

基于超长指令字的定点 DCT 算法研究

鞠汶奇, 肖创柏, 邬 鹏

(北京工业大学 计算机学院, 北京 100022)

摘 要: 针对浮点转换为定点运算的精度问题证明了一个定理和一个推论, 给出了一种在可以同时执行多个指令操作的 DSP 中估计算法实际运行时间的方法, 并提出了一种新的适合于 VLIW 的定点 8×8 DCT 快速算法。仿真实验证明新的 8×8 DCT 算法比已有的基于 VLIW 结构的高精度算法的运算速度分别提高 13.4% 和 21.4%, 而误差方差总和分别降低了 80% 和 67.5%, 比已有的低精度算法运算速度分别提高 8.5% 和 17.2%, 而误差方差总和分别降低了 98.98% 和 98.3%。

关键词: 超长指令字; 离散余弦变换; 快速算法; 并行算法; 视频压缩; DSP

中图分类号: TP391.41

文献标识码: A

文章编号: 1673-629X(2008)01-0101-05

Research of Fixed-Point DCT Algorithm Based on VLIW Architecture

JU Wen-qi, XIAO Chuang-bai, WU Peng

(College of Computer Science and Technology, Beijing University of Technology, Beijing 100022, China)

Abstract: In this paper, one theorem and one deduction are proved, which indicate how to improve the precision of the turning floating point computation into fixed computation. A method which could estimate the efficiency of algorithms based on the DSP supporting the parallel operations and a new 8×8 DCT algorithm based on the theorem and the deduction are also presented, which includes two different computational manners. The simulations prove that computational manners can reduce the number of clock cycles required by more than 13.4% and 21.4% reduction respectively compared with the existing high precision algorithm and more than 8.5% and 17.2% reduction respectively compared with the existing low precision algorithm. The sum of the variance of error of our algorithms decrease by 80% and 67.5% compared with the existing high precision algorithm respectively and by 98.98% and 98.3% compared with the existing low precision algorithm respectively.

Key words: VLIW; DCT; fast algorithm; parallel algorithm; video compression; DSP

0 引言

DCT 已广泛用于视频压缩中, H. 263, H. 264, MPEG-1/2/4 均采用 8×8 DCT 变换。目前关于 8 点 DCT 的快速算法已有很多^[1~5], 其中较为著名的是文献[4]的 8 点 DCT 快速算法, 只用了 11 次实数乘法和 29 次实数加法。但如把这些算法的实现方法照搬到支持超长指令字(VLIW)的 DSP 中则不一定能得到最优的算法, 必须根据具体芯片的特性做相应修改。

PNX1500 DSP 是 PHILIPS1300 系列的升级换代

产品, 主要性能均有明显的提升, 已经逐步取代了 TM1300 成为主流产品。PNX1500 使用的基本命令以及指令的执行方式和 TM1300 基本相同, 文献[6, 7]对于在 VLIW 的微处理器上实现快速 8×8 DCT 算法做了初步研究, 但是文献[6]方法的计算误差较大, 而文献[7]采用的是文献[6]的低精度方法, 运算速度比文献[6]的高精度算法有所提高, 但误差方差总和却达到了文献[6]中高精度方法的几十倍。

文中对浮点转换为定点运算的精度问题提出了一个定理和一个推论, 同时, 针对文献[7]只从实验结果上给出了该文提出的快速算法的有效性, 但是并没有从理论上说明为什么算法在 PHILIPS 的允许多种指令并行的媒体处理器上能提高效率的问题, 给出了一种估计在基于 VLIW 的 DSP 中估计算法实际运行时间的方法, 并以 PNX1500 的媒体处理器为例, 提出一种新的适合于 VLIW 的 8×8 DCT 快速算法, 该算法可以衍生出两种精度的计算方式。

收稿日期: 2007-03-21

基金项目: 北京市自然科学基金(4072004); 北京市教委科技发展计划项目(KM200510005012)

作者简介: 鞠汶奇(1978-), 男, 重庆人, 硕士研究生, 研究方向为视频音频和图像处理; 肖创柏, 博士生导师, 教授, 从事数字信号处理、模式识别方向的研究工作。

注: 中国计算机学会微机(嵌入式)专委会学术会议优秀论文。

1 一种新的基于 VLIW 的 8×8 DCT 算法

PNX1500 芯片支持 VLIW, 在该系列芯片上指令执行具有一定的并行性。它的功能单元分配和 DCT 变换中特殊指令所占指令位参见文献[8]。

1.1 浮点算法转换为定点算法的精度问题

函数 $\text{int}(k)$ 表示把 k 通过四舍五入转换为一个相邻的整数 ($0 \leq |k - \text{int}(k)| < 0.5$) 或者通过直接截尾把 k 转换为一个相邻的整数 ($0 \leq |k - \text{int}(k)| < 1$)。

