文章编号: 1001-2486 (2000) 04-0090-04

H.263 编码器中运动补偿的 MMX 加速*

程翥, 楼生强, 皇甫堪

(国防科技大学电子科学与工程学院, 湖南 长沙　410073)

摘　要: 分析了甚低码率视频压缩标准 H.263 建议中运动补偿算法, 介绍了 MMX 技术的特点、数据结构和使用方法, 在此基础上用 MMX 技术实现了运动补偿算法。

关键词: H.263; 运动补偿; MMX 技术; SIMD

中图分类号: TN919.3+1　　文献标识码: B

MMX Acceleration of Motion Compensation in H.263 Coder

CHENG Zhu, LOU Sheng-qiang, HUANGFU Kan

(College of Electronic Science and Engineering, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: We use the technology of MMX to realize the motion compensation algorithm of H.263 based on the analysis of motion compensation in the H.263 and the MMX instruction set.

Key words: H.263; motion compensation; MMX; SIMD

1996 年 3 月, ITU-T 公布了甚低码率视频压缩的方案 H.263 建议[1] (Video Coding for Low Bit-rate Communication), 该方案具有较高的压缩比和较高的图像质量, 旨在发展 PSTN 和移动通信网上的可视电话业务。测试模型表明, 在相同的信道条件下 H.263 的 PSNR 较 H.261 高 3~4dB。ITU-T 于 1998 年 2 月公布了该建议的最新方案——H.263 Version 2 (即 H.263+), 它在提高编码器总体性能, 尤其在抗信道干扰方面做了许多重要改进, 现已成为低码率多媒体通信终端系统中视频解决方案的标准。H.263 视频编码器的整体框架如图 1, 主体部分是块变换、量化、变长编码和运动估计。

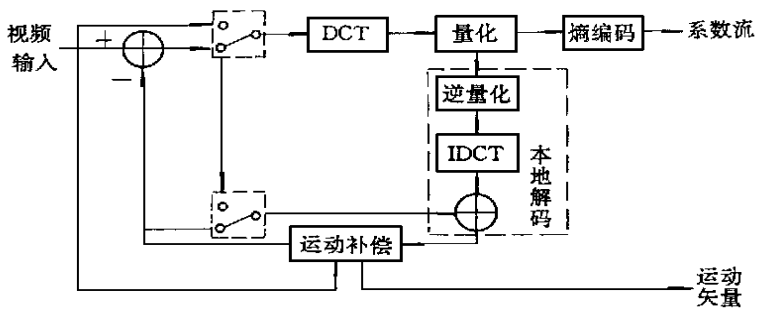

图 1　H.263 编码器构成框图

Fig. 1　H.263 coder structure

Intel 公司分析了大量的通信、语音图像处理等多媒体程序, 总结出它们的特点: 对大量的短数据重复运算, 开发出适合于多媒体应用的多媒体扩展指令集 (MMX) 技术, 可望在同主频的条件下提高多媒体处理能力 50%~800%。MMX 技术和信号处理技术的结合使 PC 机实时复杂信号处理成为可能。

1　H.263 编码器中运动补偿[2]

运动补偿的基本思想是, 由于视频的连续性, 相邻帧之间有很大的相似性, 其中蕴含了大量的冗余信息。对于待编码帧的每一宏块, 在前一帧中搜索与之最相似的宏块 (匹配宏块), 记录下两宏块

