第35卷 第17期 计 算 机 工 程 2009年9月
Vol.35 No.17 Computer Engineering September 2009
·多媒体技术及应用· 文章编号：1000—3428(2009)17—0223—03 文献标识码：A 中图分类号：TP51

面向龙芯平台的快速DCT算法及其实现

王 明，彭成磊，都思丹
(南京大学电子科学与工程系，南京 210093)

摘 要：针对龙芯 2E 平台不能流畅播放视频文件的问题，对视频变换过程进行优化，采用一种新的离散余弦变换(DCT)算法，基于龙芯 2E 多媒体指令集对其进行实现，用该算法替代 FFmpeg 软件中的 DCT 算法。测试结果表明，新的 DCT 算法性能比传统 DCT 算法提高近 11 倍，优化后的 FFmpeg 软件编码速度提高 10%左右。

关键词：离散余弦变换算法；龙芯 2E 多媒体指令集；FFmpeg 软件

Loongson Platform Oriented Fast DCT Algorithm and Its Implementation

WANG Ming, PENG Cheng-lei, DU Si-dan
(Department of Electronic Science and Engineering, Nanjing University, Nanjing 210093)

【Abstract】Aiming at the problem that video files can not be played fluently on the Loongson 2E platform, this paper optimises the video transform process and uses a new Discrete Cosine Transform(DCT) algorithm. This algorithm is realized based on multimedia instruction set of Loongson 2E platform and used to alternate the DCT algorithm in FFmpeg software. Text results show that the performance of new DCT algorithm is nearly 11 times higher than the traditional one, and the coding rate of optimised FFmpeg software increases nearly by 10%.

【Key words】 Discrete Cosine Transform(DCT) algorithm; Loongson 2E multimedia instruction set; FFmpeg software

1 概述

1.1 龙芯 2E 处理器

龙芯 2E 处理器是中科院于 2005 年研制成功的最高主频达 1.0 GHz 的处理器。它采用 4 发射超标量、超流水结构，其片内一级指令和数据高速缓存均为 64 KB，片外二级高速缓存最多可达 8 MB。龙芯 2E 处理器的指令集与 MIPS3 完全兼容，且拥有专用指令集[1-2]。

1.2 FFmpeg 软件

FFmpeg 软件提供音频和视频流媒体的解决方案，支持多数格式的音频、视频编解码。开源软件 Mplayer 使用 FFmpeg 编解码库。

FFmpeg 软件主要由以下 5 个部分组成：(1)Ffmpeg 为视频文件转换命令行工具，支持通过实时电视卡抓取和编码形成视频文件。(2)Ffserver 为基于 HTTP、用于实时广播的多媒体服务器，支持时间平移。(3)Ffplay 是一个简单的媒体播放器，它使用 SDL 和 FFmpeg 库开发。(4)Libavcodec 包含所有 FFmpeg 音频、视频编解码库。(5)Libavformat 包含所有普通音频、视频格式的解析器和产生器的库。

2 DCT 算法

2.1 DCT 算法介绍

离散余弦变换(Discrete Cosine Transform, DCT)是常用的数据压缩变换方法。对于任何连续的实对称函数，其傅里叶变换中只含有余弦项，其物理意义明确。

在视频变换中，先将整体图像分成 $N\times N$ 像素块，然后逐一对每个像素块进行 DCT。由于视频图像在时间上和空间上有一定连续性，其高频分量较小，相应图像的高频分量系数几乎为零，因此可以采用粗量化，忽略高频分量以减少数据传输，而不会在视觉上产生太大影响。

DCT 算法在视频编解码中具有重要作用，因此，有必要研究 DCT 快速算法。本文针对龙芯平台的特点，实现一种非 X86 通用平台结构的 DCT 快速算法。

2.2 蝶形 DCT 算法分析

FFmpeg 软件中采用 8 点二维 DCT 变换，其定义如下：

$$f_{nm}=\frac{1}{4}c_n c_m\sum_{i=0}^{7}\sum_{j=0}^{7}\cos\frac{\pi n(2i+1)}{16}\cos\frac{\pi m(2j+1)}{16}x_{ij} \tag{1}$$

