基于片上存储器的视频编码优化

温淑鸿[1,2], 崔慧娟[1], 唐 昆[1]
(1. 清华大学电子工程系, 北京 100084; 2. 中国传媒大学信息工程学院, 北京 100024)

摘 要: 为了提高视频编码器性能, 并降低功耗, 本文提出了一种缩短数据生命期减小存储器需求的方法. 分阶段计算中间结果, 减少临时结果存储, 可减小存储器需求和存储器访问次数. 另外提出了一种有效利用 CPU 寄存器字宽和桶型寄存器移位能力, 缩减变字长编码输出运算复杂性的方法. 实验结果表明, 该方法能显著缩短视频编码中变字长编码的时间.

关键词: 存储器; 视频编码; 变字长编码
中图分类号: TP343 文献标识码: A 文章编号: 0372-2112 (2005) 12-2246-04

Video Encoder Optimization Based on Scratch-pad Memory

WEN Shu-hong[1,2], CUI Hui-juan[1], TANG Kun[1]
(1. *Depart. of Electronic Engineering, Tsinghua University, Beijing* 100084, *China*;
2. *Information Engineering School, Communication University of China, Beijing* 100084, *China*)

Abstract: To improve video coder performance and reduce power consumption, a method is proposed to shorten data lifetimes and decrease memory requirement. Memory needs and memory access times are reduced by computing intermediate results by stages and decreasing temporary results storage. We propose an algorithm to reduce the complexity of variable length code output by exploiting register width and shift ability of barrel shifter. Experimental results show that it can lessen the time of variable length code output in video encoders.

Key words: memory; video coder; variable length code

1 引言

CPU 运算速度迅速提高, 外部数据存储器的访问能力相对较低, 为了匹配 CPU 和外部存储器的速度差异, 通常采用片上 Cache 或者片上存储器. 片上 Cache 采用硬件控制数据的存取, 片上存储器的使用则完全受软件的控制. Cache 结构由于采用更多的硬件, 因而具有更大的功耗[1]. 低功耗是嵌入式设备的基本要求, 因而在嵌入式处理器中, 片上 RAM 可能作为 Cache 的替代. 目前主要的半导体厂商都生产包含片上存储器的高性能处理器, 尤其是用于手持设备的嵌入式低功耗芯片, 如 TI 公司的 OMAP 系列媒体处理器[2]. 多媒体应用的数据访问具有较大的数据集和较少的数据再利用, 而应用程序的执行时间主要消耗在一些较小的循环上, 目前针对片上数据存贮器结构的视频算法优化却很少研究. 本文试图讨论针对这类处理器的视频编码器优化, 以改善实时性能, 同时尽可能降低片外存储器访问次数, 以便降低算法的功耗.

2 减少存储器需求

通过减小存储器需求, 可以缩减存储在片外存储器的数据数量, 从而缩减片外存储器访问次数, 降低功耗. 为了减小存储器需求, 数据在其生命期一结束, 就立即释放其所占的存储器, 并用来存储其它数据. 生命期不相交的变量或者矩阵可以共用相同的存储器, 从而缩减存储器需求. 数据的生命期定义为从初始化或者产生(CPU 写) 到不再被 CPU 读取进行处理间隔. 称 CPU 采用与 i 无关的值改写 i 的过程为变量 i 初始化. 如"$i=0$"为 i 的初始化, "$i=i+1$"或者"$i=f(i)$"称为对 i 的修改. 为了缩短数据的生命期, 调整数据的初始化或者改写操作到 CPU 读取该数据的前面, 可减小数据的生命期. 图 1 中 i 和 j 的生命期有重叠, 图 2