* 收稿日期: 2000-01-10
作者简介: 程翥 (1974), 男, 博士生。

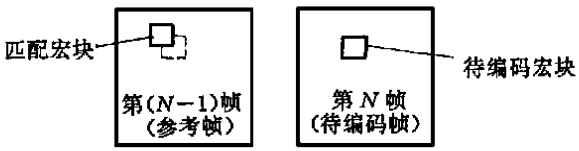

图2 运动补偿中的宏块匹配

Fig 2 The macroblock matching in motion compensation

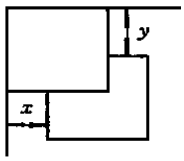

图3 运动矢量的定义

Fig. 3 The definition of motion vector

之间的相对位移，用 (x, y) 表示，称为“运动矢量”(Motion Vector)，如图2、3所示。然后求出两宏块像素的差值，再对该差值进行编码。解码时只需要运动矢量和宏块差值就能恢复出第 N 帧图像。一般情况下，由于待编码宏块与参考宏块差异不大，对其差值进行的DCT、VLC编码效率可以很高，而运动矢量占用的比特在整个码流中的比例是很小的。运动补偿使得编码效率提高了许多，然而其运算量也非常大，主要消耗在匹配宏块的搜索上。运动补偿在整个编码器中所占的耗时比重约为40%，因而在很大程度上决定了编码器的实时性。匹配准则的确立对于运算量和PSNR等参数有着很大影响。从减小匹配误差的角度出发，可选择MSE (Mean Square Error) 准则。

$$\mathrm{MSE}(x,y,k,l)=\frac{1}{N^2}\times\sum_{i,j=0}^{N-1}(F_{-1}(i+x,j+y)-F_0(i+k,j+l))^2 \qquad (1)$$

其中：F_0 和 F_{-1} 分别代表当前帧和重构帧 (参考帧)；k，l 为待编码宏块在当前帧中的坐标；x，y 为重构帧中参考宏块的坐标；N 表示宏块的尺寸，此处为16。对于QCIF格式的图像，对所有的宏块 (允许交叠) 进行匹配运算，在单指令单数据 (SISD) 的CPU上，仅完成一帧内所有宏块的匹配就需要 5.26×10^8 次操作。为了降低运算复杂度，ITU一方面要求采用恰当的搜索策略，一方面提出了MSE的一种替代准则——SAD (Sum of Absolute Difference)。与MSE准则相比，它将像素差值的平方运算简化为绝对值运算，显著减少了运算时的操作数。SAD的定义如 (2) 式。

$$\mathrm{SAD}(x,y,k,l)=\sum_{i,j=0}^{N-1}\left|F_{-1}(i+x,j+y)-F_0(i+k,j+l)\right| \qquad (2)$$

式中符号意义同 (1) 式。SAD实质上是在匹配精度与运算复杂度之间的一种折衷，目前不少国际标准 (建议)，如MPEG-1，H.261和H.263均采用此准则。SAD减少运算操作次数的依据是：要表示一个8位的数的平方，必须使用一个16位的寄存器，而且在大多数的CPU (DSP和MMX Pentium CPU除外) 中，乘积的高部和低部分别放在不同的寄存器中，对乘积求和时会增加很多操作，然而大多数的CPU都提供取绝对值的指令，取绝对值不会引起存储器的扩展，一般而言，SAD的计算量要比MSE的计算量减少1/3。我们对100帧分别采用了SAD和MSE的图像质量进行了比较，证明采用SAD的图像与采用MSE的图像相差不超过0.1dB。

2 MMX技术

针对多媒体信号处理的特点，MMX采用了单指令多数据 (SIMD) 技术，即在单个指令执行过程中，同时操作多个数据。通过MMX技术Intel解决了宽数据总线操作短数据时的带宽浪费，可提高系统效率1至8倍[3,4]。为了同时操作多个短数据，把短数据放在一个64位的寄存器中，MMX定义了四种64位数据格式：它们是由8个字节构成的字节组，由四个16位的字构成的字组，由两个32位双字构成的双字组，由一个64位的四字构成的四字，统称数据组数据类型。

为了操作这些新的数据类型，Intel定义了8个64位的MMX寄存器，这些寄存器只能由MMX指令操作，同时这些寄存器又是浮点寄存器的别名，操作系统运用MMX寄存器如同操作浮点寄存器一样，在任务切换，保护现场时，MMX寄存器可以获得和浮点寄存器一样的保护，从而保证了MMX技术出现之前的操作系统对MMX技术的兼容。MMX的寄存器可操作64位的数据类型。MMX指令使用

