快速 DCT 修剪在 DSP 上的内存访问优化方法

刘项洋,许 勇
(安徽师范大学数学计算机科学学院,安徽芜湖 241000)

摘 要: 在本论文中,我们提出一个新的内存访问优化方法以减少由权重因子(在 DCT 的快速修剪计算图中的余弦系数)和输入点而产生的内存访问量,实现在 DSP 上的快速 DCT 修剪. 该方法通过两个步骤来减少内存访问量:1. 减少权重因子的个数;2. 将快速 DCT 修剪的计算流程图中两个阶段中的蝴蝶运算单元合并到一个阶段中,从而形成一个高效的蝴蝶运算单元. 我们在 TI TMSC320C64x DSP 上应用该方法来实现修剪 FCT. 实验结果表明,与传统的实现方法相比,修剪 FCT 方法在 DSP 上可以平均减少 40% 的内存访问量,平均减少 48.6% 的时钟周期和平均节约 32.6% 的由存储加权因子导致的内存访问.

关键词: 数字信号处理器 (DSP); 离散余弦变换 (DCT); 内存访问
中图分类号: TN911.23 **文献标识码:** A **文章编号:** 0372-2112 (2016)01-0227-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.01.034

Memory Access Optimization Method for the Implementation of Fast DCT Pruning on DSP

LIU Xiang-yang, XU Yong
(*School of Math and Computer Science, Anhui Normal University, Wuhu, Anhui* 241000, *China*)

Abstract: In this paper, we propose a memory access optimization method to minimize the memory accesses due to weighting factors (cosine coefficients in the computation diagram of fast DCT pruning) and input points for implementing fast DCT pruning on DSP. The proposed method reduces the number of memory accesses in two steps: 1. Reduce the number of weighting factors; 2. Combine butterflies at two stages in fast DCT pruning diagram to form an efficient butterfly structure in one stage and calculate them. The proposed method is applied to implement Pruning FCT on TI TMSC320C64x DSP. Experimental results show that the proposed method can achieve an average of 40% memory access reduction, 48.6% clock cycle reduction and 32.6% of memory space saving for weighting factors to compute Pruning FCT on DSP comparing with the conventional implementation.

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

1 引言

离散余弦变换(Discrete Cosine Transform, DCT)从 1974 年被文献[1]定义以来,在许多图像、语音编码应用程序中发挥了非常重要的作用,而其中最常用的是二型 DCT (DCT-II). 文献[2]给出了二型 DCT 的具体定义:

$$X[m] = \sqrt{N/2}k_m \sum_0^{N-1} x(n)\cos[(2n+1)m\pi/2N]$$
$$m = 0,1,\cdots,N-1 \quad (1)$$

N 是 2 的整数幂,$m=0$ 时 $k_m = 1/\sqrt{2}$ 否则 $k_m = 1$.

直接用硬件或软件来实现这个公式是非常低效的. 为解决这个问题,文献[2]中也提出了各种直接或间接的计算的方法. 此后,DCT 算法的研究进展包括了 radix -2^n DCT[3],量化二维 DCT[4]、多维矢量 DCT 正交矩阵变换[5] 以及 FPGA 实现 DCT[6,7] 等,均是直接计算 DCT.

通常这些算法都假定 DCT 系数的输入与输出数量相同. 然而在大多数图像处理应用程序中只有低频 DCT 系数部分才保存最有用的信号信息,所以实际上中只需要计算这部分 DCT 系数就可以了. 这种只计算部分 DCT 系数的

收稿日期:2014-05-19;修回日期:2015-07-06;责任编辑:李勇锋

方法通常被称为 DCT 修剪或修剪 DCT[10]. 由于只计算一部分而不是完整的 DCT 系数,DCT 修剪可以减少一些 CPU 的运算操作和一定数量的内存访问. 因此 DCT 修剪一般比传统 DCT 速度更快,效率更高. 目前也有许多算法已用于计算 DCT 修剪[8,9]等. Wang[10]在这些论文中首先提出有效的快速 DCT 修剪算法(Fast DCT Prunning). 后来又有学者提出修剪 FCT(Prunning FCT)[11],并且通过与前者比较计算复杂度而证明文献[10]更快.

