计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20160123
53(7):1438-1446,2016

DSP 芯片中的高能效 FFT 加速器

雷元武　陈小文　彭元喜
(国防科学技术大学计算机学院　长沙　410073)
(leiyuanwu@163.com)

A High Energy Efficiency FFT Accelerator on DSP Chip

Lei Yuanwu, Chen Xiaowen, and Peng Yuanxi
(*College of Computer*, *National University of Defense Technology*, *Changsha* 410073)

Abstract　Fast Fourier transform (FFT) is a most time-consuming algorithm in the domain of digital signal processing (DSP). The performance and energy efficiency of FFT will make significant effect on different DSP applications. Thus, this paper presents a high energy efficiency variable-size FFT accelerator based on matrix transposition on DSP chip. Several parallel schemes are employed to exploit instruction level parallel and task level parallel of batch of small-size FFTs or big-size Cooley-Tukey FFT. A "Ping-Pong" structure of multi-bank data memory (MBDM) is presented to overlap the overhead of data move and FFT calculation. Moreover, based on MBDM, fast matrix transposition algorithm with basic block transposition is presented to avoid the matrix access with column-wise and improve the utilization of DDR bandwidth. Hybrid twiddle factor generating scheme, combining lookup table and on-line calculation with CORDIC, is presented to reduce the hardware for twiddle factor. Experimental results show that our FFT accelerator prototype with power efficiency of 146 GFLOPs/W, achieves energy efficiency improvement by about two orders of magnitude with multi-thread FFTW on Intel Xeon CPU.

Key words　fast Fourier transform (FFT); accelerator; high energy efficiency; matrix transposition; digital signal processing (DSP)

摘　要　快速傅里叶变换(fast Fourier transform, FFT)是数字信号处理(digital signal processing, DSP)领域中最耗时的核心算法,该算法的计算性能和计算效率将影响整个应用的执行效率.因此,在DSP芯片上设计实现了一个基于矩阵转置操作的高能效可变长度FFT加速器,采用多种并行策略开发批量小规模FFT算法与大规模Cooley-Tukey FFT算法中指令级和任务级并行.设计"乒乓"多体数据存储器,重叠数据搬移和FFT计算之间的开销,提高FFT加速器计算效率.并基于此存储器,提出基于基本块的快速矩阵转置算法,从而避免对数据矩阵的列访问;提出混合旋转因子产生策略,结合查表和基于CORDIC算法在线计算方式,最大限度降低旋转因子产生的硬件开销.实验结果表明:FFT加速器原型的峰值能效为146 GFLOPs/W,相比Intel Xeon CPU上的多线程FFTW实现,取得2个数量级的能效提升.

收稿日期:2016-03-07;**修回日期**:2016-05-14
基金项目:国家自然科学基金项目(61402499,61502508);湖南省自然科学基金项目(2015JJ3017)
This work was supported by the National Natural Science Foundation of China (61402499, 61502508) and the Natural Science Foundation of Hunan Province of China (2015JJ3017).
通信作者:陈小文(xwchen@nudt.edu.cn)

关键词 快速傅里叶变换;加速器;高能效;矩阵转置;数字信号处理

中图法分类号 TP302

随着半导体技术的发展,单芯片晶体管数量和性能持续以摩尔定律方式增加,当前单芯片已经能够集成几十亿个晶体管,计算能力达到 TFLOPS 量级.然而,由于晶体管的物理结构和特征没有发生本质变化,单个晶体管的能耗并未随工艺进步而大幅降低,使得功耗成为处理器性能提升的瓶颈. Hadi 等人认为:由于能耗的限制,未来芯片上的功能单元不可能同时持续处于峰值运行状态,这导致实际运行时往往无法将芯片所有的性能完全发挥出来,即所谓"暗场硅晶"现象[1].

针对特定应用定制加速器是提升计算效率、缓解功耗问题的一种有效的方法.针对特定应用设计相应的硬件结构,使得计算和存储资源最佳匹配应用的并行特征和存储模式获得最高能效.目前,多种新型高效能芯片采用通用处理核和加速计算核相结合的异构多核体系结构,如 OpensPlySER[2],QuicklA[3].

快速傅里叶变换(fast Fourier transform, FFT)是数字信号处理(digital signal processing, DSP)领域中的基本操作,广泛应用于声纳、图像、雷达、电信和无线信号处理等领域.通常,在这些应用中,FFT 算法的计算开销所占比重较大. FFT 算法的计算效率将直接影响整个应用的执行效率.然而,在不同的平台上,FFT 算法的执行效率相差很大.在通用处理平台,如 CPU,GPU 等处理芯片[4-5],FFT 算法的执行效率较低,达不到峰值性能的 10%[4].这是因为 FFT 计算过程中需要以 2 的幂次方为步长交叉访问数据,这与通用处理器的组相联 Cache、组相联地址映射和 DDR 存储器的突发访问机制不匹配,从而无法充分发挥芯片的计算性能,尤其对于大规模 FFT 计算,不能将整行数据存储到 Cache 或者片上存储器中,需要频繁访问片外存储器,存储带宽利用率较低.

针对 FFT 算法的计算特征和存储特征定制的 FFT 加速器能够获得高性能、低功耗和高能效. Chung 等人预测:相比 GPU 和 CPU,定制 FFT 加速器能够获得 100 倍和 1 000 倍的性能加速、2 个数量级的效能提升[6].学者们提出了不同流水和并行 FFT 体系结构,如阵列并行结构[7-8]、单通路延时转换结构(SDC)[9]、单通路延时反馈结构(SDF)[10]、多通路延时转换结构(MDC)[11]、多通路延时反馈结构(MDF)[12]等.但是上述 FFT 处理器都是针对通信应用设计的,如 OFDM,LTE 和 WiMax 等,支持的最大规模通常小于 2048 点,这很难满足雷达等大规模 FFT 应用.