定理 1: 用 $\text{int}(k)$ 把浮点运算 $a_0 = b_0 \times c_0$ (a_0, c_0 为浮点数, b_0 为整型或者浮点数据) 转换为定点运算: $a_1 = \text{int}(b_0 \times c_0 \times k_1)/k_1$ 和 $a_2 = \text{int}(b_0 \times c_0 \times k_2)/k_2$, 其中 k_1, k_2 为正整型数据 (正整数) 且 $t = k_2/k_1$, t 为正整数, $k_2 > k_1$, $\Delta_1 = a_0 - a_1$, $\Delta_2 = a_0 - a_2$, 则 $|\Delta_2| \leq |\Delta_1|$ 。

证明: 因为 $\Delta_1 = a_0 - a_1$

所以 $k_2\Delta_1 = k_2a_0 - k_2a_1 = b_0c_0k_2 - \text{int}(b_0c_0k_1)k_2/k_1$,

所以 $k_2\Delta_2 = k_2a_0 - k_2a_2 = b_0c_0k_2 - \text{int}(b_0c_0k_2)$ 。

(1) 当 $\text{int}(k)$ 表示通过直接截尾转换为相邻的一个整数时, 显然有: $0 \leq \Delta_1, \Delta_2 < 1$, 且 $0 \leq k_2\Delta_2 < 1$, 又因 k_2/k_1 为正整数, 所以 $\text{int}(b_0c_0k_1)k_2/k_1$ 也是一个整数, 如果 $\Delta_2 > \Delta_1$ 则 $0 \leq k_2\Delta_1 < k_2\Delta_2 < 1$ 。这说明还有一个整数 $\text{int}(b_0c_0k_1)k_2/k_1$ 满足条件:

$0 \leq b_0c_0k_1 - \text{int}(b_0c_0k_1)k_2/k_1 < 1$ 且

$\text{int}(b_0c_0k_1)k_2/k_1 \neq \text{int}(b_0c_0k_1)$

这与 $\text{int}(k)$ 定义矛盾, 命题得证。

(2) 当 $\text{int}(k)$ 表示通过四舍五入转换为相邻的整数时, 显然有: $|\Delta_1|, |\Delta_2| < 0.5$, 且 $|k_2\Delta_2| < 0.5$, 又 k_2/k_1 为正整数, 所以 $\text{int}(b_0c_0k_1)k_2/k_1$ 也是一个整数, 如果 $|\Delta_2| > |\Delta_1|$, 则 $|k_2\Delta_1| < |k_2\Delta_2| < 0.5$ 。这也说明还有一个整数 $\text{int}(b_0c_0k_1)k_2/k_1$ 满足条件:

$|b_0c_0k_1 - \text{int}(b_0c_0k_1)k_2/k_1| < 0.5$ 和

$\text{int}(b_0c_0k_1)k_2/k_1 \neq \text{int}(b_0c_0k_1)$

这与 $\text{int}(k)$ 定义矛盾, 命题得证。

推论 1: 把定点运算 $a_0 = b_0 \times c_0$ (a_0, c_0 为浮点数, b_0 为整数) 转换为: $a_1 = \text{int}((b_0k_1) \times [c_0k_2]/k_3)$ 和 $a_2 = \text{int}((b_0k_4) \times [c_0k_2]/k_3)$, 其中 $[\]$ 表示四舍五入操作, a_1, a_2 为整数, k_1, k_2, k_3, k_4 为正整数且 k_4/k_1 为整数, $k_3 > k_1, k_3 > k_2, k_3 > k_4$ 。定义:

$\Delta_1 = a_0 - a_1k_3/(k_1k_2), \Delta_2 = a_0 - a_2k_3/(k_4k_2), \Delta_3 = c_0 - [c_0k_2]/k_2$

$a_1^* = \text{int}((b_0k_1)/[c_0k_2]/k_3)/k_1$

$a_2^* = \text{int}((b_0k_4)/[c_0k_2]/k_3)/k_4$

$a_0^* = b_0 \times [c_0k_2]/k_3 = b_0 \times (c_0 - \Delta_3)k_2/k_3$

$\Delta_1^* = a_0^* - a_1^*, \Delta_2^* = a_0^* - a_2^*$

如果 $|b_0\Delta_3k_2/k_3|$ 相对于 $|\Delta_1^*|$ 和 $|\Delta_2^*|$ 是个很小的值, 则 $|\Delta_2| \leq |\Delta_1|$ 。