尽管 DCT 修剪比传统 DCT 和许多已经出现在文献上的算法快,但是到目前为止,大多数快速 DCT 修剪算法并没有具体的软件实现,而且仅有的一些软件实现也都是基于通用处理器的. 基于通用处理器的软件实现通常远比基于 DSP 的实现速度慢. DSP 是一类特殊的经过优化以用来处理各种信号处理应用程序的处理器,如 FIR 滤波器、傅里叶变换 FFT、DCT 以及他们各自的修剪算法. 在工业上,DSP 作为各种信号处理应用程序的平台,发挥着重要的作用. 德州仪器(TI),模拟设备公司(ADI)和飞思卡尔等 DSP 制造商已经提供了很多不同的 DSP 芯片及其评估板材和仿真工具,供大学研究和工业使用. 由于其性能的优异性、灵活性以及合适的成本,基于 DSP 的软件实现已经成为优良的商业产品解决方案.

然而在实际当中,基于 DSP 的快速而有效的 FCT 修剪实现仍然很困难. 众所周知,DSP 系统的内存访问会导致较长的时钟延迟和大量的功耗,因此运算成本非常高. 如在 TI TMS320C64x DSP 中,执行一个内存装载指令会花费四个时钟周期. 因此频繁内存访问会大幅增加时钟周期数. 尽管 DCT 修剪与快速 DCT 算法相比能减少一定数量的内存访问,但却没有根本解决问题. 诸如修剪 FCT[11]中还是存在大量由输入点和权重因子(余弦系数)而导致的内存访问数量,这也在很大程度上影响了速度的提高.

本文提出了一个新的减少内存访问的修剪 FCT 算法. 首先利用修剪 FCT 中权重因子的性质来减少运算所需的权重因子数. 然后将运算流程图内两个阶段中的蝴蝶计算结构合并到一个阶段,从而形成一个高效的蝴蝶计算结构来进行计算. 实验结果表明,该方法可以节约用于保存权重因子的内存空间,并且也能显著减少运算所需的内存访问以及时钟周期的数量.

2 快速离散余弦变换的修剪算法

修剪 FCT 回顾:本节首先简要介绍 FCT(快速离散余弦变换)的基本思想. 然后简要介绍基于 FCT 的修剪 FCT.

2.1 直接 FCT 计算

式(1)给出了最常用的离散余弦变换:II-型 DCT.

为简便起见,我们将公式右边的比例系数一起整合到 $X[m]$ 中,并应用于文献[12]中提到的输入映射:

$$y(n)=x(2n)\text{和} \quad (2)$$
$$y(N-n-1)=x(2n+1),n=0,1,\cdots,N/2-1$$

然后式(1)就改写为

$$X[m]=\sum_0^{N-1}y(n)\cos[\pi(4n+1)m/2N]$$
$$m=0,1,\cdots,N-1 \quad (3)$$

接着通过使用三角函数的性质(4)以及将式(3)中的输出序列 $X[M]$ 分解为奇数与偶数索引部分.

$$\cos[(2k+1)\Phi]=2\cos\Phi\cos(2\Phi)-\cos[(2k-1)\Phi] \quad (4)$$

于是便有了两个 $N/2$ 点 DCT[12]. 然后进一步分解直到一系列两点输入 DCT,便得到了按频率分解的 FCT[11]. 图 1 列出了 FCT 运算流程图. (注:目前在流程图中虚线和实线是相同作用的). 它由两部分组成:蝴蝶计算部分和后序操作部分. 在蝴蝶计算部分除了输入点和系数($2c$)不同以外,算法流程与 FFT 类似. 如图所示,蝴蝶计算部分与后序操作部分相比,产生了大部分内存访问和 CPU 计算. 本文的内容将专注于蝴蝶计算部分.