目前,部分 DSP 芯片内部集成了 FFT 加速器,如 TI C55X 系列 DSP 芯片包含一个紧耦合 FFT 加速器(称为 HWAFFT)[13],通过使用加速器指令实现 HWAFFT 与 DSP 内核协同工作.然而,HWAFFT 仅支持 32 位定点格式、规模不超过 1024 点的 FFT 计算,这限制了 HWAFFT 的应用范围.

对于不同 DSP 应用,FFT 运算规模差别较大,例如在数字和无线电通信应用中,FFT 规模通常为 K 量级(10^3),而对于雷达、声纳等应用,FFT 规模能够可达兆(M)量级(10^6).因此,需要为可变规模 FFT 算法提供高性能和高能效支持,以满足实时信号处理领域中不同应用需求.我们先前工作[14]在 DSP 芯片上设计了一个支持可变规模无转置操作的 FFT 加速器,利用 DSP 芯片上 SRAM 存储器的随机访问特性来避免大规模 Cooley-Tukey FFT 计算过程中对 DDR 进行矩阵列访问[15].然而,该方法需要增加 2 次 SRAM 和 DDR 之间的搬移步骤(文献[14]图 1(B)中的 Step1 和 Step4 或图 1(C)中的 Step1 和 Step5),同时,DSP 芯片上的数据网络宽度通常为 128 b(或 256 b),在文献[14] 图 1(B)和图 1(C)中的 Step2-1 和 Step2-2 矩阵列读和列写过程中,每个时钟周期同时读出 2 列(或 4 列)数据,导致数据读/写和 FFT 计算无法重叠,从而降低 FFT 计算效率.

本文在先前工作[14]的基础上进行优化,主要贡献如下:

1) 设计了"乒乓"多体数据存储器(multi-bank data memory, MBDM),重叠数据搬移和 FFT 计算之间的开销,提高 FFT 加速器计算效率.

2) 提出基于基本块的快速矩阵转置算法,复用 MBDM,避免对数据矩阵的列访问.

3) 在 DSP 芯片上设计实现了基于此快速矩阵转置算法的可变规模 FFT 加速器,增加 3 次快速矩阵转置来避免文献[14]中的存储问题.

4) 提出混合旋转因子产生策略,结合查表和基于 CORDIC 算法在线计算方式,最大限度降低旋转因子产生的硬件开销.

1 Cooley-Tukey FFT 算法

FFT 加速器中,至少需要将一行数据存储到加速器内部数据存储器中,因此,FFT 加速器所需存储容量随 FFT 规模的增大而线性增加. 存储器容量将成为大规模 FFT 加速器设计的主要限制.

本文采用 Cooley-Tukey FFT 算法来实现大规模 FFT. 该算法采用分而治之的思想,使用规模较小的二维 FFT 模拟实现规模较大的一维 FFT. 对于 $NF=N_1\times N_2$ 点的 FFT 可以用 N_2 个 N_1 点和 N_1 个 N_2 点的 FFT 算法来实现,迭代公式为

$$X[k_1N_2+k_2]=\sum_{n_1=0}^{N_1-1}\left[e^{-j\frac{2\pi n_1k_2}{N}}\left(\sum_{n_2=0}^{N_2-1}x[n_2N_1+n_1]e^{-j\frac{2\pi n_2k_2}{N_2}}\right)\right]e^{-j\frac{2\pi n_1k_1}{N_1}},$$

其中,$0\leqslant k_1<N_1$,$0\leqslant k_2<N_2$.

从上述公式和图 1(a)可以看出,将初始数据从逻辑上看成按行存储为 $N_1\times N_2$ 的矩阵,Cooley-Tukey FFT 算法可以分为 3 个步骤:

步骤 1. 列方向 FFT 计算. 进行 N_2 次的 N_1 点 FFT 运算,即执行 N_2 次公式:

$$\sum_{n_2}^{N_2-1}x[n_2N_1+n_1]e^{-j\frac{2\pi n_2k_2}{N_2}}.$$

步骤 2. 补偿旋转因子计算. 步骤 1 的结果乘以补偿旋转因子 $e^{-j\frac{2\pi n_1k_2}{N}}$.

步骤 3. 行方向 FFT 计算. 在步骤 2 的基础上进行 N_1 次 N_2 点 FFT 运算,即执行 N_1 次公式:

$$\sum_{n_1=0}^{N_1-1}[e^{-j\frac{2\pi n_1k_2}{N}}(\sum_{n_2=0}^{N_2-1}x[n_2N_1+n_1]e^{-j\frac{2\pi n_2k_2}{N_2}})]e^{-j\frac{2\pi n_1k_1}{N_1}}.$$

