矢量基二维 DCT 修剪在 DSP 上的内存存取减少方法

刘项洋,许 勇,郑孝遥,陈付龙
(安徽师范大学计算机与信息学院,安徽芜湖 241000)

摘 要: 本文针对矢量基二维 DCT 修剪提出内存存取减少方法. 该方法旨在减少计算中因权重因子和信号输入而导致的内存存取. 它首先利用权重因子的属性将计算流程图内每相邻两阶段内的蝴蝶运算单元进行融合,然后再以较少的权重因子来计算. 本文采用通用 DSP 处理器来验证该方法对矢量基二维 DCT 修剪算法的有效性. 并且实验结果显示该方法相比于常规方法可以大幅度减少运算所需的时钟周期数、降低对运算中对内存的存取量、以及占用更少的内存.

关键词: 数字信号处理器(DSP);离散余弦变换(DCT);蝴蝶运算单元;内存存取
中图分类号: TN911.23 **文献标识码:** A **文章编号:** 0372-2112 (2019)03-0757-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.03.034

Memory Access Reduction Method for Vector-Radix 2D DCT Pruning on DSP

LIU Xiang-yang, XU Yong, ZHENG Xiao-yao, CHEN Fu-long
(*School of Computer and Information, Anhui Normal University, Wuhu, Anhui* 241000, *China*)

Abstract: This paper proposes a novel memory reference reduction method for vector-radix 2D DCT pruning. This method aims to reduce the memory reference owing to weighting factors and signal input. The proposed method merges the butterflies at every neighboring two stages in the computation diagram, and then computes them with fewer weighting factors. Hardware platform based on general purpose processor is used to verify the effectiveness of the proposed method for vector-radix 2-D FCT pruning implementation. Experimental results validate the benefits of the proposed method with less clock cycle, less memory reference, and fewer memory space compared with the conventional implementation.

Key words: digital signal processor(DSP); discrete cosine transform(DCT); butterflies; memory access

1 引言

自从文献[1]定义了离散余弦变换(Discrete Cosine Transform,简称 DCT),它便在图像和视频应用中起着举足轻重的作用. 并且它也成为 JPEG、MPEG、H.264 等图形影像编码标准中的重要组成成分. 随着 DCT 的广泛应用,学术界和工业界逐渐提出了一些直接和间接计算 DCT 的算法[2]. 如近些年发表的并行 DCT[3]、分块 DCT[4]、矢量矩阵 DCT[5]以及近似矩阵 DCT[6]等.

目前二维 DCT 又可以分为直接计算和分解计算,其中分解计算是将二维 DCT 按行或列逐步分解成一维 DCT 来计算. 这种计算因为需要进行耗时的矩阵转置运算,所以其运算量远高于直接计算. 文献[7~11]提出了一些有效的直接计算二维 DCT 的算法. 矢量基二维 DCT 算法在他们当中具有最低的计算复杂度[11],并且其规整的运算流程也使它最适合做修剪运算(即只计算部分 DCT 系数).

目前所有二维 DCT 算法(包括矢量基二维 DCT)都基于相同输入和输出的 DCT 系数. 然而实际中只有低频 DCT 系数才保存最有用的图像信息,所以计算部分低频 DCT 系数可基本替代全部系数. 这种只计算部分 DCT 系数的运算称为修剪运算,它在一定程度上能减少 CPU 的运算量和内存存取.

矢量基二维 DCT 的修剪运算也因其规则的运算流程,使它可以在很多硬件和软件平台上运行,包括针对具体应用程序的集成硬件电路(ASIC)[12]. 可是硬件平台一般是针对具体条件下的应用,因此不具备扩展性,而软件平台虽具备扩展性,但速度又远低于同类的硬件平台. 因此本文最终采用一类特殊的专用处理器:数字信号处理器(Digital Signal Processor,简称 DSP)来作为实现平台. DSP 是一系列经过优化以专门运行各种信号处理程序的处理器,并且性能上兼具硬件和软件的

收稿日期:2018-02-22;修回日期:2018-11-26;责任编辑:梅志强
基金项目:国家自然科学基金(No.61772034);安徽省自然科学基金(No.1808085MF172)

长处.

可惜的是,DSP 中的内存存取非常耗时(如 TI TMS320C64x 中,写入内存的指令占用四个 CPU 时钟周期),也会因此产生大量功耗. 频繁的内存存取更是会严重降低 CPU 效率. 因此在 DSP 上高效实现矢量基二维 DCT 还是很困难. 尽管文献[13]做了一定算法改进,但测试平台单一,不能很好的证明其普适性. 并且实验数据是基于对代码进行优化后的模拟环境,因此也很难证明其在真实环境中的效果.

本文提出一个新的减少内存存取的方法来实现矢量基二维 DCT. 该方法利用权重因子的属性将计算流程图内每相邻两阶段内的蝴蝶运算单元进行融合,然后再以较少的权重因子来计算. 为了更好的检测方法的通用性,本文选取三个当前通用的 DSP 作为实验平台,从实验数据得知,该方法不仅可以大幅提高 CPU 的执行效率,显着减少运算中的内存存取,而且可以用更少的内存来保存权重因子.