2.2 修剪 FCT

DCT 在具备快速算法的一类转换运算中具备非常优越的能量压缩属性. 正是这种可以仅仅计算部分系数的属性,使得 DCT 在各种图形信号处理应用程序中最适合处理有损数据压缩转换. 通常,25% 的 DCT 系数已经能够很好的压缩和恢复图像,而不会造成视觉上的差异[2]. 如在图 1 的 N 个 DCT 系数中,如仅计算前 N_0 个低频组件(其中 $N_0=3,N=16,N_0$ 不一定是 2 的整数幂),那么就不用处理图中虚线对应的所有运算,因此修剪 FCT 相比一般 FCT 能节省一些操作(包括 CPU 运算和内存访问操作). 不过实际运算中,它还是存在很多由权重因子和输入点 $X[\,]$ 而导致的内存访问.

3 修剪 FCT 的内存访问优化方法

为了解决上述提到的问题,我们提出了内存访问优化方法,该方法通过两个步骤来应用于修剪 FCT.

步骤 1:利用权重因子的余弦函数属性(5),将实现 N 点修剪 FCT 的权重因子数量由 $N-1$ 减少到$\lfloor(2N-1)/3\rfloor$.

$$C_{2N}^{2m}=(C_{2N}^{m})^2-(C_{2N}^{m+N/2})^2,m\in(0,N/2)$$
$$\text{其中 } C_{2N}^{m}=\cos(m\pi/2N),m=1,2,\cdots N/2-1 \quad (5)$$