证明: 显然根据定理 1 有: $|\Delta_2^*| \leq |\Delta_1^*|$,

又因 $a_1 = (a_0^* - \Delta_1^*)k_1, a_2 = (a_0^* - \Delta_2^*)k_4$, 则:

$\Delta_1 = a_0 - (a_0^* - \Delta_1^*)k_3/k_2 = a_0 - (b_0c_0 - b_0\Delta_3 - \Delta_1^*k_3/k_2) = b_0\Delta_3 + \Delta_1^*k_3/k_2 = (k_3/k_2)(b_0\Delta_3k_2/k_3 + \Delta_1^*)$

因为 $|b_0\Delta_3k_2/k_3|$ 相对于 $|\Delta_1^*|$ 是个很小的值,

所以 $\Delta_1 \approx (k_3/k_2)\Delta_1^*$ 。

同理可得: $\Delta_2 \approx \Delta_2^*k_3/k_2$

又因 $k_3/k_2 > 1$ 且 $|\Delta_2^*| \leq |\Delta_1^*|$

所以 $|\Delta_2| \leq |\Delta_1|$, 命题得证。

在后面的定点快速算法中, 输入整型数据相当于推论 1 中的 b_0 , 对余弦系数值扩大 $2^{14.5}$ (相当于 k_2) 倍取整^[9] 相当于推论 1 中的 $[c_0k_2]$ 系数, 此时 $[c_0k_2]/k_2$ 与 c_0 的误差绝对值 (根据文中用到的余弦值) 小于 1.2×10^{-5} , 对结果用 PACK16MSB 做处理相当于推论 1 中的除以 k_3 , 此时 k_2/k_3 取值为 $2^{-1.5}$, 在文献[9] 以及文中算法中 b_0 (IFIR16 操作的参数), $|b_0| \leq 255 \times 2^5$ 。所以 $|b_0\Delta_3k_2/k_3| \leq 0.035$, 实验表明, 实际中绝大多数输入产生的取值远远小于该值, 而在文献[8] 不对输入数据 b_0 扩大相当于使 $a_1^* = \text{int}((b_0k_1) \times [c_0k_2]/k_3)/k_1$ 中 k_1 取 1, 并产生 $|\Delta_1^*|$, 文中对输入数据扩大 16 倍后相当于使 $a_2^* = \text{int}((b_0k_4) \times [c_0k_2]/k_3)/k_4$ 中的 k_4 取 16, 并产生 $|\Delta_2^*|$, 对绝大多数输入而言, 它们大于 0.1, 所以根据推论 1, 要高精度还应当适当扩大输入数据的值。

1.2 一种在基于 VLIW 的 DSP 芯片上执行时间的计算方法

基于 VLIW 的 DSP 支持并行计算, 但是其并行受到一定的限制, 以 PNX1500 为例, 该芯片 CPU 最多允许 5 条指令并行执行, 而且由于如 IFIR16 等指令不是可以放在任意指令位的, 所以从理论上说一种算法在 VLIW 的 DSP 上的执行时间不能简单地依靠所需要的指令数来衡量。例如: 方法 1 要执行 6 条 IFIR16 操作和 2 条 DSPIDUALADD 操作; 方法 2 要执行 6 条 IFIR16 操作和 4 条普通加法。表 1 给出了两种方法的指令排列情况。

其中“+”表示普通加法。从表 1 看出两种方法各执行 10 次都需要 30 个时钟周期来安排指令执行。

文献[7]的算法虽然通过实验验证了该快速算法的有效性, 但是没有从理论上说明为什么算法在 PHILIPS 生产的允许多种指令并行的媒体处理器上达

到提高效率的效果。在这里给出了一种用来计算算法在 VLIW 的 DSP 上的执行时间的方法,以帮助分析现有算法的优劣,同时指导改进算法。对于只需要执行一次的算法必须排出各个指令的执行顺序才能得到计算所需要的指令周期,但是对于需要反复连续执行多次的算法可以按照下面的步骤计算多次执行的时间:

表1 两种方法的指令排列表

	Issue slot 1	Issue slot 2	Issue slot 3	Issue slot 4	Issue slot 5
方法1	DSPIDUALADD	IFIR16	IFIR16		
	DSPIDUALADD	IFIR16	IFIR16		
		IFIR16	IFIR16		
方法2	+	IFIR16	IFIR16	+	
	+	IFIR16	IFIR16	+	
		IFIR16	IFIR16		