式(1)可以表示为 $\boldsymbol{Y}_8=\boldsymbol{A}_8\boldsymbol{X}_8\boldsymbol{B}_8$ 的矩阵相乘形式。由于 cos 系数矩阵的转置性质，因此进一步改写为 $\boldsymbol{Y}_8=\boldsymbol{C}_8\boldsymbol{X}_8\boldsymbol{C}_8^{\mathrm{T}}$，即存在一个正交矩阵 $\boldsymbol{C}_8$，$\boldsymbol{C}_8$ 与 $\boldsymbol{C}_8^{\mathrm{T}}$ 对矩阵 $\boldsymbol{X}_8$ 左右相乘，实现了矩阵 $\boldsymbol{X}_8$ 的 DCT 变换。

根据式(1)可以求得

$$\boldsymbol{C}_8=\frac{1}{2}\begin{bmatrix} r(4) & r(4) & r(4) & r(4) & r(4) & r(4) & r(4) & r(4) \\ r(1) & r(3) & r(5) & r(7) & -r(7) & -r(5) & -r(3) & -r(1) \\ r(2) & r(6) & -r(6) & -r(2) & -r(2) & -r(6) & r(6) & r(2) \\ r(3) & -r(7) & -r(1) & -r(5) & r(5) & r(1) & r(7) & -r(3) \\ r(4) & -r(4) & -r(4) & r(4) & r(4) & -r(4) & -r(4) & r(4) \\ r(5) & -r(1) & r(7) & r(3) & -r(3) & -r(7) & r(1) & r(5) \\ r(6) & -r(2) & r(2) & -r(6) & -r(6) & r(2) & -r(2) & r(6) \\ r(7) & -r(5) & r(3) & -r(1) & r(1) & -r(3) & r(5) & -r(7) \end{bmatrix}$$

其中，$r(k)=\cos(\pi k/16)$。由于 cos 函数具有周期性质，因此矩阵 $\boldsymbol{C}_8$ 进一步分解为 $\boldsymbol{C}_8=\frac{1}{2}\boldsymbol{P}_8\boldsymbol{M}_8\boldsymbol{A}_8$ 的矩阵相乘形式。

基金项目：国家“863”计划基金资助项目(2006AA010201)
作者简介：王 明(1984－)，男，硕士，主研方向：嵌入式系统，视频处理，Linux 系统；彭成磊，讲师；都思丹，教授、博士
收稿日期：2009-04-11 **E-mail**：wangming0304@gmail.com

$$\boldsymbol{A}_8=\begin{bmatrix}1&0&0&0&0&0&0&1\\0&1&0&0&0&0&1&0\\0&0&1&0&0&1&0&0\\0&0&0&1&1&0&0&0\\1&0&0&0&0&0&0&-1\\0&1&0&0&0&0&-1&0\\0&0&1&0&0&-1&0&0\\0&0&0&1&-1&0&0&0\end{bmatrix}$$

$$\boldsymbol{P}_8=\begin{bmatrix}1&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0\\0&1&0&0&0&0&0&0\\0&0&0&0&0&1&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&0&0&1&0\\0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&1\end{bmatrix}$$

$$\boldsymbol{M}_8=\begin{bmatrix}r(4)&r(4)&r(4)&r(4)&0&0&0&0\\r(2)&r(6)&-r(6)&-r(2)&0&0&0&0\\r(4)&-r(4)&-r(4)&r(4)&0&0&0&0\\r(6)&-r(2)&r(2)&-r(6)&0&0&0&0\\0&0&0&0&r(1)&r(3)&r(5)&r(7)\\0&0&0&0&r(3)&-r(7)&-r(1)&-r(5)\\0&0&0&0&r(5)&-r(1)&r(7)&r(3)\\0&0&0&0&r(7)&-r(5)&r(3)&-r(1)\end{bmatrix}$$

上述 3 个矩阵在程序实现时体现为:(1)矩阵 $\boldsymbol{A}_8$ 表示数字的加减法运算;(2)矩阵 $\boldsymbol{P}_8$ 由于每行仅有一个不为零的元素,且其值为 1,因此在存放计算结果时考虑位置,不需要计算;(3)矩阵 $\boldsymbol{M}_8$ 仅需计算 2 个 4×4 的子矩阵,且其值固定,在构造系数矩阵时实现。

2.3 优化后的 DCT 算法

由上述蝶形 DCT 算法的分解可知,分解得到的矩阵 $\boldsymbol{M}_8$ 比原始矩阵 $\boldsymbol{C}_8$ 多了很多 1 和 0 元素。1 和 0 代表计算时不需要具体计算,可以减少变换所需计算次数。因此,可以尝试继续分解矩阵 $\boldsymbol{C}_8$,增加 1 和 0 元素,以减少计算次数。