2 矢量基二维 DCT 的修剪运算

2.1 矢量基二维 DCT

二维 DCT 中的离散信号 $x[m,n]$ $(m,n=0,1,2,\cdots,N-1)$ 可用如下公式来计算[2]:

$$X[k,l]=\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}x[m,n]\cos\left[\frac{\pi}{2N}(2m+1)k\right]\cdot\cos\left[\frac{\pi}{2N}(2n+1)l\right],\quad k,l=0,1,2,\cdots,N-1 \tag{1}$$

然后对该公式采用按频率抽取(DIF),则为

$$\begin{aligned}X[k,l]=&\sum_{m=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[m,n]\cdot\cos\left[\frac{\pi}{2N}(2m+1)k\right]\cos\left[\frac{\pi}{2N}(2n+1)l\right]\\&+\sum_{m=N/2}^{N-1}\sum_{n=0}^{N/2-1}x[m,n]\cdot\cos\left[\frac{\pi}{2N}(2m+1)k\right]\cos\left[\frac{\pi}{2N}(2n+1)l\right]\\&+\sum_{m=0}^{N/2-1}\sum_{n=N/2}^{N-1}x[m,n]\cdot\cos\left[\frac{\pi}{2N}(2m+1)k\right]\cos\left[\frac{\pi}{2N}(2n+1)l\right]\\&+\sum_{m=N/2}^{N-1}\sum_{n=N/2}^{N-1}x[m,n]\cdot\cos\left[\frac{\pi}{2N}(2m+1)k\right]\cos\left[\frac{\pi}{2N}(2n+1)l\right],\quad k,l=0,1,2,\cdots,N-1\end{aligned} \tag{2}$$

接着我们对 k,l 进行奇偶分组,就得到了:

$$\begin{aligned}X[2k,2l]=&\sum_{m=0}^{\frac{N}{2}-1}\sum_{n=0}^{\frac{N}{2}-1}\{x[m,n]+x[N-1-m,n]+x[m,N-1-n]\\&+x[N-1-m,N-1-n]\}\cdot\cos\left[\frac{\pi(2m+1)k}{2\times\frac{N}{2}}\right]\cos\left[\frac{\pi(2n+1)l}{2\times\frac{N}{2}}\right]\\=&\sum_{m=0}^{\frac{N}{2}-1}\sum_{n=0}^{\frac{N}{2}-1}x_{00}[m,n]\cdot\cos\left[\frac{\pi(2m+1)k}{2\times\frac{N}{2}}\right]\cos\left[\frac{\pi(2n+1)l}{2\times\frac{N}{2}}\right]\end{aligned}$$

$$X[2k+1,2l]=\sum_{m=0}^{\frac{N}{2}-1}\sum_{n=0}^{\frac{N}{2}-1}x_{10}[m,n]\cdot\cos\left[\frac{\pi(2m+1)(2k+1)}{2\times N}\right]\cos\left[\frac{\pi(2n+1)l}{2\times(\frac{N}{2})}\right]$$

$$X[2k,2l+1]=\sum_{m=0}^{\frac{N}{2}-1}\sum_{n=0}^{\frac{N}{2}-1}x_{01}[m,n]\cdot\cos\left[\frac{\pi(2m+1)k}{2\times(\frac{N}{2})}\right]\cos\left[\frac{\pi(2n+1)(2l+1)}{2\times N}\right]$$

$$X[2k+1,2l+1]=\sum_{m=0}^{\frac{N}{2}-1}\sum_{n=0}^{\frac{N}{2}-1}x_{11}[m,n]\cdot\cos\left[\frac{\pi(2m+1)(2k+1)}{2\times N}\right]\cdot\cos\left[\frac{\pi(2n+1)(2l+1)}{2\times N}\right]$$

其中

$$\begin{aligned}x_{ij}[m,n]=&x[m,n]+(-1)^{i}x[N-1-m,n]\\&+(-1)^{j}x[m,N-1-n]\\&+(-1)^{i+j}x[N-1-m,N-1-n]\end{aligned} \tag{3}$$

然后进一步分解直到一系列(2×2)点输入 DCT,便得到了按频率分解的矢量基二维 DCT[11]. 图 1 给出了(4×4)点输入的矢量基二维 DCT 修剪运算流程图.

它由蝴蝶计算和后序操作两部分构成. 蝴蝶计算部分由一系列蝴蝶运算单元组成. 它可以进一步划分为若干阶段,每个阶段都包含固定数量的蝴蝶运算单元. 从图中不难看出,蝴蝶计算与后序操作相比,包含了主要的 CPU 运算和内存存取. 因此本文将内存存取减少方法应用于蝴蝶计算部分.