寄存器名 MM0 到 MM7 直接访问寄存器。MMX 寄存器仅能被应用于数值处理。

Intel 设计了 57 条 MMX 指令，通过 MMX 寄存器以并行的方式操作全部的成组数据类型中的各数据元素。MMX 指令可以支持有符号和无符号数据元素的操作。特别的是为了适合数据处理的要求，MMX 技术首次在通用 CPU 中引入了溢出的饱和处理方式。饱和处理方式是指，在运算结果出现上溢或下溢后，在寄存器中保持极端值，防止环绕式处理中的符号逆转，减少有限字长的影响。是否采用饱和处理方式可由指令指定。

MMX 指令集包括：数据转移指令，算术运算指令，比较运算指令，转换运算指令，逻辑运算指令，移位运算指令，MMX 状态置空指令。MMX 的指令能够对被操作数据组各元素独立进行相同的操作，在饱和方式下的加减，可以根据指令指定数据有无符号而采用不同的饱和处理方式。特别的是数据组的乘法可以根据用户的需要取出 32 位积高 16 位或低 16 位或进行乘加。

MMX 技术充分利用了 CPU 的 64 位数据操作能力，可以将程序效率提高 50% ~ 400%。但使用 64 位的数据转移和数值运算必需考虑数据对齐的问题[3,5]。在 pentium 处理器上，对一个在高速缓存或总线上进行未对齐的数据访问将至少多耗费 3 个时钟周期，为了达到较高处理效率，8 字节数据必须以 8 字节为边界对齐。常量数据和较少的数据对齐通过复制原始数据来进行比较合适。对于较长的数据可以先按对齐方式把数据读入寄存器，再在寄存器中进行拼接，或者先将未对齐的首部处理掉，以后的数据按对齐方式进行处理。在多组数据进行运算时，尽可能选择小数据或固定数据用复制的方法进行对齐。

3 MMX 技术加速 H.263 编码器

H.263 编码器中的运动补偿是算法中运算量最大的模块。考察它的相近准则 (1)、(2) 可以看出，MSE 和 SAD 的计算主要是短数据的重复运算，这种数据运算形式比较适合使用专门为多媒体程序开发的 SIMD MMX 指令实现，充分发挥 Pentium 处理器的 64 位数据传送和运算性能，提高运算速度。但是 MMX 技术的提速只是增加了系统的单个指令操作数据的数目，不能提高单个指令的执行速度。为了适应 MMX 技术，必须做一些辅助工作，保证每个 MMX 指令在满足对齐规则的条件下操作尽可能多的数据。

MMX 的数据都应该以 64 位数据组形式存放，根据 Intel 的对齐规则，64 位数据必须以 8 字节为单位对齐，而运动补偿的计算中任何一个数据 (除部分边界点) 都可能成为首字节，因此所有字节都必须放在能够被 8 整除的地址上，要充分利用 MMX 技术的性能必须将采样数据在内存复制八次，如果用指令实现数据的搬移，则需要大量的指令和时间才能完成数据搬移，这个搬移的时间几乎可以与采用 MMX 技术得到的时间节省相当，采用指令搬移数据是不行的。本文采用了直接内存存取 (DMA) 技术利用 CPU 的空闲机器周期完成了数据的复制。在 DMA 完成后进行运动补偿的运算。以下讨论总假设数据是对齐的。

MMX 技术没有提供数据组的绝对值指令，求绝对值只能通过比较、取反等指令完成。首先将 mm1 清零，把输入数据与 mm1 比较，在 mm1 中对应输入数据中大于零的元素的位置置 00h，小于零的置 FFh，通过输入数据与 mm1 异或将输入数据中的负数取反，再减去 mm1，使负数的反加 1，完成负数的补码取反。加上求和一步，完成 MMX 数据组取绝对值求和需五步方可完成，而数据组乘加只需一步，MSE 在 MMX 上实现的效率似乎比 SAD 在 MMX 上实现效率要高。MMX 只提供了字组 (16 位) 的乘和乘加指令，要取节组的平方和应该将字节组拆成两个字组，再分别乘加，这样平方求和也是五步。在指令层次上 MSE 和 SAD 的指令数是相当的。