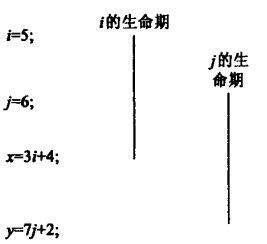

图 1 原始程序

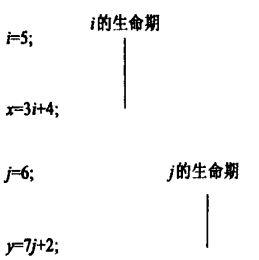

图 2 调整数据的初始化顺序

收稿日期: 2004-09-30; 修回日期: 2005-06-06
基金项目: 国家自然科学基金(No. 60272020)

中调整 i 变量的初始化到 CPU 读取 i 的前面, 缩短了变量的生命期, 从而变量 i 和 j 的生命期不相交. 将不相交的变量共用存储器, 减少了对 j 变量的存储器需求, 结果见图 3.

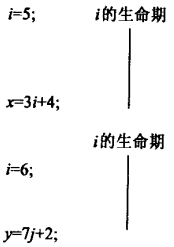

图 3　共用存储器

在视频编码中运动搜索采用单一参考帧情况下, 通常的方法采用两个帧存, 一帧存储当前编码帧重建图像, 一帧用来存储前一帧重建图象, 用作当前帧的预测参考. 当运动搜索范围为 $[-16, 16]$ 时, 也就是说在编码第 j 个 GOB 时, 搜索窗位于前一帧重建图像的第 $j-1$、j、$j+1$ 个 GOB, 前一帧重建图像里前 $j-2$ 个 GOB 已经不再用作搜索参考图像, 生命期结束可尽快释放, 用来存储新编码宏块的重建图像. 因此存放重建图像的存储器大小为: 当前编码帧的 j 个GOB 加上前一帧 $M-(j-2)$ 个 GOB, 总计为 $M+2$ 个GOB, 其中 M 为一帧图像包含的 GOB 数量, 重建图像减少了 $(M-2)$ 个 GOB.

2.1　分阶段计算中间结果

数据处理算法总是处理一些数据, 得到一些结果, 程序的执行表现为周期性读入原始数据并计算得到最终结果. 在图像编码算法中, 中间结果的存储需要大量的存储器. 一次计算很多中间数据, 每次计算最终结果只需要利用其中部分数据, 这相当于延长了数据的生命期. 每个数据处理任务的输出数据用作相邻的数据处理任务的输入, 可以缩减中间结果的的生命期. 分阶段地计算中间结果, 可显著减小存储中间结果的存储器需求. 在 H.263 编码器中, 除了输入图像数据和最后输出的 H263 码流外, 其它为计算最后输出 CPU 计算的大量数量可以统称为中间结果. 在 INTER 帧编码过程中, 为了在整像素运动搜索以后, 进行半像素运动搜索, 必需对参考图像进行半像素内插. 通常先对整帧图像进行半像素内插, 内插结果为一帧中所有的宏块编码所利用, 半像素图像为原始图像的 4 倍, 存储半像素内插结果需要大量的存储器. 单个宏块的码流只用到少数半像素图像, 因而在编码前只计算编码该宏块所必需的半像素, 存储器需求将显著减少. 半像素搜索是在整像素搜索后以整像素运动矢量为中心的搜索, 为减小中间结果的生命期, 半像素内插这一处理任务应与半像素搜索相邻, 位于整像素搜索和半像素搜索的中间. 因而可以在半像素搜索前对整像素运动矢量指向的 18× 18 的块进行内插, 存储器需求大约为 33× 33.

2.2　减少中间结果存储

减少中间结果不必要的存储, 不仅可以减少存储器需求, 而且可以减少存储器的访问. 在图 4 的 ***C*** 代码片段中, 存在三个矩阵 ***A***、***B***、***C***, 其中 ***B*** 矩阵元素与 ***A*** 矩阵元素一一对应, ***C*** 矩阵元素与 ***B*** 矩阵元素一一对应, 因而 ***C*** 矩阵元素与 ***A*** 矩阵元素一一对应. $f1, f2$ 为矩阵基元值的映射. 省去中间结果存储后计算方法见图 5 和图 6, 中间结果 $B[i]$ 并不写入存储器, 直接由保存在寄存器的 $B[i]$ 计算 $C[i]$, 从而节省存储了存储 ***B*** 矩阵和读取 ***B*** 矩阵的操作.

```
for( i= 0; i< N; i++ )   B[i]= f1(A[g1[i]]);
for( i= 0; i< N; i++ )   C[i]= f2( B[g2[i]]);
```
图 4　两个循环

```
for( i = 0; i< N; i++ )   C[i]= f2(f1(A[g1[g2[i]]]));
```
图 5　按照结果索引递增计算

```
for( i= 0; i< N; i++ )   C[g2^-1[g1^-1[i]]]= f2(f1(A[i]);
```
图 6　按照源索引递增计算