2.2 矢量基二维 DCT 的修剪运算

修剪运算是只计算 DCT 系数中的低频部分,因为这部分包含了主要的图形信息. 这种部分运算一定程度上节省了 CPU 运算量以及内存存取. 图 1 列出了在 $N\times N$ ($=4\times4$)点输入的矢量基二维 DCT 修剪中,仅计算前 N_0($=2$)个低频 DCT 系数的情况.

3 内存存取减少方法

本文的方法,在具体实现中将分两步应用.

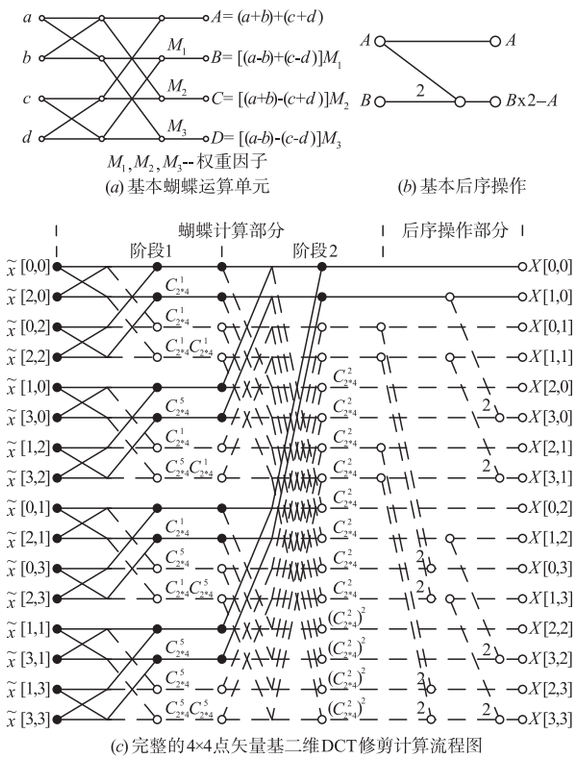


图1　4×4点矢量基二维DCT修剪

（注：黑点处表示由输入点$x[]$导致的内存存取，
而白点处因为没有任何计算，所以没有内存存取，
但与白点连接的虚线变为实线后，该白点也要变为黑点）

步骤 1　在计算 $N\times N$ 点矢量基二维 DCT 时,根据权重因子的余弦函数性质式(4),将它的数量由 $(N^2-1)/3$减少到$\lceil 4(N^2-1)/15\rceil$

$$C_{2\times N}^{2\times m}=(C_{2\times N}^{m})^2-(C_{2\times N}^{m+N})^2,\ m(0,N/2)$$

其中 $C_{2\times N}^{m}=\cos\dfrac{m}{2\times N}\pi,\ m=1,2,\cdots N/2-1$　　(4)

如图 1 所示,阶段 1 中四个蝴蝶运算单元需要分别与权重因子 $C_{2\times 4}^{1}$和 $C_{2\times 4}^{5}$结合在一起计算,而阶段 2 中四个蝴蝶运算单元需要与权重因子 $C_{2\times 4}^{2}$结合. 其中 $C_{2\times 4}^{2}$可用$(C_{2\times 4}^{1})^2-(C_{2\times 4}^{5})^2$ 来取代,其推导过程如下:

$$\begin{aligned}(C_{2\times 4}^{1})2-(C_{2\times 4}^{5})^2&=\left(\cos\frac{\pi}{2\times 4}\right)^2-\left(\cos\frac{5\pi}{2\times 4}\right)^2\\&=\left(\cos\frac{\pi}{2\times 4}\right)^2-\left[-\cos\left(\pi-\frac{5\pi}{2\times 4}\right)\right]^2\\&=\left(\cos\frac{\pi}{2\times 4}\right)^2-\left(\cos\frac{3\pi}{2\times 4}\right)^2\\&=\left(\cos\frac{\pi}{2\times 4}\right)^2-\left(\sin\frac{\pi}{2\times 4}\right)^2\\&=\cos\frac{2\times\pi}{2\times 4}\\&=C_{2\times 4}^{2}\end{aligned}$$

这样,只需用阶段 1 中权重因子 $C_{2\times 4}^{1}$和 $C_{2\times 4}^{5}$即可计算图 1 里蝴蝶计算部分的全部蝴蝶运算单元.

步骤 2　利用权重因子的属性将计算流程图内每相邻两阶段内的蝴蝶运算单元进行融合.