(1) 找出方法中的最小运算周期。例如 8×8 DCT 的水平 8 点 DCT 周期是 2 次,垂直 8 点 DCT 的周期是 1 次。

(2) 如果一共有 N 个指令位,每个指令位的使用次数为: $x_1, x_2, x_3, \dots, x_n$, 每个指令位的权值为: $w_1, w_2, w_3, \dots, w_n$, 则给每个指令位使用次数和权值都赋予初始值零。

(3) 列出每一个运算周期所有需要执行的指令并把指令按可以使用指令位的数量从小到大排序,例如 IFIR16 可以使用 2、3 两个指令位, DSPIDUALADD 可以使用 1、3、5 三个指令位, 则 IFIR16 排在 DSPIDUALADD 之前。

(4) 计算权值:从第一条指令开始,如果 A 可以使用 n 个指令位: $a_2, a_2, a_3, \dots, a_n$, 则做如下变换: $w_{a1} = w_{a1} + 1/n, w_{a2} = w_{a2} + 1/n, \dots, w_{an} = w_{an} + 1/n$ 。直到计算完最后一条指令的权值。

(5) 计算指令位的指令数:从排序后的第一个指令开始查看,如果指令 A 可以使用 n 个指令位: $a_1, a_2, a_3, \dots, a_n$, 则找出其中 x_i 最小的指令位,然后作 $x_i = x_i + 1$, 如果同时有几条指令的值相同则取权值最小的指令位作 $x_i = x_i + 1$ 。如果还有权值相同的则任取一个指令位作上述变换。

(6) 找出所有指令位中使用次数最大的值 x_{\max} , 这个值就是一次执行周期所需要的时间。

注意,以上的估计方法是针对一种需要连续执行多次的算法的执行时间。虽然一种算法在一次执行中,它所使用到的指令之间有相互间的依赖关系,但是由于该算法是多次反复执行,这种依赖关系不影响文中计算方法的正确性,例如需要计算 10000 次算法 AA:

$$y_1 = \text{IFIR16}(x_1, x_2); y_2 = \text{IFIR16}(x_3, x_4);$$

$$y_3 = \text{IFIR16}(x_5, x_6); y_4 = \text{IFIR16}(x_7, x_8);$$

$$z_1 = \text{DSPIDUALADD}(y_1, y_2);$$

$$z = \text{DSPIDUALADD}(y_3, y_4);$$

该算法第 i 次计算可以按表 2 安排。

表2 算法 AA 第 i 次计算的指令安排

	Issue slot 1	Issue slot 2	Issue slot 3	Issue slot 4	Issue slot 5
1	$z_{i-1,1}$	$y_{i,1}$	$y_{i,2}$		$z_{i-1,2}$
2		$y_{i,3}$	$y_{i,4}$		

其中 i 表示执行第 i 次的值。例如 $y_{i,1}$ 表示第 i 次计算 y_1 。但是为了更为准确地计算运行时钟周期数,一般还要单独计算第 1 次和最后一次执行的时钟周期数。

1.3 改进的 8×8 DCT 算法

对于在 VLIW 结构的 CPU 上运行的算法,它的执行时间与算法和算法的实现方式有关。例如,把算法单独写成一个子程序和把算法与其他方法写在一个子程序中的执行时间就不一样。前者由于算法在单独的程序中,所以进入和退出子程序的时候都要访问内存,而后者由于是嵌入在子程序中,所以在得到数据的时候可以直接从前面使用过的寄存器中直接取值,而输出的时候由于后面的(例如量化)算法又要用到这些结果所以可以不必先写回内存。为了说明算法优越性,必须要有一个标准,所以先作如下假设:

按 PHILIPS 公司提供的优化算法,文献[6]把 8×8 DCT 作为一个单独的子程序,输入数据和输出数据都采用文献[6]的格式,即输入输出都把两个相邻的 32 位数据的低 16 位合并,分别放入一个 32 位数据的高 16 位和低 16 位中。对于 VLIW 结构的芯片,把两个 16 位数据组合为一个 32 位数据,利于使用其特殊的媒体处理指令,另外也利于减少内存访问次数。在 PHILIPS 公司的 1300 系列、1500 系列处理器的内存访问受到限制,从内存取数据的指令只能在第 4、5 两个指令位,所以 1 次只能取 2 个数据,而按上面的数据存放方法,1 次就相当于可以取 4 个数据到寄存器,有利于提高并行度。

图 1 给出了文献[6]提出的 8×8 DCT 算法的核心部分,移位和作四舍五入(加 0×8000)本应该一起列出,考虑到习惯和图的可读性,这些步骤在图中省略了,详细情况参见文献[6]。图中的符号解释可以参见文献[6,7],文中给出的改进 DCT 算法如图 2 所示。