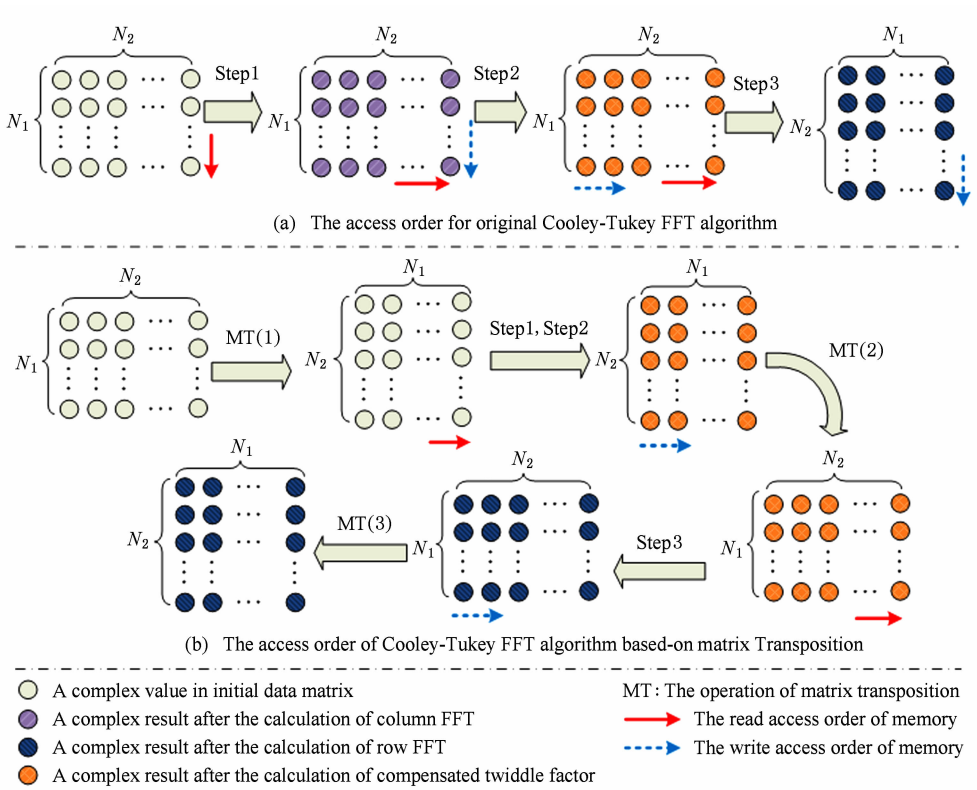


Fig. 1 The access order of Cooley-Tukey FFT algorithm.

图 1 Cooley-Tukey FFT 算法中数据访问顺序

2 FFT 加速器的实现

2.1 总体结构

如图 2 所示,全定制并行 FFT 加速器挂载于 DSP 芯片上的高速数据网络中、是数据网络中的一个主机,通过 AXI 主机数据接口访问到 DSP 芯片内部存储器(DSP 内核中的 Cache、共享存储器 SMC 等)和芯片外部存储器(如 DDR3 存储器等). 同时,FFT 加速器通过 AXI 从机命令通路完成 DSP 内核与 FFT 加速器之间的命令交互. FFT 加速器接收来自 DSP 内核的启动命令及参数配置,然后从 SMC 或 DDR3 中读取数据进行指定规模的 FFT 运算,并将结果写入到指定位置,最后通过

AXI 从机命令通路查询方式或者中断方式将完成信号及完成情况返回给 DSP 内核.

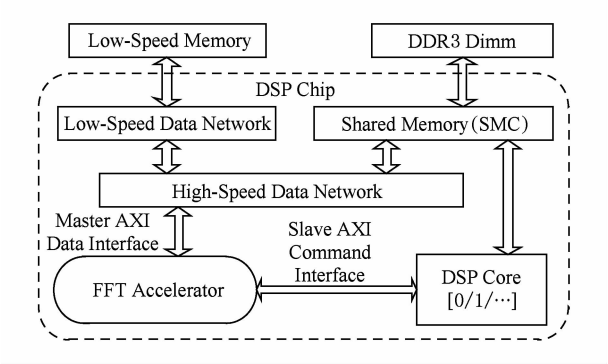


Fig. 2 The position of FFT accelerator on DSP chip.

图 2 DSP 芯片上 FFT 加速器的位置

如图 3(a)所示,FFT 加速器主要由 FFT 控制器、DMA 控制器、FFT-PE 阵列和异步数据/命令 FIFO 组成. 异步数据/命令 FIFO 用于隔离 FFT 加速器时钟域(FFT_CLK)和 DSP 芯片时钟域(SYS_CLK),使得 FFT 加速器频率不再受 DSP 系统频率限制,可通过单独调节 FFT_CLK 来匹配 FFT-PE 阵列计算带宽和 AXI 数据网络带宽,从而获得最佳能效.

FFT 控制器控制着整个 FFT 加速器的运行,将批量 FFT 计算或大规模 FFT 计算分解为若干子操作序列,如 DMA 读、DMA 写、DMA 矩阵转置和小规模 FFT 计算,并将这些子操作分配到各个执行单元,在保证数据相关前提下尽可能通过重叠来隐藏延时,提高 FFT 加速器性能.

DMA 控制器负责完成数据在 SMC,DDR 和 FFT 数据存储器之间搬移,该模块将 DMA 读、DMA 写、DMA 矩阵转置等子操作转换为一系列 AXI 总线事务,再以突发方式访问片上 SMC 或者片外 DDR 存储器.

FFT 计算阵列完成所有计算任务,由 N 个单存储器结构的 FFT-PE 和 CORDIC 补偿旋转因子产生逻辑组成,并行执行多行小规模 FFT 运算. 如图 3(b)所示,FFT-PE 由 M 个并行的蝶形运算单元、