观察矩阵 $\boldsymbol{M}_8$ 的如下 4×4 子矩阵,可以发现其具有对称性,因此,可以进一步分解矩阵 $\boldsymbol{M}_8$ [3-4]为 $\boldsymbol{M}_8=\boldsymbol{D}_8\cdot\boldsymbol{B}_8\cdot\boldsymbol{E}_8\cdot\boldsymbol{F}_8$ 的形式。

$$\boldsymbol{D}_8=\begin{bmatrix}r(4)&0&0&0&0&0&0&0\\0&0&r(2)&0&0&0&0&0\\0&r(4)&0&0&0&0&0&0\\0&0&0&r(2)&0&0&0&0\\0&0&0&0&r(1)&0&0&0\\0&0&0&0&0&0&r(3)&0\\0&0&0&0&0&0&0&r(3)\\0&0&0&0&0&r(1)&0&0\end{bmatrix}$$

$$\boldsymbol{B}_8=\begin{bmatrix}1&1&0&0&0&0&0&0\\1&-1&0&0&0&0&0&0\\0&0&1&\frac{r(6)}{r(2)}&0&0&0&0\\0&0&\frac{r(6)}{r(2)}&-1&0&0&0&0\\0&0&0&0&1&\frac{r(7)}{r(1)}&0&0\\0&0&0&0&\frac{r(7)}{r(1)}&-1&0&0\\0&0&0&0&0&0&1&\frac{r(5)}{r(3)}\\0&0&0&0&0&0&\frac{r(5)}{r(3)}&-1\end{bmatrix}$$

$$\boldsymbol{E}_8=\begin{bmatrix}1&0&0&1&0&0&0&0\\0&1&1&0&0&0&0&0\\1&0&0&-1&0&0&0&0\\0&1&-1&0&0&0&0&0\\0&0&0&0&1&1&0&0\\0&0&0&0&0&0&1&1\\0&0&0&0&1&-1&0&0\\0&0&0&0&0&0&1&-1\end{bmatrix}$$

$$\boldsymbol{F}_8=\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&r(4)&r(4)&0\\0&0&0&0&0&r(4)&-r(4)&0\\0&0&0&0&0&0&0&1\end{bmatrix}$$

相比矩阵 $\boldsymbol{M}_8$,$\boldsymbol{D}_8$,$\boldsymbol{B}_8$,$\boldsymbol{E}_8$,$\boldsymbol{F}_8$ 中出现了更多 1 和 0 元素,由上述分析可知,1 和 0 在程序中仅体现为存储或加减运算,而处理器的特点是加减运算需要的时钟周期数远少于乘除运算。

程序实现时采用如下方法:由于 $\boldsymbol{D}_8$ 矩阵每行只有一个非零元素,但其值不为 1,因此将其绝对值放到系数矩阵 $\boldsymbol{B}_8$ 的构造中实现,此时,$\boldsymbol{D}_8$ 矩阵中所有非零元素都可以视为 1,对于 $\boldsymbol{D}_8$ 矩阵的操作可以参考 $\boldsymbol{P}_8$ 矩阵,存放数据时按一定方式存放,以进一步减少计算次数。对矩阵 $\boldsymbol{B}_8$ 的行数进行如表 1 所示的变换。

表 1 矩阵 $\boldsymbol{B}_8$ 的变换

所在行	所乘因子
0, 4	$r(4)$
1, 7	$r(1)$
2, 6	$r(2)$
3, 5	$r(3)$

经过上述处理后,可得

$$\begin{aligned}\boldsymbol{Y}_8&=\boldsymbol{C}_8\boldsymbol{X}_8\boldsymbol{C}_8^{\mathrm{T}}=\\&\frac{1}{2}\boldsymbol{P}_8\boldsymbol{M}_8\boldsymbol{A}_8\boldsymbol{X}_8\frac{1}{2}\boldsymbol{A}_8^{\mathrm{T}}\boldsymbol{M}_8^{\mathrm{T}}\boldsymbol{P}_8^{\mathrm{T}}\\&\frac{1}{2}\boldsymbol{P}_8\boldsymbol{D}_8\boldsymbol{B}_8\boldsymbol{E}_8\boldsymbol{F}_8\boldsymbol{A}_8\boldsymbol{X}_8\frac{1}{2}\boldsymbol{A}_B^{\mathrm{T}}\boldsymbol{M}_8^{\mathrm{T}}\boldsymbol{P}_8^{\mathrm{T}}\end{aligned}\tag{2}$$