对于 8×8 DCT 变换的改进主要在水平 8 点 DCT 上,图 2 给出了改进的水平 8 点 DCT 变换的核心部分,仍然采用文献[6]的垂直 8 点 DCT 方法的核心部分。

附录 A(略)给出了 C_i 的值^[6]。事实上通过图 2 计算的结果还不是真正要输出的结果,由于处理结果时需要把几次水平 DCT 变换的结果一起处理,所以并未

在图中画出,稍后将详细介绍。下面用例子来说明图中的计算。

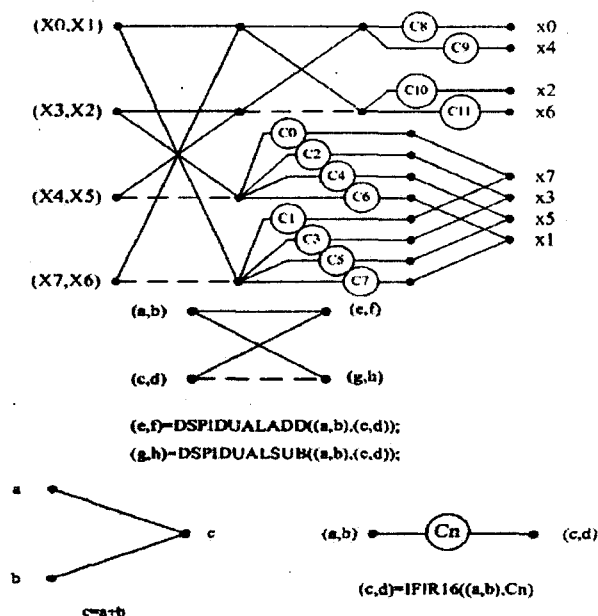


图 1 文献[6]提出的 8×8 DCT 算法的核心部分

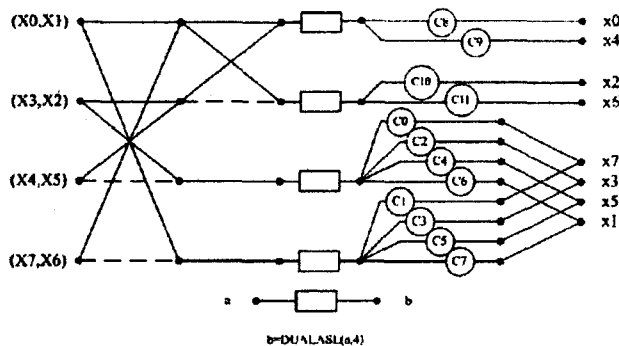


图 2 文中的水平 8 点 DCT

(1) 在 stage1 中的操作可以表示如下:

$$(t_0, t_1) = \text{DSPIDUALADD}((x_0, x_1), (x_7, x_6));$$

实际上是在 (t_0, t_1) 的高 16 位做 $x_0 + x_7$, 低 16 位做 $x_1 + x_6$; stage 2 中的操作和 stage 1 中类似。

(2) 在 stage 3 中的操作可以表示如下:

$(q_4, q_5) = \text{DUALASL}((t_4, t_5), 4)$; 这个操作实际上是在高 16 位做 $t_4 \ll 4$, 在低 16 位做 $t_5 \ll 4$ 。

(3) 在 stage 4 的做法和文献[6,7]相同。其中圆圈中的符号 C_i 表示该操作对应的系数为 C_i 。在做完 8 次水平 DCT 后需要对数据进行处理得到垂直 DCT 的输入数据。令:第 i ($i = 0, 1, 2, \dots, 8$) 次水平变换后的输出为: $Y0_i, Y1_i, Y2_i, Y3_i, Y4_i, Y5_i, Y6_i, Y7_i$ 。

第 i 次垂直变换的每个独立数据输入为:

$$y0_i, y1_i, y2_i, y3_i, y4_i, y5_i, y6_i, y7_i。$$

在水平 DCT 结束时可以采用两种方法处理结果:

方法 1 对结果加 0×8000 后再取高 16 位。

方法 2 对结果直接取高 16 位。

文中把方法 1 叫做高精度方法,方法 2 叫低精度方法。由于输入数据都是整数,而且 C_8, C_9 是比实际需要的系数扩大了 2^{14} 倍,所以 $Y0_i, Y4_i$ 肯定是 2^{16} 的整数倍,因此,即使按方法 1 计算,对于 $Y0_i, Y4_i$ 也无需加 0×8000 的处理(仅限于水平 DCT 变换中)。下面是对于水平 DCT 的输出的处理方法。例如第 7 次垂直 DCT 的第 1 个输入为: $(y0_7, y1_7) = \text{PACK16MSB}(Y7_0, Y7_1 + 0 \times 8000)$, 同样在做完垂直 DCT 后也需要做类似的以便重新组织数据,例如:如果第 i 次垂直 DCT 输出的独立数据为: $Y0_i, Y1_i, Y2_i, Y3_i, Y4_i, Y5_i, Y6_i, Y7_i$, 最后组成 $(Y0_i, Y1_i), (Y2_i, Y3_i), (Y4_i, Y5_i), (Y6_i, Y7_i)$ 的形式,则有:

$$(Y0_i, Y1_i) = \text{PACK16MSB}(Y0_i + 0 \times 8000, Y1_i + 0 \times 8000)$$

$$(Y2_i, Y3_i) = \text{PACK16MSB}(Y2_i + 0 \times 8000, Y3_i + 0 \times 8000)$$

$$(Y4_i, Y5_i) = \text{PACK16MSB}(Y4_i + 0 \times 8000, Y5_i + 0 \times 8000)$$

$$(Y6_i, Y7_i) = \text{PACK16MSB}(Y6_i + 0 \times 8000, Y7_i + 0 \times 8000)$$

2 理论分析及实验结果

2.1 关于溢出的分析

在计算机上普遍采用的颜色模式是每个分量均用 8 位表示,每个分量输入范围是 $0 \sim 255$ 。有的时候可能会用到差值,即每个分量的绝对值范围在 $0 \sim 255$ 。文中在水平 DCT 变换的 stage3 把值扩大 16 倍,此时图 2 中 q 的最大绝对值不会大于 $255 \times 16 \times 4 = 16320$,即用 14 位来表示绝对值就足够了。在水平 DCT 结束后输出的最大绝对值不会大于 $16320 \times 2 = 32640$,即可以使用 15 位表示,而此时的 Y 值实际上已经是 32 位数据,所以在做水平 DCT 变换的过程中使用高 16 位表示有符号数据不会产生溢出。在做水平和垂直 DCT 之间的处理后,由于使用了 PACK16MSB 操作,垂直 DCT 的每个独立的输入数据使用 13 位可以表示所有值的绝对值,在做 IFIR16 操作之前最大绝对值不会大于 $8191 \times 4 = 32764$,正好在 15 位的表示范围之内。在做完 IFIR16 后数据虽然扩大了 2^{14} 倍,但是此时数据使用 32 位表示,肯定不会溢出。 8×8 DCT 结束后数据的绝对值不会大于 $255 \times 64 = 16320$,同样完全可以用 15 位表示,不会产生溢出。所以文中算法非常适合于颜色分量绝对值使用 8 位或者更少的位表示的图像和视频处理。

2.2 仿真实验的结果

通过 PHILIPS 公司提供的编译器把源文件优化后编译为汇编语言文件后(后缀为“.s”的文件)得出的指令周期数如下:文献[6]中低精度方法需要 156 个指令周期,高精度方法需要 168 个指令周期;文中低精度方法需要 132 个指令周期,高精度方法需要 140 个指令周期。

下面根据 1.1 节的分析方法分析比较文献[6]中高精度方法和文中的方法,文献[6]的低精度方法可以按文中的方法做类似的分析。

正如前面所说,由于访问内存并不是一定需要的,所以在理论上做比较的时候省去了内存访问的指令,另外一般情况下输入数据一般是按顺序排列的即图 2 中第 2 个和第 4 个输入不是 (x_3, x_2) 和 (x_7, x_6) 而是 (x_2, x_3) 和 (x_6, x_7) ,所以文中需要如同文献[6]一样使用两个 ROLI 指令来重新安排为图 2 中的输入,该指令可以在 5 个指令位中任意一个中解释执行,恢复时间 1,执行时间 1。另外将 8 点 DCT 后的移位和四舍五入等附加数据处理应该算 8 点 DCT 的一部分,对于 8×8 DCT 变换分为水平 DCT 和垂直 DCT 变换,由于每两次水平 DCT 后有一次数据重组织,所以水平 DCT 的周期为 2 次 DCT 变换,而每个垂直 DCT 可单独作为一个周期。表 3 给出了几种方法需要使用的指令和数目,表 4 按照 1.1 节分析方法给出的一个周期内每个指令位需要的指令数。