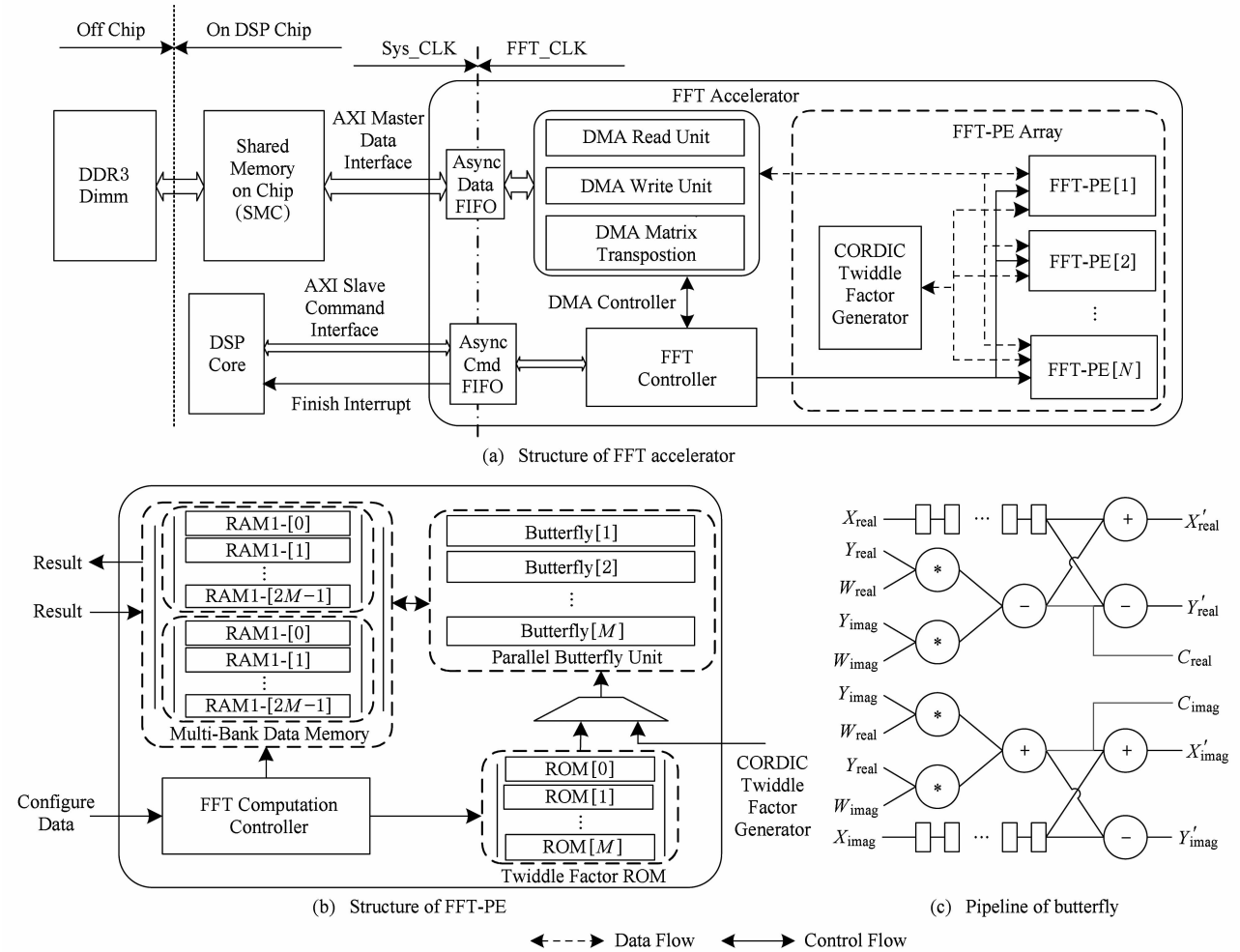


Fig. 3 Structure of FFT accelerator.

图 3 FFT 加速器结构

FFT-PE 计算控制器、“乒乓”多体结构数据存储器、旋转因子存储器等组成,每个 FFT-PE 能够完成一行小规模 FFT 计算,直接支持最大规模为 Nm. 在 FFT 计算过程中,同一级的蝶形运算不存在数据相关,可以并行执行. 因此,每个 FFT-PE 设置 M 个蝶形运算并行完成同一级 FFT 计算,同时每个蝶形运算单元采用全流水技术提高计算吞吐率. 如图 3(c)所示,蝶形运算单元由 6 个单精度浮点乘法和 4 个单精度浮点加法组成,采用复数乘法与蝶形运算的复用结构,能够以流水方式完成蝶形运算或复数乘法,减少中间计算结果的规格化操作,降低硬件开销,减少计算延时.

FFT 加速器支持的运算规模 NF 为 2 的幂次方,根据规模 NF 的大小,将 FFT 分为:

1) 小规模 FFT 算法. 规模 NF 不超过 Nm 的 FFT 运算,通过 DMA 方式从 DDR 存储器或 SRAM 中读取数据,FFT-PE 直接完成一维 FFT 运算,结果写回到存储器指定位置.

2) 大规模 FFT 算法. 规模 NF 满足 $Nm<NF<Nm^2$,采用 Cooley-Tokey 二维 FFT 算法实现. 如图 1(b)所示,FFT 加速器自动完成 3 次数据矩阵的转置操作和 2 个批量小规模 FFT 计算,最后将结果写入到指定位置.

2.2 混合旋转因子产生策略

旋转因子可以通过查表或者在线计算这 2 种方式得到. 查表方式可以快速获得旋转因子,然而查找表的存储需求随 FFT 规模 NF 而线性增加,为 $4\times NF$ 字节,对于大规模 FFT 算法,在芯片内部存储所有旋转因子的硬件开销巨大. 在线计算方式通常采用数字迭代算法,如 CORDIC 算法[16],计算相应角度正弦值和余弦值,这种方法能够计算出任意规模 FFT 的旋转因子. 然而,以全流水方式实现正弦和余弦在线计算的硬件开销较大. 本文提出混合旋转因子产生策略,结合查表和在线计算方式,满足 Cooley-Tukey FFT 算法需求,最大限度降低硬件开销.

从 Cooley-Tukey FFT 算法可以看出,步骤 1 列方向 FFT 计算和步骤 3 行方向 FFT 计算可视为批量小规模 FFT 计算在 N 个 FFT-PE 上执行,FFT 规模不超过 Nm. 在每个 FFT-PE 中,M 个蝶形计算单元为了获得流水吞吐率,每个时钟周期需要提供 M 个旋转因子,因此,在 FFT-PE 内部设置基于查表方式的多体旋转因子 ROM(M 个单端口 $(Nm/2)\times 64$ 的 ROM,如图 3(b)所示),为 M 个蝶形计算单元提供旋转因子数据.