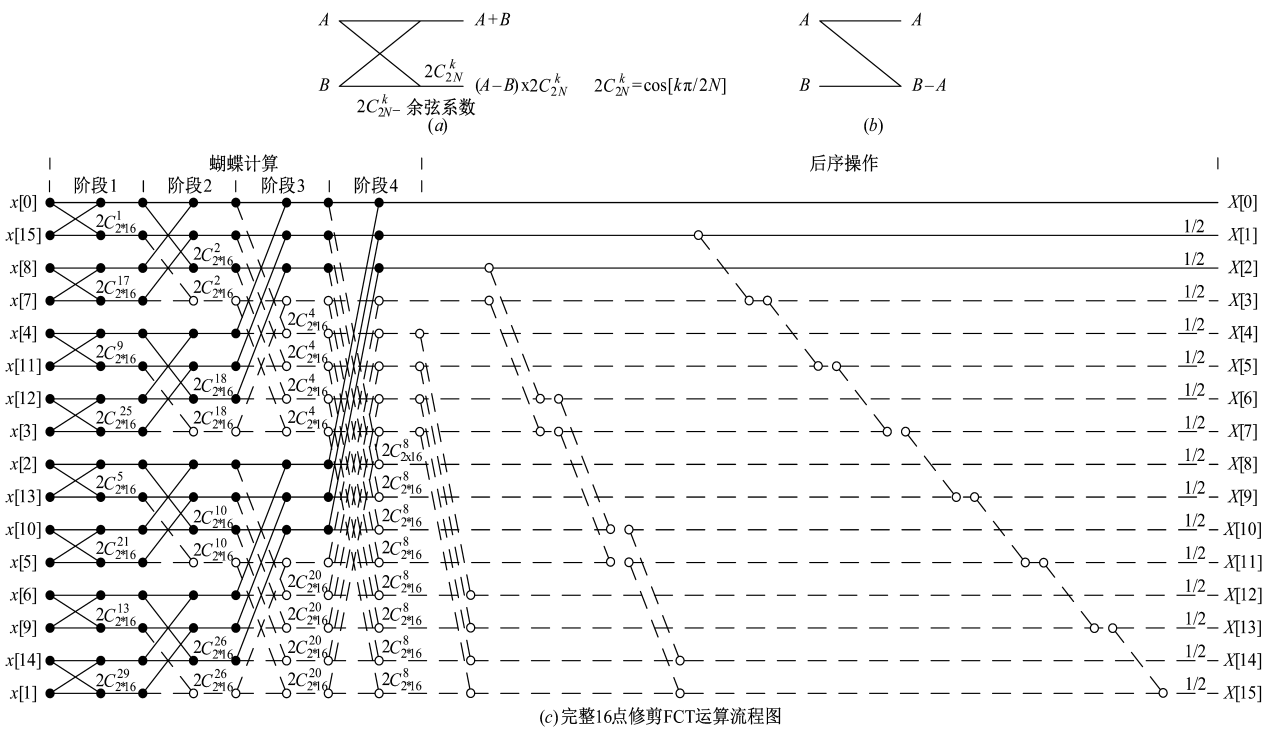

图1　16点修剪FCT (*a*)蝴蝶计算部分中的基本蝴蝶运算单元 (*b*)后序操作部分中的基本运算单元

例如在图 1 中的阶段 3 和 4 中的蝴蝶运算单元需要分别与权重因子:$2C_{2\times16}^{4}$,$2C_{2\times16}^{20}$和 $2C_{2\times16}^{8}$结合计算. 其中,$2C_{2\times16}^{8}$可用$[(2C_{2\times16}^{4})^2-(2C_{2\times16}^{20})^2]/2$ 取代,推导过程如下:

$$
\begin{aligned}
&[(2C_{2\times16}^{4})^2-(2C_{2\times16}^{20})^2]/2\\
&=2\left[\left(\cos\frac{4\pi}{2\times16}\right)^2-\left(\cos\frac{20\pi}{2\times16}\right)^2\right]\\
&=2\left\{\left(\cos\frac{4\pi}{2\times16}\right)^2-\left[-\cos\left(\pi-\frac{20\pi}{2\times16}\right)\right]^2\right\}\\
&=2\left[\left(\cos\frac{4\pi}{2\times16}\right)^2-\left(\cos\frac{12\pi}{2\times16}\right)^2\right]\\
&=2\left\{\left(\cos\frac{4\pi}{2\times16}\right)^2-\left[\sin\left(\frac{\pi}{2}-\frac{12\pi}{2\times16}\right)\right]^2\right\}\\
&=2\left[\left(\cos\frac{4\pi}{2\times16}\right)^2-\left(\sin\frac{4\pi}{2\times16}\right)^2\right]\\
&=2\cos\frac{8\pi}{2\times16}\\
&=2C_{2\times16}^{8}
\end{aligned}
$$

因此,阶段 3 和 4 中的所有的蝴蝶运算单元只需结合权重因子 $2C_{2\times16}^{4}$和 $2C_{2\times16}^{20}$就可以计算. 同样的,权重因子$2C_{2\times16}^{2}$和$2C_{2\times16}^{18}$可分别由$[(2C_{2\times16}^{1})^2-(2C_{2\times16}^{17})^2]/2$ 和$[(2C_{2\times16}^{9})^2-(2C_{2\times16}^{25})^2]/2$ 所取代. 这种方法同样适用于阶段 1 和 2 中的其他剩余权重因子. 因此只需用阶段 1 和 3 的权重因子就可以处理全部蝴蝶运算单元了.

步骤 2:在运用上述特性后,将相邻两个阶段中的全部蝴蝶运算单元合并到一个阶段中进行运算.

在修剪 FCT 计算流程图中蝴蝶计算部分的阶段 s 中有 $N/2^s$个不同的权重因子,它们按照如下方式进行排列:阶段 s ($s=0,1,2,\cdots\cdots,\log_2N-1$)的第一个权重因子是:$2C_{2N}^{k}$(其中 $k=2^s$和 $C_{2N}^{k}=\cos(k\pi/2N)$);阶段 s 的第二个权重因子由第一个权重因子中的 k 加上 N 得到,即 $2C_{2N}^{k+N}$. 第三和第四个权重因子是由前两个因子分别在他们的参数 k 上加 $N/2$ 得到. 在同一阶段的其余权重因子都是通过这个递归过程而产生. 因为在运行修剪 FCT 算法前对输入点进行了重排序[12],所以在每个阶段里的所有的蝴蝶运算单元都可以按照自然数输入序列的常规方式来从上向下逐一进行计算. 图 2 (*a*)描绘出一般情况下在阶段 s 和 $s+1$ 中相邻的 4 个蝴蝶运算单元之间的计算关系.

定理　如图 2(*a*)所示,修剪 FCT 的蝴蝶计算部分阶段 s 和 $s+1$($s\leqslant\log_2N-1$)中的四个蝴蝶运算单元只需通过加载如图 2(*b*)所示的两个权重因子 $2C_{2N}^{m}$和 $2C_{2N}^{m+N/2}$来计算.