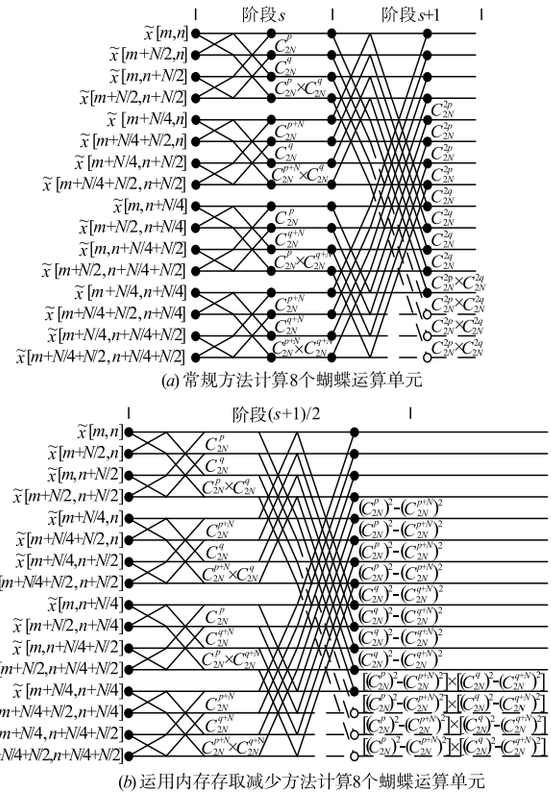


图2　用4个权重因子计算8个蝴蝶运算单元

一般情况下,在 $N\times N$ 点矢量基二维 DCT 计算流程图的蝴蝶计算部分中,第 s 阶段的蝴蝶运算单元具有 $N\times N/4^s$个不同的权重因子. 它们的排列如下:每个阶段中权重因子可以均分成四组,一般地,阶段 s 的第一个权重因子是 $C_{2\times N}^{k}$(其中 $k=2^{s-1}$并且 $C_{2\times N}^{k}=\cos\dfrac{k\pi}{2\times N}$). 四组内的蝴蝶运算单元需要结合四类权重因子:$C_{2\times N}^{p}$,$C_{2\times N}^{q}$,$C_{2\times N}^{p+N/2^{\lg N-s-1}}$和 $C_{2\times N}^{q+N/2^{\lg N-s-1}}$来计算. 其中 $C_{2\times N}^{p}$,$C_{2\times N}^{q}$是第一组中的前两类权重因子,$C_{2\times N}^{p+N/2^{\lg N-s-1}}$和 $C_{2\times N}^{q}$是第二组中的前两类权重因子,而第三组中的前两类权重因子分别是 $C_{2\times N}^{p+N/2^{\lg N-s-1}}$ and $C_{2\times N}^{q+N/2^{\lg N-s-1}}$,以及第四组中分别是 $C_{2\times N}^{p+N/2^{\lg N-s-1}}$ and $C_{2\times N}^{q+N/2^{\lg N-s-1}}$. 每组中的权重因子可以再进一步等分成具有类似结构的四组权重因子:$C_{2\times N}^{r}$,$C_{2\times N}^{t}$,$C_{2\times N}^{r+N/2^{\lg N-s-2}}$和 $C_{2\times N}^{t+N/2^{\lg N-s-2}}$,而且 $C_{2\times N}^{r}$和 $C_{2\times N}^{t}$不仅作为第一小组的前两类权重因子,也是进一步分组前就确定的前两类权重因子. 这种递归分组过程一直持续到每个最小组中仅需要 4 个权重因子:$C_{2\times N}^{r}$,$C_{2\times N}^{t}$,$C_{2\times N}^{r+N}$和 $C_{2\times N}^{t+N}$. 最后为了让输出规则有序,并方便自上往下逐个计算蝴蝶运算单元,本文对信号输入进行了重排序[14]. 图 2(a)给出

了第 s 和 $s+1$ 阶段中共 8 个蝴蝶运算单元相互之间的关系.

定理 1 $N\times N$ 点矢量基二维 DCT 修剪的蝴蝶计算部分中,只需用如图 2(b)所示的四个权重因子 $C_{2\times N}^{p}$, $C_{2\times N}^{q}$, $C_{2\times N}^{p+N}$和 $C_{2\times N}^{q+N}$即可计算第 s 和 $s+1$($s\leqslant\log_2 N-1$)阶段里的共 8 个蝴蝶运算单元.

证明 计算图 2(a)中第 $s+1$ 阶段的蝴蝶运算单元需要调用权重因子 $C_{2\times N}^{2p}$和 $C_{2\times N}^{2q}$,其中 $C_{2\times N}^{2p}=\cos\dfrac{2p}{2\times N}\pi$,因为有三角函数定理:$\cos(2A)=\cos^2 A-\sin^2 A$,权重因子 $\cos\dfrac{2p}{2\times N}\pi=\cos^2\dfrac{p}{2\times N}\pi-\sin^2\dfrac{q}{2\times N}\pi$,这里的 $\cos^2\dfrac{p}{2\times N}\pi=(C_{2\times N}^{p})^2$. 而 $\sin^2\dfrac{q}{2\times N}\pi$ 可利用三角函数属性转化:

$$\begin{aligned}\sin^2\frac{q}{2\times N}\pi&=\cos^2\left(\frac{\pi}{2}-\frac{q}{2\times N}\pi\right)\\&=\cos^2\left[\pi-\left(\frac{\pi}{2}-\frac{q}{2\times N}\pi\right)\right]\\&=\cos^2\left(\frac{\pi}{2}+\frac{q}{2\times N}\pi\right)\\&=\cos^2\left(\frac{q+N}{2\times N}\pi\right)\\&=(C_{2\times N}^{q+N})^2\end{aligned}$$

这样 $C_{2\times N}^{2p}=(C_{2\times N}^{p})^2-(C_{2\times N}^{p+N})^2$.