表 3 8×8 DCT 变换使用的指令比较

		文献[6]高精度算法	文中高精度算法	文中低精度算法
水平 DCT	IFIR16	24	24	24
	DSPIDUALADD/ DSPIDUALSUB	12	12	12
	DUALASL	0	8	8
	普通加减法,移位 ROLI 操作,	44	24	12
	PACK16MSB	8	8	8
垂直 DCT	IFIR16	12	12	
	DSPIDUALADD/ DSPIDUALSUB	6	6	
	DUALASL	0	0	
	普通加减法,移位操作	20	12	
	PACK16MSB	4	4	

经过分析,忽略第一次开始执行 2 个水平 DCT 与最后一次执行 2 个水平 DCT 和第一次与最后一次执行垂直 DCT 的额外指令周期数,理论上文献[6]的算法需要 144 个指令周期,而新的算法分别需要 120 和 108 个指令周期,运算速度分别提高了 16.6% 和 25% 左右。

实际上,按照前面提到的代码安排方式,每次水平

DCT 会多 4 个从内存取操作数的指令,每次垂直 DCT 会多 4 个向内存存操作数的指令,由于存取操作数的指令只能放在第 4、5 个指令位上,所以每个指令周期只能有 2 个操作数被放到寄存器中,而且取操作数的指令执行时间是 3 个指令周期,恢复时间 1 个指令周期,就是从取操作数指令开始到操作数可用有 3 个指令周期的延迟,这一点对于第一次执行水平 DCT 和最后一次执行垂直 DCT 变换的影响较大,增加了许多额外操作,另外由于在程序的开始需要向寄存器写入常量 C_i ,这些操作也会占据操作位,所以按文献[6]和文中的组织代码的方式实际需要的指令位会更多。表 5 给出了两种方法在同一环境下编译为 .s 文件后得到的实际指令周期数和新方法的计算值。

表 4 按 1.1 方法所得一个周期内每个指令位需要的指令数

		文献[6]高精度算法	文中的高精度算法	文中的低精度算法
水平 DCT	Issue slot 1	18	16	13
	Issue slot 2	17	15	12
	Issue slot 3	17	15	13
	Issue slot 4	18	15	13
	Issue slot 5	18	15	13
垂直 DCT	Issue slot 1	9	7	7
	Issue slot 2	8	7	7
	Issue slot 3	8	6	6
	Issue slot 4	9	7	7
	Issue slot 5	8	7	7

表 5 8×8 DCT 实验结果和理论结果

	文献[6]低精度算法	文献[6]高精度算法	文中的低精度算法	文中的高精度算法
实际的指令周期	152	168	132	140
计算的指令周期	151	167	131	142

表 6 是重复做 1000 次实验,每次随机取输入数据得出的误差统计表。其中 x_i 为定点计算的输出结果, x 为使用浮点计算的精确结果, error 为误差方差总和。则输出误差是: $x_i - x$, 误差方差总和是所有误差的平方和:

$$\text{error} = \sum_{\text{所有输出}} (x_i - x)^2$$

表 6 误差统计

	文献[6]低精度方法	文献[6]高精度方法	文中的低精度方法	文中的高精度方法
error	707661	36053	11697.4	7180.02
$1 \leq x_i - x \leq 2$ 的数量	13756	10984	2333	47
$2 \leq x_i - x \leq 3$ 的数量	11412	819	0	0
$3 \leq x_i - x $ 的数量	24337	7	0	0

(下转第 110 页)

$\approx 0.5195 \cos(\text{User2}, \text{User3}) \approx 0.479 \cos(\text{User2}, \text{User4})$
 $\approx 0.701 \cos(\text{User2}, \text{User5}) \approx 0.6637 \cos(\text{User2}, \text{User6})$
 $\approx 0.5157 \cos(\text{User2}, \text{User7}) \approx 0.503 \cos(\text{User3}, \text{User4})$
 $\approx 0.4756 \cos(\text{User3}, \text{User5}) \approx 0.4793 \cos(\text{User3}, \text{User6})$
 $\approx 0.4833 \cos(\text{User3}, \text{User7}) \approx 0.5142 \cos(\text{User4}, \text{User5})$
 $\approx 0.6699 \cos(\text{User4}, \text{User6}) \approx 0.5192 \cos(\text{User4}, \text{User7})$
 $\approx 0.4991 \cos(\text{User5}, \text{User6}) \approx 0.5017 \cos(\text{User5}, \text{User7})$
 $\approx 0.4756 \cos(\text{User6}, \text{User7}) \approx 0.4941$ 。