Cooley-Tukey FFT 算法步骤 2 补偿旋转因子计算过程中,需要 NF 个补偿旋转因子(对于 FFT 加速器支持最大规模为 Nm^2),采用查表方式的存储需求将达到 M 量级,硬件开销较大. 然而,在整个 Cooley-Tukey FFT 计算过程中,每个补偿旋转因子只使用一次. 我们设计了基于 CORDIC 算法的补偿旋转因子计算引擎,采用 41 级全流水方式实现. 由于步骤 2 补偿旋转因子计算与步骤 1 和步骤 3 批量 FFT 计算存在数据相关性,需要串行执行,将步骤 1 和步骤 2 相结合,步骤 2 视为一级 FFT 计算,以减少数据搬移,如图 1(b)中 Step1 和 Step2. 同时,相对于步骤 1 和步骤 3,步骤 2 所需时钟周期较少,多个 FFT-PE 可以共享同一个 CORDIC 补偿旋转因子计算引擎,进一步提高硬件资源利用率.

2.3 “乒乓”多体数据存储器(MBDM)

每个 FFT-PE 内部设置一个 2 级多体数据存储器来存储初始数据、中间结果和最终结果,如图 3(b)所示. 第 1 级由 2 组多体存储器组成“乒乓”结构,用于重叠数据搬移与 FFT 计算,提高数据通路带宽和蝶形运算单元的利用率;第 2 级由 $2M$ 个 $\frac{Nm}{M}\times 64$ b 双端口 RAM 组成(RAMi_[0],RAMi_[1],…,RAMi_[$2M-1$],i 表示组号为 0 或者 1),为 M 个蝶形计算单元提供数据.

每组多体数据存储器的数据以体低位地址交叉方式进行组织,如图 4 所示,最大限度将同时访问的数据存储在不同存储体中. FFT 加速器运算是原位运算,即参与蝶形运算数据的原始位置和结果位置相同,对于第 i 级 FFT 运算,同一蝶形操作的 2 个数据间隔为 2^{i-1},针对多体存储器,采用如下访问策略:

RAMi_[0]	0	$2M$	…	$Nm-2M$
RAMi_[1]	1	$2M+1$	…	$Nm-(2M-1)$
⋮	⋮	⋮		⋮
RAMi_[$2M-1$]	$2M-1$	$4M-1$	…	$Nm-1$

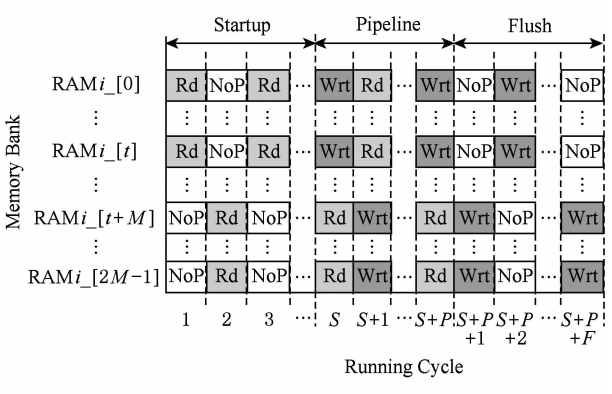
Fig. 4 Organization of data in MBDM.

图 4 多体数据存储器的数据组织方式

1) 第 i 级 FFT 计算,其中 $2^{i-1}\leqslant M$. 蝶形运算 2 个数据间隔不超过 M,即这 2 个数据存储在不同的存储体中. 每个存储体的 2 个端口设置为一个读端口和一个写端口,同时从 2 个不同存储体中的读端口中读取数据,蝶形运算结果通过写端口写入各自存储体中.

2) 第 i 级 FFT 计算,其中 $2^{i-1}>M$. 蝶形运算

2 个数据存储在同一存储体中.因此,只能通过该存储体的 2 个端口同时读才能获得蝶形运算的 2 个数据;对于结果,需要将存储体的 2 个端口设置为写端口,同时完成写操作.然而,$2M$ 个存储体中,每个时钟周期最多 M 个存储体进行读操作和 M 个存储体进行写操作.因此,可以采用存储体交叉方式流水读取数据,然后,再以交叉方式将数据写入,避免存储体的读写端口冲突,如图 5 所示:



NoP refers to no operation in this cycle

Fig. 5 The space-time graph of "Ping-Pong" access mode of MBDM.

图 5 多体数据存储器乒乓访问方式时空图

3 基于基本块的矩阵转置算法

FFT 加速器通过 AXI 主机数据接口以突发方式访问数据,需尽量避免对数据矩阵的列方式访问.然而,如图 1(a)所示,在原始 Cooley-Tukey FFT 算法需要对数据矩阵进行 3 次列访问(步骤 1 的读矩阵、步骤 1 的写矩阵和步骤 3 的写矩阵).

本文利用 FFT 加速器内部数据存储器(MBDM)来实现快速矩阵转置,从而避免对数据矩阵的列方式访问.本文提出基于基本块的矩阵转置策略:

1) 将原始 $N_1\times N_2$ 矩阵分解若干基本块 $NB\times NB$(方阵,NB 为 2 的幂次方),通过一次按行读和按列写操作完成基本块转置;

2) 以基本块为元素,对基本块矩阵进行转置,实现整个矩阵的转置,基本块矩阵转置通过控制基本块转置的读、写地址完成;

3) 充分利用 AXI 数据通路的读通路与写通路完全分开,可以重叠读操作和写操作的特征,设计"乒乓"结构重叠不同基本块的读操作和写操作.

3.1 基本块转置算法

FFT-PE 阵列的内部数据存储器能够存储 $2\times N\times Nm$ 点数据,为了能够实现"乒乓"结构,将所有 FFT-PE 数据存储分成 2 组,每组存储的基本块为 $NB\times NB$,其中 $NB=2^{\left\lfloor \mathrm{lb}\sqrt{N\times Nm}\right\rfloor}$.图 6 举例说明基本矩阵转置的数据存储方式,其中 $N=2$,$M=2$,$Nm=1024$,$NB=32$,FFT-PE 每组数据存储器由 4 个存储体组成.基本块转置过程有 2 步骤:

步骤 1. 以行的顺序连续读取一个基本块的数据,并以体交叉方式存储到一组 FFT-PE 的数据存储器中.基本块中相邻行列号相同的数据存储到不同存储体内,这样保证按列读取时能同时取出相邻行相同位置的数据.对于基本块中任意数据 $E[i][j]$ $(0\leqslant i,j<NB)$,该数据写入的存储体和体内地址为

$$\begin{cases} Num\_Bank=((i\%NB)+j)\%NB,\\ Addr\_Bank=\left\lfloor j/NB\right\rfloor. \end{cases}$$

步骤 2. 以列顺序连续将数据从 FFT-PE 的数据存储器 MBDM 中读出,并以突发方式写入到 SMC 或 DDR.