类似的,$C_{2\times N}^{2q}=(C_{2\times N}^{q})^2-(C_{2\times N}^{q+N})^2$. 因此就有了第 $s+1$ 阶段中权重因子的性质:

$$C_{2\times N}^{2m}=(C_{2\times N}^{m})^2-(C_{2\times N}^{m+N})^2,\ (m=p,q).$$

由于 $C_{2\times N}^{p}$, $C_{2\times N}^{q}$, $C_{2\times N}^{p+N}$和 $C_{2\times N}^{q+N}$是计算第 s 阶段内蝴蝶运算单元所需加载的四个权重因子,而第 $s+1$ 阶段内四个重叠的蝴蝶运算单元所需加载的权重因子又可以经过这四个权重因子推导出,所以计算这 8 个蝴蝶运算单元只需调用这 4 个权重因子.

根据这个属性,该方法可将每相邻两阶段内的蝴蝶运算单元进行融合,如图 2(b)所示. 这样,$N\times N$ 点矢量基二维 DCT 的运算流程图的蝴蝶计算部分便可以划分为($\lg N$)/2 个阶段. 当 $\lg N$ 是奇数时,虽然所有蝴蝶运算单元可以融合到($\lg N-1$)/2 个阶段内后由新方法来计算,但是仍需用常规方法[11]来计算第 $\lg N$ 阶段(即第奇数阶段)的蝴蝶运算单元. 根据本文的方法,图 3 重新绘制出原图 1 中的蝴蝶计算部分.

假设图 1 和 3 中需计算 16 个二维 DCT 系数,如图所示,图 1 的常规矢量基二维 DCT 需调用 5 个权重因子,而图 3 中则仅需调用 4 个权重因子. 另外,图中的圆点表示由信号输入 $x[\,]$产生的内存存取. 两图对比明显:图 1 的蝴蝶计算部分中,共有 64 个圆点(包括黑点和白点,下同),而图 3 的蝴蝶计算部分中,只有 32 个圆点.

如只计算部分 DCT 系数,程序中就可以省略两图中虚线所对应的运算. 在图 1 中,当 $N_0=2$,也即只需计算 2 个 DCT 系数时,蝴蝶计算部分中因输入点 $x[\,]$产生的内存存取数量为 34,而当 $N_0=16$ 时该数据增加为 64;在图 3 中,当 $N_0=2$ 时,因 $x[\,]$产生的内存存取数量为 18,而当 $N_0=16$ 时该数据增加为 32.

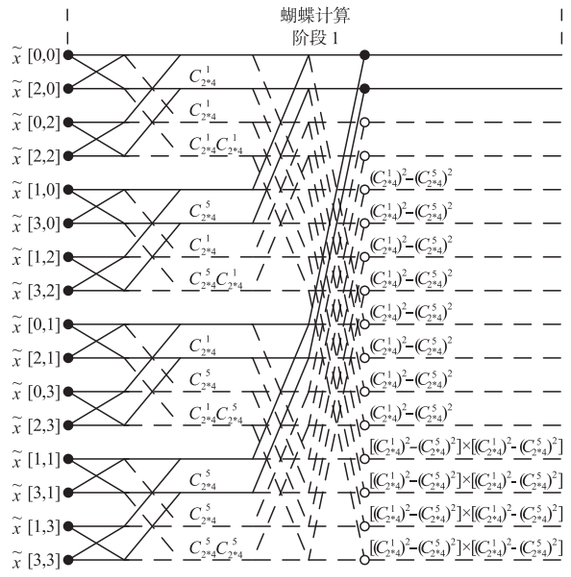


图3 使用内存存取减少方法后的4×4点矢量基二维DCT修剪中的蝴蝶计算部分

4 实验结果

用常规方法[11]实现 $N\times N$ 点矢量基二维 DCT 修剪时,由权重因子而产生的内存存取量是(N^2-1)/3. 本文的方法将它减少到$\lceil 4(N^2-1)/15\rceil$. 另一方面,考虑到裁剪值 N_0独立于输入的 $N\times N$ 点,因而导致裁剪后的计算流程图(即图 1 和 3 中实线部分对应的图)没有理论的规则性,致使因输入点而产生的内存存取只能由图 1 和 3 中的黑色原点处进行对比. 从图中可明显看出,图 3 中的黑色原点显着少于图 1 的黑色原点数量. 最后实验中测得的内存存取数量是由权重因子产生的内存存取量和由输入点而产生的内存存取量的总和.