根据上面的计算,在这里采用基于中心的方法为当前用户 User1 寻找相似群体。取相似度的值大于 0.5 作为当前用户的相似用户群体,因此可得到 User3, User7 同 User1 形成相似用户群体,且 User7 同用户最相似,User3 次之。再根据公式(4)计算 User1 对微机原理和操作系统这两个项目的预测兴趣的值分别为 0.509 和 0.455,所以可以给当前用户 User1 推荐微机原理。从而完成了利用协作过滤技术实现个性化推荐。

4 结束语

随着互联网的普及和信息服务需求的增长,协作过滤技术在电子图书馆、电子商务、WEB 个性化、自动问答等系统得到了广泛的应用。文中将显式和隐式计

算用户兴趣度的方法相结合,提出了一种新的计算用户兴趣度的方法,并在此基础上研究了基于兴趣度的协作过滤技术,给出了利用协作过滤技术寻找兴趣相似的用户群体的算法。

并通过实验验证了利用该方法可以为当前用户推荐他所关心的网络事务。

参考文献:

- [1] 张莹.从商务网站用户行为数据提取用户兴趣[J].潍坊学报,2005,5(4):21-23.
- [2] Aggarwal C, Yu P. Data Mining Techniques for Personalization[J]. IEEE Data Engineering Bulletin, 2000, 23(1):4-9.
- [3] 张新香. WEB 日志挖掘在电子商务中的应用研究[J]. 计算机系统应用, 2006(1): 52-55.
- [4] 周晓兰,王随平. WEB 文本挖掘中用户兴趣模型的建立和更新[J]. 湘潭师范学院学报:自然科学版, 2006, 28(3):33-36.
- [5] Resnick P, Iacovon N, Bergstrom P, et al. Gouplen: An Open Architecture for Collaborative Filtering of Netnews[C]//In: Proceedings of CSCW'94. Chapel Hill, NC: [s. n.], 1994.
- [6] Dahlen B J, Konstan J A, Herlocker J L, et al. Turn - starting Movielens: user Benefits of starting a Collaborative Filtering system with "Dead Data"[R]. Minneapolis: university of Minnesota, 1998.

(上接第 105 页)

3 结论

给出了浮点转定点问题的一个定理和一个推论,对于文中的算法在浮点转定点运算精度问题上给出了理论上的支持。针对目前已有 DCT 算法误差较大的缺点,提出了两种新的 VLIW 结构下的定点 8×8 DCT 快速算法,按照做完水平 DCT 变换后的处理方式,新算法又可分为高精度算法和低精度算法。新高精度算法比已有的定点高精度算法^[6]分别提高 13.4% 和 21.4%,而误差方差总和分别降低了 80% 和 67.5%。高精度算法比已有的低精度算法^[6]运算速度分别提高 8.5% 和 17.2%,而误差方差总和分别降低了 98.98% 和 98.3%。另外,文中还给出来了一种估计在同时执行多个指令操作的 DSP 中估计算法实际运行时间的方法,这些方法对于在允许多个指令同时执行的 DSP 上优化算法有理论上的指导作用。

参考文献:

- [1] Elnaggar A, Alnuweiri H M. A new multidimensional recursive architecture for computing the discrete cosine transform[J]. IEEE Trans on Circuits and Systems for Video Technology, 2000, 10(1):113-119.
- [2] Wang Z S, He Z Y. A generalized fast algorithm for 2-D discrete cosine transform and its application to motion picture coding[J]. IEEE Trans on Circuits and Systems II, 1999, 46(5):617-627.
- [3] Chen Xinjian, Dai Qionghai, Li Chunwen. A fast algorithm for computing multidimensional DCT on certain small sizes[J]. IEEE Trans on Signal Processing, 2003, 51(1):213-220.
- [4] Loeffler C. Practical fast 12D DCT algorithms with 11 multiplications[J]. Acoustics, Speech, and Signal Processing, 1989, 289(12):988-991.
- [5] Chen W A, Harrison C, Fralick S C. A Fast computational Algorithm for the Discrete Cosine Transform[J]. IEEE Transactions on Communications, 1977, 25(9):1004-1011.
- [6] Volume 2 Part D CookBook Trimedia Compilation System 4.3 User Manual v1.0[S]. 2003:22-28.
- [7] 李学明,李继.用超长指令实现 DCT 的新算法[J]. 电子学报, 2003, 31(7):1074-1077.
- [8] TM3260 Architecture Databook TriMedia VLIW Core Rev. 1.02[S]. 2004.
- [9] Hou H S. A Fast Recursive Algorithm for computing the Discrete Cosine Transform[J]. IEEE Transactions on Acoustics, Speech and Signal Processing, 1987, 35(10):1455-1461.