式(2)是编码时用到的计算式,变换顺序如下:$\boldsymbol{X}_8$ 往前进行列变换,得到的结果往后进行行变换。程序中包括 8 次行变换,2 次列变换。利用龙芯 2E 64 位存储的特点,一次处理矩阵 4 列,这是上述过程继续分解 M_8 的原因,变换后方便编程。解码是式(2)的逆过程,实现方法与编码类似。

2.4 优化后 DCT 算法的误差分析

由上述分解过程可知,该变换方法存在一定误差,主要原因如下:

(1)有理数代替无理数计算,$\cos(x)$ 为无理数。

(2)指令 pmulhw 的影响。

(3)IEEE 精度要求以及该变换方法的精度分析带来的影响。

原因(1)无法避免,因此,程序中采用 16 位无符号数来计算,以尽量减少其影响。

对原因(2)分析如下:变换中使用的乘法操作,指令 pmulhw 计算的结果比真实值偏小,因为该指令实际计算 $\left\lfloor\frac{x\cdot\boldsymbol{iconst}}{2^{16}}\right\rfloor$,$\boldsymbol{iconst}$ 代表前面定义的系数矩阵。由于乘加指令要除以 2^{16},为了避免计算结果太小,因此在构造系数矩阵时,将数据左移 16 位,以消除右移的影响。

对原因(3)分析如下：IEEE 要求变换精度的最大误差为 0.001 5[5]。由于构造系数矩阵时加上了 0.5，因此 2E 处理器实际计算的表达式为$\lfloor A\cdot x+B\cdot y+0.5\rfloor$，分析变换数据 x，y 的各种情况，包括以下 3 种组合：

$\lfloor A\cdot x+B\cdot y+0.5\rfloor=\lfloor A\cdot x\rfloor+\lfloor B\cdot y\rfloor$，出现概率为 1/8；

$\lfloor A\cdot x+B\cdot y+0.5\rfloor=\lfloor A\cdot x\rfloor+\lfloor B\cdot y\rfloor+1$，出现概率为 3/4；

$\lfloor A\cdot x+B\cdot y+0.5\rfloor=\lfloor A\cdot x\rfloor+\lfloor B\cdot y\rfloor+2$，出现概率为 1/8。

在程序实现中，使用概率最大的组合，即$\lfloor A\cdot x+B\cdot y+0.5\rfloor=\lfloor A\cdot x\rfloor+\lfloor B\cdot y\rfloor+1$代表所有情况，以避免对多种情况进行判断而耗费时间，即在精度和计算速度之间取平衡点。只要在列变换的最后一步乘法计算中加上常数 1，对于一个 8×8 的矩阵只要增加 8 次加法操作，就能极大降低误差带来的影响。

2.5 程序结构和主要指令

用如下程序来实现式(2)，从而实现 DCT 过程：

(1)行变换宏

```
#define DCT_8_ROW_LOONGSON(A1,A2,A3,A4)//A1 导入处
//理前的行数据，A2 存放处理后的行数据，A3 系数矩阵，A4 精度
//调整矩阵
    {.set mips3  //指明使用 mips3 汇编指令集
     导入行数据并完成加减法操作
     导入系数完成乘加，精度调整操作
     调整数据并存回 A2}
```

(2)列变换宏

```
#define DCT_8_COL_LOONGSON(A1,A2)//A1 导入列数据，A2
//存放列数据
    {.set mips3  //指明使用 mips3 指令集
     导入行数据完成加法运算
     引入列变换系数计算
     重新存放数据}
    void ff_fDCT_loongson(short *block){
    asm volatile(
```

(3)2 个对列变换的宏(列变换一次操作 4 列数据)

```
DCT_8_COL_LOONGSON(A1,A2)
…
```

(4)8 个对行变换的宏(行变换一次操作 1 行数据)

```
DCT_8_ROW_LOONGSON(A1,A2,A3,A4)
…
 ::"r"(block),"r"(rounder_0),"r"(tab_i_04_xmm),"r"(tg_1_16)
//block 待变换的数据，rounder_0 精度调整矩阵，tab_i_04_xmm,
//tg_1_16 为系数矩阵
      );  }
```