在运算复杂度上,虽然本文的方法增加了一定的运算(如在计算权重因子的过程中因为额外的 CPU 加法和乘法而增加了一定的 CPU 时钟周期数),但是内存存取操作要比 CPU 的加法和乘法更耗时. 实验数据也表明,这种对内存存取的减少所带来 CPU 时钟周期的减少还是大大超过了额外增加的那一部分.

为进行全面而准确的实验,本文选用了三款主流 DSP 作为实验平台:德州仪器的 TMSC320C64x、模拟器件(ADI)的 ADSP-TS101、以及飞思卡尔(freescale)的

SC3400. 在这些平台上,本文编写了两段 C 程序来进行比对. 程序 1 是用常规的算法[11]实现 $N\times N$ 点矢量基二维 DCT 修剪,而程序 2 是用内存存取减少方法来实现. 两个程序均包含了相同的后续操作部分代码. 由于篇幅限制,本文针对每一个 DSP 只能用三个表,即总共 9 个表来报告当修剪值 N_0 为 2 的整数幂时的数据:表 1 列出在 TMSC320C64x 上 CPU 时钟周期数的对比;表 2 列出在 TMSC320C64x 上的由权重因子和信号输入导致的内存存取量的对比;表 3 给在 TMSC320C64x 上用于存储权重因子的内存空间的对比;表 4、5、6 则分别给出在 TS101 上对应的对比数据;而表 7、8、9 则分别给出在 SC3400 上对应的对比数据.

表 1　TMSC**320C64**x 上时钟周期数对比

$N\times N$		$N_0=2$	4	8	16	32	64
4×4	程序 1	92	167	278	319		
	程序 2	61	101	123	197		
8×8	程序 1	977	1453	1632	1933	1910	2122
	程序 2	122	127	251	277	331	357
16×16	程序 1	4901	5277	6671	7012	7301	8412
	程序 2	2431	2621	2811	3712	3231	3381
32×32	程序 1	14213	21543	26991	30012	32775	33021
	程序 2	7832	11912	12323	12501	13012	13997
64×64	程序 1	74002	86982	100032	130202	141017	140974
	程序 2	37992	40031	43995	56342	57972	58076
$N\times N$		$N_0=128$	256	512	1024	2048	4096
16×16	程序 1	7921	8302				
	程序 2	3011	3415				
32×32	程序 1	33831	34032	35181	36932		
	程序 2	13991	14315	14921	15032		
64×64	程序 1	137233	149321	150133	159926	162576	169031
	程序 2	52113	56994	59931	60037	62099	64012

表 2　TMSC**320C64**x 上内存存取数量对比

$N\times N$		$N_0=2$	4	8	16	32	64
4×4	程序 1	40	58	62	70		
	程序 2	22	24	28	36		
8×8	程序 1	168	250	286	358	374	406
	程序 2	100	118	154	226	242	274
16×16	程序 1	680	1018	1182	1510	1654	1942
	程序 2	390	456	588	852	868	900
32×32	程序 1	2728	4090	4766	6118	6774	8086
	程序 2	1572	1846	2394	3490	3634	3922
64×64	程序 1	10920	16378	19102	24550	27254	32662
	程序 2	6278	7368	9548	13908	14436	15492
$N\times N$		$N_0=128$	256	512	1024	2048	4096
16×16	程序 1	2006	2134				
	程序 2	964	1092				
32×32	程序 1	8662	9814	10070	10582		
	程序 2	4498	5650	5906	6418		
64×64	程序 1	35286	40534	42838	47446	48470	50518
	程序 2	17604	21828	22084	22596	23620	25668

表 3　TMSC**320C64**x 上存储权重因子所需内存空间(字节)对比

$N\times N$	4×4	8×8	16×16	32×32	64×64
程序 1	10	42	170	682	2730
程序 2	8	34	136	546	2184

表 4　ADSP-TS**101** 上时钟周期数对比

$N\times N$		$N_0=2$	4	8	16	32	64
4×4	程序 1	109	198	243	289		
	程序 2	73	121	134	211		
8×8	程序 1	1003	1289	1727	1863	1814	1921
	程序 2	110	139	242	249	326	315
16×16	程序 1	4809	5389	6543	6989	7258	8724
	程序 2	2518	2824	2980	3576	3124	2997
32×32	程序 1	15421	22675	25699	28992	31766	32765
	程序 2	7965	12006	12312	12475	12966	11723
64×64	程序 1	73443	85077	99108	121426	134211	129873
	程序 2	36798	41993	45698	57628	50132	50764
$N\times N$		$N_0=128$	256	512	1024	2048	4096
16×16	程序 1	8142	8257				
	程序 2	3212	3521				
32×32	程序 1	35027	33265	34891	36538		
	程序 2	14347	15012	14578	14067		
64×64	程序 1	136952	146972	148928	160127	163532	163267
	程序 2	55138	58931	59633	60431	61539	62015