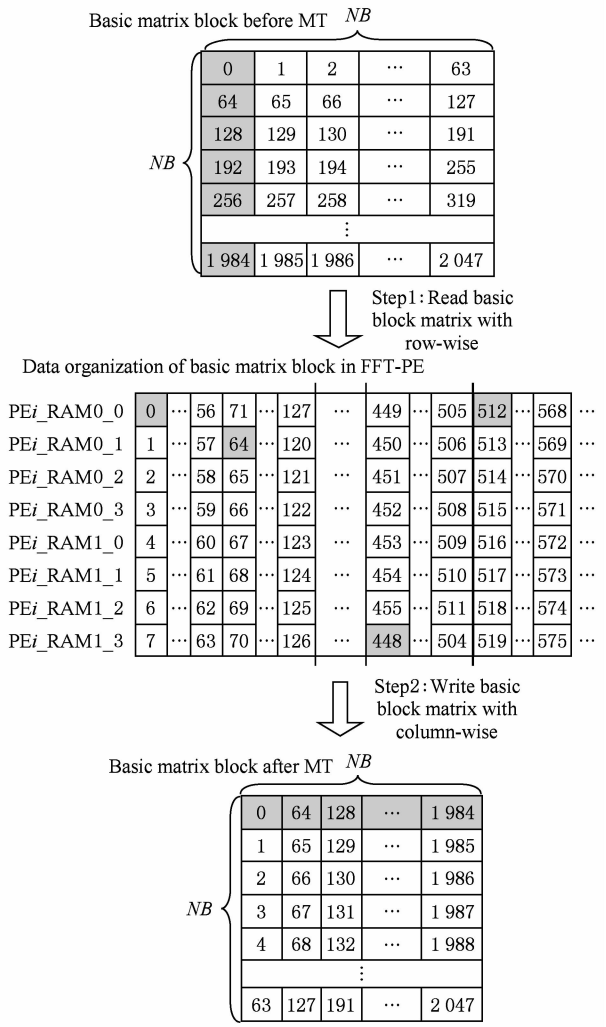


Fig. 6 The organization of basic block matrix in MBDM.

图 6 多体数据存储器中的基本块组织

3.2 块矩阵转置算法

块矩阵转置如图 7 所示,对于规模为 NF 的 FFT 计算($NF>1024$),NF 点的一维数据可视为二维矩阵($NB\times R$)×($NB\times C$),其中 $R>0$,$C>0$. 假定初始数据矩阵起始地址为 A_I,转换后数据矩阵起始地址为 A_R,则矩阵中任意一个数据(序号为 i,j),则行列位置:

$$NR=\left\lceil i/(NB\times C)\right\rceil;$$

$$NC=i\%(NB\times C).$$

基本块的行列位置:

$$NBR=\left\lceil \frac{NR}{NB}\right\rceil=\left\lceil \left\lceil \frac{i}{NB\times C}\right\rceil \Big/ NB\right\rceil;$$

$$NBC=\left\lceil \frac{NC}{NB}\right\rceil=\left\lceil \frac{i\%(NB\times C)}{NB}\right\rceil.$$

基本块内的行列位置:

$$NER=NR\%NB=\left\lceil \frac{i}{NB\times C}\right\rceil\%NB;$$

$$NEC=NC\%NB=(i\%(NB\times C))\%NB.$$

矩阵转置后对应的地址位置:

$$Addr\_Trans[i]=A\_R+NER+NB\times NBR+NB\times C\times(NB\times NBC+NEC).$$

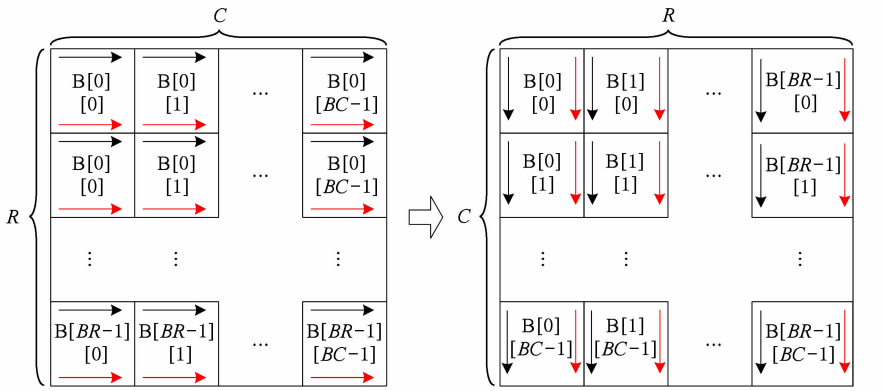


Fig. 7 The transposition of basic block matrix.

图 7 基本块矩阵转置

块矩阵转置采用“乒乓”方式重叠读、写时间开销,首先以行块的顺序读取基本块,以列块顺序写入到目标地址中,除第 1 个基本块的读和最后一个基本块的写不能重叠外,其余的都能重叠,最大化利用读、写通路的数据带宽.