证明　根据图 2(*a*),阶段 $s+1$ 的蝴蝶运算单元需要结合权重因子 $2C_{2N}^{2m}$计算,而 $2C_{2N}^{2m}=2\cos(2m\pi/2N)$,根据三角函数属性:$\cos2A=\cos^2A-\sin^2A$,权重因子 $2\cos(2m\pi/2N)$可改写成 $2(\cos(m\pi/2N))^2-2[\sin(m\pi/2N)]^2$,第一项 $2(\cos(m\pi/2N))^2=2(C_{2N}^{m})^2=(2C_{2N}^{m})^2/2$. 而第二项

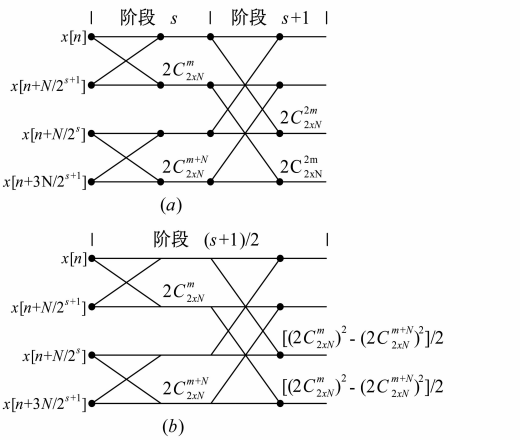

图2 用两个权重因子计算4个蝴蝶运算单元

$2[\sin(m\pi/2N)]^2$ 可变为:$2\{\cos[(\pi/2)-(m\pi/2N)]\}^2 = 2\{\cos[(\pi/2)+(m\pi/2N)]\}^2 = 2\{\cos[(m+N)\pi/2N)]\}^2 = \{2\cos[(m+N)\pi/2N)]\}^2/2 = (2C_{2N}^{m+N})^2/2$,这样就可以得到阶段 $s+1$ 的权重因子:$2C_{2N}^{2m} = (2C_{2N}^{m})^2/2 - (2C_{2N}^{m+N})^2/2$.

因此,阶段 $s+1$ 内的权重因子可以由阶段 s 内的两个权重因子推导出,所以只需加载这两个权重因子便可以一起计算这 4 个蝴蝶运算单元.

在减少权重因子后,我们将每两阶段中的蝴蝶运算单元合并在一个阶段中以形成一个如图 2(b)所示的高效蝴蝶运算单元,所以当 $\log_2 N$ 是奇数时,该方法要将阶段 1 到阶段 $\log_2 N-1$ 内的所有蝴蝶运算单元合并到$(\log_2 N-1)/2$ 个阶段内来计算,而在阶段 $\log_2 N$ 最后奇数阶段)中的所有蝴蝶运算单元还需用传统的修剪 FCT 方法来计算.

利用完该方法后,图 1 中的蝴蝶计算部分便可重新被画为图 3 所示. 假设图 1 和 3 均需要计算 16 个 DCT 系数,即图中的实线和虚线没有区别,那么图 3 需要加载 10 个权重因子,而图 1 的传统修剪 FCT 却需加载 15 个权重因子. 此外在图 3 中由输入点 $X[\ ]$ 产生的内存访问只发生在图中黑色和白色圆圈,而图 1 里的由输入点导致的内存访问却发生在所有蝴蝶运算单元两边所有圆圈内.

如果仅需计算部分 DCT 系数,则可以省略两图中的虚线部分,也就省去了相应的运算操作,并减少了图中由黑圈所标识的内存访问量. 如图 1 所示,在 16 点修剪 FCT 中,当 $N_0=3$,也即只计算 3 个 DCT 系数时,蝴蝶计算部分中因输入点产生的内存访问量为 87,而当 $N_0=16$ 时它增加为 128,相比之下在图 3 中,当 $N_0=3$,因输入点产生的内存访问量为仅为 43,而当 $N_0=16$ 时它增加为 64.