SAD 算法的具体实现：对于特定的 k，l，x，y，首先计算他们在内存中合适与数据对齐的对应地址，在 ESI，EDI 中装入合适的指针地址，在 ECX 中装入 8，mm3 中装入 0。

```
SAD:    movq mm0, qword ptr [esi+ ecx* 8- 8]    ; 装入 F_{-1}(i+x, j+y)
        movq mm1, qword ptr [edi+ ecx* 8- 8]    ; 装入 F_0(i+x, j+y)
        psubb mm0, mm1                          ; F_{-1}(i+x, j+y) - F_0(i+x, j+y)
```

```
pxor mm1, mm1                    ; 取绝对值 step1
pcmpgtb mm1, mm0                 ; 取绝对值 step2  比较
pxor mm0, mm1                    ; 取绝对值 step3  负数取反
psubb mm0, mm1                   ; 取绝对值 step4    负数+ 1
paddusb mm3, mm0                 ; 饱和处理的累加
loop SAD
```

最后输出结果在 mm3 中。在此计算中 SAD 计算一共用去了 9 × 8= 72 个指令周期，相比不使用 MMX 技术的算法需要 64 × 6= 384 个周期提速 5 倍有余。MSE 的实现与 SAD 实现相当，不再赘述。

4 效果

考虑到 Pentium 系列处理器有两条并行执行管道，消除相邻指令对同一寄存器的操作则可将运算速度再次提高一倍[3]。对于 SAD 算法，程序运行速度又可以提高一倍，对于 MSE 算法，因为 MMX Pentium 只提供一个 MMX 乘法器，乘加指令不能在两个执行管道并行执行，消除相邻指令对同一寄存器的操作对 MSE 的速度提高较为有限。因此采用 MMX 技术对整个编码器速度提高见表 1，对图像质量的影响如图 4，MMX 实现和非 MMX 实现稍有差别是因为处理流程不一样，有限字长效应的影响不同。经过 MMX 优化，运动补偿的计算量已经不是 H. 263 的主要部分。

表 1 运算速度比较

Tab. 1 Speed comparison

测试序列	MMX™ SAD (s)	MMX™ MSE (s)	Non-MMX™ (s)
Miss America (83 帧)	4. 52	5. 37	8. 13
Carphone (280 帧)	17. 91	21. 06	29. 17
Grandma (800 帧)	43. 44	52. 72	75. 41

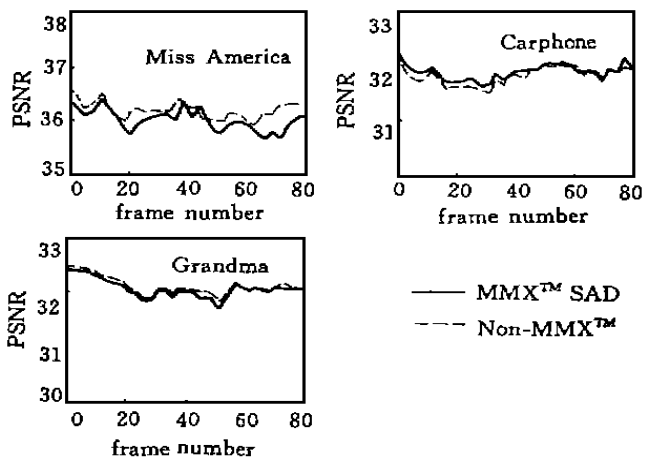

图 4 MMX 和非 MMX 图像质量比较

Fig. 4 PSNR comparison between MMX and non-MMX

参考文献:

[1] ITU-T Recommendation H. 263 (1996), Video coding for low bit rate communication [S].

[2] Zeng B, Li R X, Liou M L. Optimization of Fast Block Motion Estimation Algorithms [J]. IEEE Trans. Circuits and Systems for video tech., December 1997, 7 (6): 833-844.

[3] Intel Architecture Software Developer Manual [R]. Volume 2: Instruction Set Reference Intel, 1997.

[4] 程翥, 易波. 多媒体扩展指令集语音信号处理中的应用 [A]. 全国多媒体会议, 1998.