4 实验及结果

4.1 原型系统

在 DSP 芯片上设计实现了上述 FFT 加速器原型. 该 DSP 芯片的片上共享存储器 SMC 的容量为 4 MB,系统时钟频率为 500 MHz,峰值 IO 带宽为 8 GBps. FFT 加速器原型包含 2 个 FFT-PE,每个 FFT-PE 包含 2 个全流水蝶形运算单元,FFT-PE 直接支持 FFT 的最大规模为 1 024,即 $N=2$,$M=2$,$Nm=1024$. FFT-PE 内部的“乒乓”结构多体数据存储器由 8 个子存储体组成,每个子存储体为一个 256×64 双端口 RAM.

在 45 nm 工艺库下,对 FFT 加速器原型进行综合,关键路径延时为 0.91 ns、面积为 1.22 mm^2、功耗为 273 mW. FFT-PE 计算阵列负责完成 FFT 算法中所有的计算工作,所占面积较大,约占 86.3%. FFT 加速器工作在 1 GHz 下,能够取得单精度浮点峰值性能和能效分别为 40 GFLOPs 和 146 GFLOPs/W.

4.2 性能比较与分析

表 1 对比了 3 个不同平台执行上小规模 FFT 的性能,这些平台分别为 TI TMS320C55X DSP 芯片上的 FFT 加速器 HWAFFT[13]、我们先前工作[14]——基于 SRAM 的无转置可变规模 FFT 加速器和 Intel Xeon X5675 CPU 平台上的 FFTW 函数库. Intel CPU 为 6 核 12 线程处理器,运行频率为 3.07 GHz,在此处理器上运行最快 FFT 函数库——FFTW-3.3-DDL64 [17],该函数库利用 Intel 处理器 SSE4 SIMD 指令对 FFT 算法进行自动向量化,加速 FFT 执行. 相对于 Intel Xeon CPU 平台上的单线程 FFTW,我们能够取得 1.14~1.65 倍的性能提升,同时取得 2 个数量级的能效提升(124 GFLOPS/W 比 0.14 GFLOPS/W). 相比于 TI DSP 芯片上的紧耦合 FFT 加速器 HWAFFT,本文的设计能够取得 2.65~4.22 倍的性能提升. 相比于我们先前工作[14],本文在数据存储方式、并行任务调度和同步等方面进行优化,性能略有提升.

Table 1 Performance Comparison Among Different Platforms for Small Size FFT

表 1 不同平台上小规模 FFT 性能对比

Platforms	Items	FFT Size Point		
		64	256	1024
FFTW[18] on Intel CPU	Running Time/μs	0.171	0.642	3.158
	Performance/GFLOPs	11.23	15.95	16.21
TI HWAFFT[13]	Running Time/μs	0.436	1.668	7.315
	Performance/GFLOPs	4.40	6.14	7.00
Our Previous Work[14]	Running Time/μs	0.142	0.576	2.806
	Performance/GFLOPs	13.52	17.78	18.24
Our FFT Accelerator	Running Cycle	93	378	1505
	Running Time/μs	0.093	0.378	1.505
	Performance/GFLOPs	18.60	18.52	18.53
	Speedup vs FFTW	1.65	1.16	1.14
	Speedup vs HWAFFT	4.22	3.02	2.65
	Speedup vs Ref[14]	1.38	1.04	1.02

图 8 对比了 3 个不同平台上执行大规模 FFT 的性能,在 Intel Xeon CPU 上分别使用单线程和多线程(12 线程)版本的 FFTW 函数库的性能进行比较,而在 TI TMS320C6678 DSP 芯片上采用优化后的单核和 8 核程序[18]的执行性能进行比较. 图 8(b)

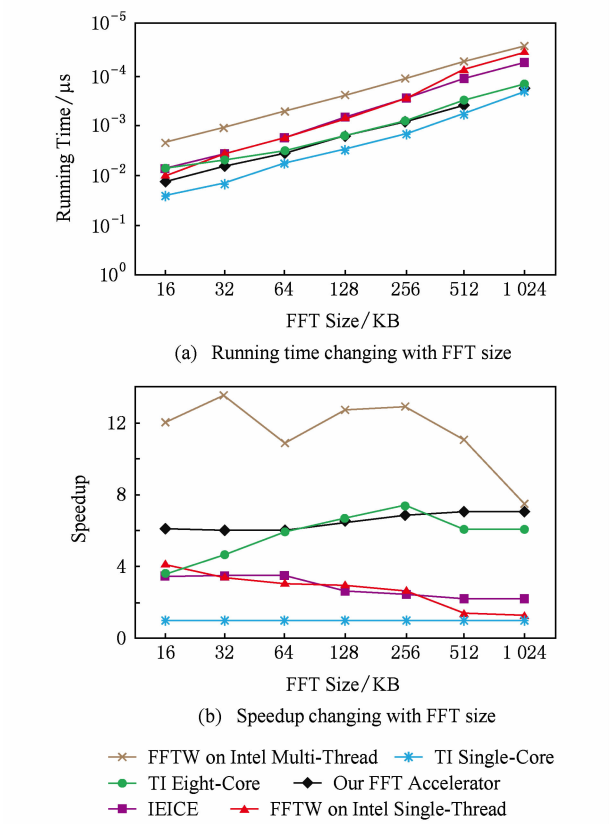


Fig. 8 Performance comparison among different platforms for big size FFT.

图 8 不同平台上大规模 FFT 性能对比

中的加速比以 TI TMS320C6678 DSP 上单核性能来评估. 由于 Intel Xeon CPU 的 L2 Cache 容量为 256 KB,能够存储 32×10^3 点 FFT 的数据,多核上执行多线程 Cooley-Tukey FFT 通过 L2 Cache 进行数据交互. 对于 FFT 规模超过 32×10^3 点,多线程 FFTW 必须通过 L3 Cache 或外部存储器进行数据交互,性能会有所降低. 因此,CPU 上的多核程序执行 32×10^3 点 FFT 运算时取得最高计算性能. 相对于 Intel CPU 上的多线程 FFTW,本文提出的 FFT 加速器性能略低,但是 Intel CPU 的功耗为 FFT 加速器的 350 倍以上(95 W 比 0.27 W),FFT 加速器能够取得 2 个数量级的能效提升. 对于 1×10^6 点 FFT 算法,本文的 FFT 加速器与多线程 FFTW 的性能相当.