表 5　ADSP-TS**101** 上内存存取数量对比

$N\times N$		$N_0=2$	4	8	16	32	64
4×4	程序 1	40	58	62	70		
	程序 2	22	24	28	36		
8×8	程序 1	168	250	286	358	374	406
	程序 2	100	118	154	226	242	274
16×16	程序 1	680	1018	1182	1510	1654	1942
	程序 2	390	456	588	852	868	900
32×32	程序 1	2728	4090	4766	6118	6774	8086
	程序 2	1572	1846	2394	3490	3634	3922
64×64	程序 1	10920	16378	19102	24550	27254	32662
	程序 2	6278	7368	9548	13908	14436	15492
$N\times N$		$N_0=128$	256	512	1024	2048	4096
16×16	程序 1	2006	2134				
	程序 2	964	1092				
32×32	程序 1	8662	9814	10070	10582		
	程序 2	4498	5650	5906	6418		
64×64	程序 1	35286	40534	42838	47446	48470	50518
	程序 2	17604	21828	22084	22596	23620	25668

表 6　ADSP-TS**101** 上存储权重因子所需内存空间(字节)对比

$N\times N$	4×4	8×8	16×16	32×32	64×64
程序 1	10	42	170	682	2730
程序 2	8	34	136	546	2184

根据表 1 ~ 表 9 得知:在 TMSC320C64x 上,本文方法平均减少了 59.4% 的 CPU 时钟周期数;在 ADSP-TS101 上,它平均减少了 56.3% 的 CPU 时钟周期数;而在 SC3400 上,它平均减少了 58% 的 CPU 时钟周期数. 综

合这三个平台,本文方法实现了平均减少 57.9% 的 CPU 时钟周期数.而且在这三个平台上,本文的方法都实现了平均减少 47.2% 的由权重因子和信号输入导致的内存存取量、以及平均减少 20% 的存储权重因子的内存空间.

表 7 SC3400 上时钟周期数对比

$N\times N$		$N_0=2$	4	8	16	32	64
4×4	程序 1	101	211	259	303		
	程序 2	69	110	151	199		
8×8	程序 1	997	1216	1811	1936	1991	2011
	程序 2	103	156	251	269	377	391
16×16	程序 1	4901	5402	6671	6891	7190	9012
	程序 2	2661	2792	2911	3492	3502	3599
32×32	程序 1	14121	21772	24983	28021	30983	32662
	程序 2	7792	10091	11994	12952	13021	14009
64×64	程序 1	72901	86799	98172	120087	130286	132775
	程序 2	36291	40992	44921	56212	59327	60342
$N\times N$		$N_0=128$	256	512	1024	2048	4096
16×16	程序 1	7917	8411				
	程序 2	2992	3613				
32×32	程序 1	34497	34958	35032	36028		
	程序 2	13974	14973	15011	16032		
64×64	程序 1	134873	145729	149291	159317	162169	164829
	程序 2	54197	57993	58992	59917	60027	63012

表 8 SC3400 上内存存取数量对比

$N\times N$		$N_0=2$	4	8	16	32	64
4×4	程序 1	40	58	62	70		
	程序 2	22	24	28	36		
8×8	程序 1	168	250	286	358	374	406
	程序 2	100	118	154	226	242	274
16×16	程序 1	680	1018	1182	1510	1654	1942
	程序 2	390	456	588	852	868	900
32×32	程序 1	2728	4090	4766	6118	6774	8086
	程序 2	1572	1846	2394	3490	3634	3922
64×64	程序 1	10920	16378	19102	24550	27254	32662
	程序 2	6278	7368	9548	13908	14436	15492
$N\times N$		$N_0=128$	256	512	1024	2048	4096
16×16	程序 1	2006	2134				
	程序 2	964	1092				
32×32	程序 1	8662	9814	10070	10582		
	程序 2	4498	5650	5906	6418		
64×64	程序 1	35286	40534	42838	47446	48470	50518
	程序 2	17604	21828	22084	22596	23620	25668

表 9 SC3400 上存储权重因子所需内存空间(字节)对比

$N\times N$	4×4	8×8	16×16	32×32	64×64
程序 1	10	42	170	682	2730
程序 2	8	34	136	546	2184

5 结论

本文提出了实现矢量基二维 DCT 算法修剪的内存存取减少方法.它利用权重因子的属性将计算流程图内每相邻两阶段内的蝴蝶运算单元进行融合,然后再以较少的权重因子来计算.实验数据显示:该方法相比于常规算法[1]可以平均减少 57.9% 的 CPU 时钟周期数、平均减少 47.2% 的内存存取量、以及平均减少 20% 的内存空间.该方法能有效节省电能消耗,所以它对将来二维 DCT 在手持及移动设备上的应用具有重要的意义.我们下一步不仅要继续研发减少内存存取、提高速度的优化方案,而且要将该方案应用到其他类型 DCT 上,以使其可以得到更广泛的应用.

