算机系统结构.

(上接第 99 页)

[9] Do T T, Lu Gan, Nguyen N, et al. Sparsity adaptive matching pursuit algorithm for practical compressed sensing[C] // Asilomar Conference on Signals, Systems and Computers. Pacific Grove, California: IEEE Press,2008:581-587.

作者简介:

隋 昊 女,(1992-),硕士研究生. 研究方向为语音识别. E-mail:Emily_Sui@163. com.

周 萍 女,(1961-),硕士,教授. 研究方向为语音识别与智能控制.

沈 昊 男,(1992-),硕士研究生. 研究方向为语音识别.

杨 青 女,(1976-),硕士,副教授. 研究方向为语音识别算法与仿真.