TI TMS320C6678 DSP 芯片的共享 L2 存储器的容量为 4 MB,能够存储 256×10^3 点 FFT 的数据,8 核 FFT 程序通过 L2 存储器进行数据交互. 因此,多核 DSP 版本的 FFT 程序在 256×10^3 点时的性能最高. 与单核 DSP 和多核 DSP 相比,本文的 FFT 加速器能够获得 6.6 倍和 1.2 倍的平均性能提升.

与我们先前工作[14]相比,本文基于 FFT 加速器内部多体数据存储器实现快速矩阵转置,并设计了基于矩阵转置操作的 Cooley-Tukey FFT 算法,避免文献[14]中的矩阵列访问时 FFT 加速器中计算单元利用率不高、数据搬移开销与 FFT 计算开销无法有效重叠的问题. 相比于文献[14],对于大规模 FFT 加速器,本文的 FFT 加速器能够获得 2.3 倍的平均性能提升.

5 结 论

在 DSP 芯片上设计实现了一个基于矩阵转置操作的高能效 FFT 加速器,采用多种并行策略、混合旋转因子产生策略和"乒乓"结构多体数据存储器来提升 FFT 加速器性能和计算效能,并基于 FFT 加速器内部多体数据存储器,提出基于基本块的快速矩阵转置算法,避免对矩阵的列访问,进一步提升 FFT 加速器能效.

参 考 文 献

[1] Hadi E, Emily B, Remnee S A, et al. Dark silicon and the end of multicore scaling [C] //Proc of the 38th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2011: 365-376

[2] Jesse B, Ryan C, Chris F, et al. Design, integration and implementation of the DySER hardware accelerator into OpenSPARC [C] //Proc of the 18th IEEE Int Symp on High performance Computer Architecture. Piscataway, NJ: IEEE, 2012

[3] Nagabhushan C, Ganapati S, Scott H, et al. QuicklA: Exploring heterogeneous architectures on real prototypes [C] //Proc of the 18th IEEE Int Symp on High performance Computer Architecture. Piscataway, NJ: IEEE, 2012

[4] James R G, Sharon M S. A transpose-free in-place SIMD optimized FFT [J]. ACM Trans on Architecture and Code Optimization, 2012, 9(3): 23:1-23:21

[5] Li Y, Zhang Y, Jia H, et al. Automatic FFT performance tuning on OpenCI GPUs [C] //Proc of the 7th IEEE Int Conf on Parallel and Distributed Systems. Piscataway, NJ: IEEE, 2011: 228-235

[6] Chung E S, Milder P A, Hoe J C, et al. Single-chip heterogeneous computing—Does the future include custom logic, FPGAs, and GPU? [C] // Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchiecture. Piscataway, NJ: IEEE, 2010: 225-236

[7] Palmer J, Nelson B. A parallel FFT architecture for FPGAs [C] //Proc of the 14th Int Conf on Field Programmable Logic and Application. Piscataway, NJ: IEEE, 2004: 948-953

[8] Dou Y, Zhou J, Lei Y, et al. FPGA SAR processor with window memory accesses [C] //Proc of the 18th IEEE Int Conf on Application-Specific Systems, Architectures and Processors. Piscataway, NJ: IEEE, 2007: 95-100

[9] Cho T, Lee H. High-speed low-complexity modified radix-25 FFT processor for high rate WPAN applications [J]. IEEE Trans on Very Large Scale Integration Systems, 2013, 21 (1): 187-191

[10] Yu C, Yen M H. Area-efficient 128- to 2048/1536-point pipeline FFT processor for LTE and mobile WiMAX systems [J]. IEEE Trans on Very Large Scale Integration Systems, 2015, 23(9): 1793-1800

[11] Tang S N, Liao C H, Chang T Y. An area-and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems [J]. IEEE Journal of Solid-State Circuits, 2012, 47(6): 1419-1435

[12] Ayinala M, Brown M, Parhi K K. Pipelined parallel FFT architectures via folding transformation [J]. IEEE Trans on Very Large Scale Integration Systems, 2012, 20(6): 1068-1081

[13] Mckeown M. FFT implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs, SPRABB6B [R]. Dallas, Texas: Texas Instruments, 2013

[14] Guo L, Tang Y, Dou Y, et al. Transpose-free variable size FFT accelerator based on-chip SRAM [J]. IEICE Electron Express, 2014, 11(15): 1-8

[15] Guo L, Tang Y, Dou Y, et al. Window memory layout scheme for alternate row-wise/column-wise matrix access [J]. IEICE Trans on Information and Systems, 2013, E96-D (12): 874-885

[16] Jack E V. The CORDIC trigonometric computing technique [J]. IRE Trans on Electronic Computers, 1959, 8(3): 330-334

[17] Frigo M, Johnson S G. The design and implementation of FFTW3 [J]. Proceeding of the IEEE, 2005, 93(2): 216-231

[18] Li X, Blinka E. Very large FFT for TMS320C6678 processors, spry277 [R]. Dallas, Texas: Texas Instruments, 2015

Lei Yuanwu, born in 1982. Received his PhD degree in computer science and technology from National University of Defense Technology (NUDT) in 2012. Assistant researcher at the College of Computer, NUDT. His main research interests include high performance computer architecture, high-performance microprocessor and DSP design.

Chen Xiaowen, born in 1983. Received his PhD degree in microelectronics from NUDT in 2011. Assistant researcher at the College of Computer, NUDT. His main research interests include VLSI design, computer architecture, microarchitecture, SOC, and NOC.

Peng Yuanxi, born in 1966. Professor and PhD supervisor. Received his PhD degree in computer science from NUDT in 2001. His main research interests include high performance computing, on-chip networks, multi-core and many-core architectures.