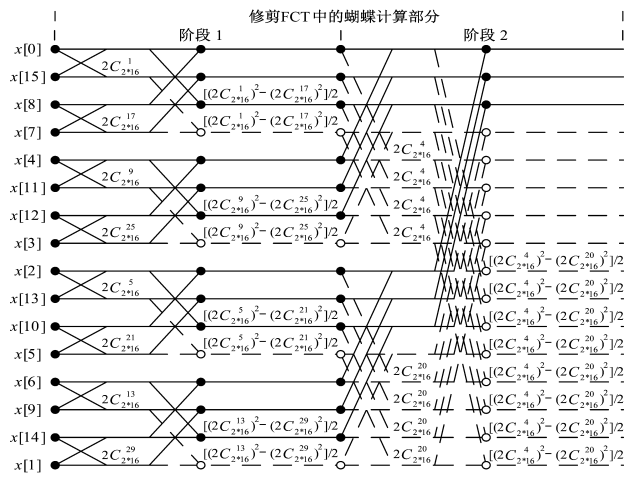

图3 使用内存访问优化方法后的16点修剪FCT中的蝴蝶计算部分

4 内存访问优化方法的性能评价

假设 N 点输入中计算 N_0 个 DCT 系数,同时查看 N_0 是 2 的整数幂的特殊情况,在传统修剪 FCT 的蝴蝶计算部分中,由输入点产生的内存访问量为$(2\log_2 N_0+3)N-3N_0$,采用本文的方法后,内存访问量减少到如下表达式:

$(\log_2 N_0+4/3)N-N_0/3$,$\log_2 N_0$和 $\log_2 N$ 均为奇数

$(\log_2 N_0+5/3)N-N_0/3$,$\log_2 N_0$是偶数而 $\log_2 N$ 均为奇数

$(\log_2 N_0+4/3)N-5N_0/3$,$\log_2 N_0$是奇数而 $\log_2 N$ 是偶数

$(\log_2 N_0+5/3)N-5N_0/3$,$\log_2 N_0$和 $\log_2 N$ 均为偶数

并且该方法可以使 N 点修剪 FCT 中由权重因子产生的内存访问量从 $N-1$ 它减少到$\lfloor(2N-1)/3\rfloor$.

为了能方便全面推广,我们选取了一款比较通用的处理器:TI TMSC320C64x DSP 为测试平台. 实验中编写了两段 C 代码来对比:代码 C1 用传统算法实现修剪 FCT,代码 C2 则基于内存访问优化方法实现修剪 FCT. 两段代码都包括一段相同的基于[11]的后序操作部分. 测试环境采用德州仪器设计的软件开发工具 Code Composer Studio (CCS). 由于篇幅限制,我们只能用如下三个表来报告当修剪值 N_0为 2 的整数幂时的对比数据:表 1 列出时钟周期数;表 2 列出由权重因子和输入点产生的内存访问量;表 3 列出存储权重因子所需内存量.

结果表明,应用该方法的修剪 FCT 相比于传统方式能减少平均 40% 的由权重因子和输入点产生的内存访问量,减少平均 48.6% 的时钟周期数和节约平均 32.6% 的内存空间. 目前公认 25% 的 DCT 系数就能很好地重建图像[2],而当修剪值 N_0为 N 的 25% 时,该方法可以减少平均 44% 的内存访问数量,以及减少平均

51.3% 的时钟周期数.

5 结论

本文提出了实现修剪 FCT 算法的内存访问优化方法. 它先利用权重因子的余弦函数属性来减少计算过程中所需加载的权重因子数,然后将计算流程图中两个阶段中的蝴蝶运算单元合并到一个阶段以形成一个高效蝴蝶单元结构. 实验结果表明:相比于传统实现算法,该方法可以减少平均 40% 的内存访问数量,减少平均 48.6% 的时钟周期数和减少平均 32.6% 的用于存储权重因子的内存空间.