对上述程序使用的主要指令集描述如下：

(1)pmaddhw，半字到字的乘加指令，每次操作 4 个数。

(2)packsswh，打包指令，算完存储之前的最后一步需要。

(3)pshufh，换字节指令，存储数据前的位置变换。

(4)pmulhh，半字相乘，扔掉低 16 位，是产生主要误差的根源。

(5)psrah，移位指令，列变换时要右移 6 位。

3 性能提升测试

用程序实现上述分解过程，以测试性能提升效果。

3.1 DCT 算法性能分析与测试方法

在处理器为龙芯 2E 的龙梦机顶盒福珑迷你 PC 上运行优化前后的 DCT 函数，查看程序运行时花费的时钟周期数。利用龙芯 2E 处理器内置的性能计数器，可以读取处理器运行的时钟周期数。在需要读取该计数器值的地方加上 asm volatile("mfc0 %0, $25":"=r"(variable))代码，读取程序运行前后该寄存器的值，就可以得到此函数运行时花费的时钟周期数。随机对 10 组矩阵数据分别进行变换，取时间开销的平均值，得到如表 2 所示的结果。

表 2 优化前后 DCT 算法的测试结果

算法	时钟个数
优化前 DCT	95 312
优化后 DCT	8 720

由表 2 可以看出，优化后的算法比定义实现的算法速度提高了 10.93 倍，对实时性要求高的视频处理而言，其影响很大。

3.2 DCT 变换的精度分析

随机生成 10 组 8×8 的矩阵，分别在龙梦机顶盒福珑迷你 PC 上使用本文算法编写的 DCT 程序和按 DCT 变换定义编写的程序对其进行 DCT 变换，并对比其变换结果。采用 IEEE 对视频中 DCT 变换提出的误差界定方法——最小平方误差进行比较，即

$$pmse=\frac{\sum_{i=1,j=1}^{8,8}\left(a(i,j)-b(i,j)\right)^2}{64} \tag{3}$$

式(3)为 2 个矩阵对应位置数据差的平方和的均值[5]，IEEE 标准要求误差低于 0.001 5，测试发现本文算法误差在 0.000 5~0.001 3 之间。对该算法转换的矩阵进行反变换后，按式(3)与原始矩阵相比得到的误差仍然符合 IEEE 要求，因此，该算法在精度上满足视频变换要求。

3.3 优化后的 FFmpeg 软件性能分析

用本文算法取代 FFmpeg 软件中原来的 DCT，查看在龙芯平台下运行性能的提升效果。分别用优化前后的 FFmpeg 软件对同一段视频编解码，结果如表 3 所示。

表 3 视频编码测试结果

所用软件	编码前视频大小/MB	编码后视频大小/MB	耗费时间/s
优化前 FFmpeg	155	1.4	43.295
优化后 FFmpeg	155	1.4	39.262

由表 3 可以看出，本文算法编码 155 MB 的视频文件时，节约了 4 s(约 10%)左右。分别用优化前后的 FFmpeg 软件解码此前编码出的 2 段视频，比较播放效果，优化前播放停顿的现象明显消失，优化效果明显。

使用软件性能分析工具 gprof 分析优化后的 FFmpeg 软件运行结果，可知 DCT 变换在 FFmpeg 软件运行中占的比重约为 10.4%。由阿姆达尔准则[6]可知，在整个软件体系中，某部分性能的提高与整体性能提高之间的关系为

$$\frac{1}{(1-\text{百分比})+\frac{\text{百分比}}{X_{\text{性能提升}}}}=\frac{T_{\text{优化前}}}{T_{\text{优化后}}} \tag{4}$$

其中，$X_{\text{性能提升}}$是性能提升参数；$T_{\text{优化前}}$和 $T_{\text{优化后}}$分别代表优化前后编解码时间。

将测试所得数据代入式(4)可知，系统整体性能可以提升 10%左右。

4 结束语

本文以国产处理器龙芯 2E 平台为基础，在其上实现一种快速 DCT 算法，使龙芯平台对视频文件的编解码处理得到优化，提高了多媒体播放性能。

本文算法有利于推动龙芯硬件平台被更广泛地应用于视

（下转第 228 页）