参考文献

[1] N AHMED, T NATARAJAN, K R RAO. Discrete Cosine-Transform[J]. IEEE Transactions on Computers, 1974, C-23(1):90 – 93.

[2] K RRao, P Yip. Discrete Cosine Transform: Algorithms, Advantages, Applications[M]. New York: Academic, 1990.

[3] 龚若皓,杨斌. 基于移动多核 GPU 的并行二维 DCT 变换实现方法[J]. 成都信息工程学院学报,2015,30(1):22 – 26.
GONG Ruo-hao, YANG Bin. Parallelization of 2D-DCT based on Mobile Multicore GPU[J]. Journal of Chengdu University of Information Technology, 2015, 30(1): 22 – 26. (in Chinese)

[4] 陈睿,王晶,黄华军,M R Alsharif. 基于分块 DCT 变换的多聚焦图像融合[J]. 小型微型计算机系统,2016,37(2):321 – 326.
CHEN Rui, WANG Jing, HUANG Hua-jun, M R Alsharif. Multi-focus image fusion based on block DCT transform [J]. Journal of Chinese Computer Systems, 2016, 37(2): 321 – 326. (in Chinese)

[5] 桑爱军,崔新宇,王艇,李晓妮. 2M 维矢量矩阵 DCT 整数变换及并行实现[J]. 东北大学学报(自然科学版),2017,38(11):1543 – 1547.
SANG Ai-jun, CHUI Xin-yu, WANG Ting, LI Xiao-ni. 2M-dimensional vector matrix DCT integer transform and parallel implementation[J]. Journal of Northeastern University (Natural Science), 2017, 38(11): 1543 – 1547. (in Chinese)

[6] Giovanni Renda, Maurizio Masera, Maurizio Martina, Guido Masera. Approximate arai DCT architecture for HEVC [A]. 2017 New Generation of CAS (NGCAS)[C]. Genova, Genoa: IEEE, 2017. 133 – 135.

[7] 贾昆霖. 基于 8×8 的整数 DCT 快速计算的 H.264/AVC 软件实现[J]. 电子科技,2017,30(6):43 – 45.
JIA Kun-Lin. Realization of H.264/AVC software based on 8 × 8 integer DCT fast calculation[J]. Electronic Science and Technology, 2017, 30(6): 43 – 45. (in Chinese)

[8] Diego F Coelho, Sushmabhargavi Nimmalapalli, Vassil

Dimitrov, Arjuna Madanayake, Renato J Cintra, Arnaud Tisserand. Computation of 2D 8 × 8 DCT based on the loeffler factorization using algebraic integer encoding[J]. IEEE Transactions on Computers, 2018, (67) 12: 1692 – 1702.

[9] S C Chan, K L Ho. A new two dimensional fast cosine transformalgorithm[J]. IEEE transactions on signal processing, 1991, 39(2): 481 – 485.

[10] E Feig, S Winograd. On the multiplicative complexity of discrete cosine transform[J]. IEEE Transactions on Information Theory, 1992, 38(4): 1387 – 1391.

[11] C A Christopoulos, J Bormans, J Cornelis, A N Skodras. The vector-radix fast cosine transform: Pruning and complexity analysis[J]. Signal Processing, 1995, 43(2): 197 – 205.

[12] Muye Norley Liu. Vector-radix DCT/ICT implementation for MPEG DSP[A]. Proceedings of the Third International Conference on Signal Processing[C]. Beijing, China: IEEE, 1996. 641 – 644.

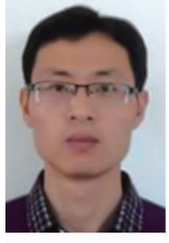
[13] X Liu, HBao. Efficient implementation of 2-D FCT with reduced memory access for programmable DSPs[J]. Journal of Signal Processing Systems, 2013, 80(2): 153 – 161.

[14] A N Skodras, A G Constantinides. Efficient input-reordering algorithms for fast DCT[J]. Electronics Letters, 1991, 27(21): 1973 – 1975.

作者简介

刘项洋　男,1979 年 6 月生于安徽省芜湖市,博士,现为安徽师范大学计算机与信息学院讲师,主要研究方向为数字信号处理算法,嵌入式计算.
E-mail:33736326@qq.com

许　勇　男,1966 年 12 月生于安徽省六安市. 博士,安徽师范大学计算机与信息学院教授,研究方向为计算机网络和嵌入式计算.
E-mail:yxull@mail.ahnu.edu.cn



郑孝遥　男,1981 年生,安徽芜湖县人,博士在读,安徽师范大学计算机与信息学院副教授,研究方向为信息安全、社区网络、嵌入式计算.
E-mail:zxiaoyao@ahnu.edu.cn

陈付龙　男,1978 年生,安徽霍邱县人,博士,安徽师范大学计算机与信息学院教授,研究方向为高性能计算机体系结构、物联网安全、嵌入式与普适计算.
E-mail:long005@ahnu.edu.cn