表 1　时钟周期数的比较

N \ N_0		$N_0=2$	4	8	16	32	64	128	256	512	1024
$N=8$	C1	638	719	789							
	C2	167	191	269							
16	C1	1150	1309	1483	1677						
	C2	771	822	890	1024						
32	C1	2099	2414	2742	3108	3607					
	C2	1386	1479	1555	1707	2055					
64	C1	3919	4546	5182	5838	6673	7768				
	C2	2359	2737	2877	3144	3510	4342				
128	C1	7483	8734	9986	11222	12700	14414	16842			
	C2	4590	5306	5538	5936	6369	7159	8918			
256	C1	14535	17034	19518	21914	24677	27609	31260	36573		
	C2	8596	9833	10273	11029	11953	13309	15205	19408		
512	C1	28563	33558	38506	43222	48554	53922	60007	67755	79281	
	C2	17520	19925	20753	22165	23661	25621	27977	31571	40642	
1024	C1	56543	66530	76406	85762	96231	106471	117424	130038	146427	171270
	C2	33164	37935	39575	42373	45155	48611	52169	58503	67947	88286

表 2　内存访问量的比较

N \ N_0		$N_0=2$	4	8	16	32	64	128	256	512	1024
$N=8$	C1	41	54	70							
	C2	23	36	52							
16	C1	89	118	150	194						
	C2	42	65	81	125						
32	C1	185	246	310	386	498					
	C2	95	140	172	248	360					
64	C1	377	502	630	770	946	1218				
	C2	188	273	321	429	541	813				
128	C1	761	1014	1270	1538	1842	2242	2882			
	C2	383	556	652	856	1032	1432	2072			
256	C1	1529	2038	2550	3074	3634	4290	5186	6658		
	C2	764	1105	1281	1645	1885	2413	3053	4525		
512	C1	3065	4086	5110	6146	7218	8386	9794	11778	15106	
	C2	1535	2220	2572	3288	3720	4632	5528	7512	10840	
1024	C1	6137	8182	10230	12290	14386	16578	19010	22018	26370	33794
	C2	3068	4433	5121	6509	7261	8813	9965	12461	15789	23213

表 3　存储权重因子所需内存空间(Byte)的比较

N	8	16	32	64	128	256	512	1024
$C1$	14	30	62	126	254	510	1022	2046
$C2$	10	20	42	84	170	340	682	1364

参考文献

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

[2] K R Rao, P Yip. Discrete Cosine Transform: Algorithms, Ad-

vantages,Applications[M]. New York:Academic. 1990.
[3] M Wezelenburg. General radix – 2n DCT and DST algori thms[A]. Proceedings of the International Conference on ECCTD'97[C]. Budapest,Hungary,1997. 789 – 794.
[4] 李永亭,齐咏生,肖志云. 基于量化的二维 DCT 优化算法研究[J]. 计算机工程与应用,2010,46(2):181 – 183.
[5] 沈鹏. 基于多维矢量 DCT 正交矩阵变换及熵编码算法的研究[D]. 吉林:吉林大学,2011.
[6] 刘斌,何剑锋,孙玲玲. 基于 FPGA 的 H. 264 DCT 算法的硬件实现[J]. 现代电子技术,2012,35(10):90 – 92.
[7] 许亚军,韩雪松,韩应征. AVS 二维 DCT 变换的 FPGA 实现[J]. 电视技术,2013,37(11):18 – 21.
[8] R Stasiliski. On pruning the discrete Cosine and Sine transforms[A]. IEEE MELECON 2004[C]. IEEE,2004. 269 – 271.
[9] Mohamed El-Sharkawy,Waleed Eshmawy. A fast recursive pruned DCT for image compression applications[A]. Proceed-ings of the 37th Midwest Symposium on Circuits and Systems[C]. Los Angeles,USA:IEEE,1994. 887 – 891.
[10] Z Wang. Pruning the fast discrete Cosine transform[J]. IEEE Trans Communications,1991,39(5):640 – 643.
[11] Athanassios N Skodras. Fast discrete Cosine transform pruning[J]. IEEE Transactions on Signal Processing,1994,42(7):1833 – 1837.
[12] 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,现为安徽师范大学数学计算机科学学院讲师,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,安徽师范大学数学计算机科学学院教授,研究方向为计算机网络和嵌入式计算.
E-mail:yxull@mail.ahnu.edu.cn