在 H.263 编码器中, INTRA 宏块编码过程包括离散余弦变换(DCT) 、量化、扫描输出码字、逆扫描、逆量化、逆 DCT 等步骤. 但 DCT 和逆 DCT 变换中的每个结果矩阵元素是源矩阵中的所有元素的函数, 变换前后矩阵的元素不是一一映射关系. 而量化、逆量化、扫描、逆扫描中结果矩阵的每个元素只与源矩阵中唯一的元素对应, 是一一映射, 每种操作都对应一个循环, 将这些循环合并, 可以省掉中间结果的存储.

3　数据处理任务

算术运算和逻辑运算通常由 CPU 或协处理器来完成, 若 CPU 访问数据不会引起延迟, 则程序的实时性能主要取决于 CPU 的处理能力. 因而将应用程序按照功能分成 CPU 顺序执行的若干数据处理任务[3], 应用程序表现为数据处理任务的循环执行和条件执行等形式. 这样划分主要是为了分析数据处理过程中的存储器需求. 多媒体应用总是需要处理大量数据, 每个数据处理任务处理一个或者多个输入数据块, 并得到一个或多个输出数据. 某个数据处理任务的输入可能来自于前面某个任务的输出, 也可能是程序的原始数据输入. 某个数据处理任务的输出可能作为后面某个任务的输入, 也可能是该程序的目标输出. 数据处理任务是数据处理的基本单元, 一旦开始执行, 就不必停下来等待数据转移. 数据处理任务具有以下两方面的性质: 一是输入数据必需包含成块的数据(矩阵) , 若一段代码仅仅处理了几个标量数据就不列入数据处理任务的范畴. 当然数据处理后必然会有计算结果输出, 但输出数据可以不包含成块数据(矩阵) , 输出数据是一个或者几个标量数据是可以的. 二是数据处理的连续性. 由于片外存储器的访问速度远远低于 CPU, 因而数据处理任务执行过程中, CPU 不能直接读写片外存储器, 也不能停下来等待直接存储器存取(DMA) 转移数据后再读写片上存储器. 数据处理任务表现为 CPU 不间断地处理某个数据块.

3.1　精细同步减小片上缓冲区需求

DMA 控制器的转移能力直接影响数据输入输出缓冲区所需的存储器数量, 从而影响存储器的分配. 若 DMA 转移和 CPU 保持精细的同步, 可以有效减小片上存储器缓冲区. 称存储输入片外数据的片上存储器为输入缓冲区, 存储需要转移到片外的输出数据的片上存储器为输出缓冲区. 假设某个处理任务处理输入缓冲区数据, 在该处理任务前的一个或多个处理任务并不访问该缓冲区, 并且 DMA 也可利用这些间隔转移数据到输入缓冲区, 这时并不需要设置乒乓缓冲区. 同样, 某数据处理任务产生输出结果, 其后的一个或多个处理任务

并不改写该输出缓冲区, DMA 也可利用这些间隔转移输出缓冲区数据到片外, 从而减小片上存储器缓冲区需求.

3.2 链表能力减小缓冲区需求

当 DMA 转移控制器的转移同步事件可以为其它 DMA 通道的转移结束事件(称之为 DMA 控制器的链表能力)时, 就可以同时初始化两个 DMA 转移上下文, 用一个通道的转移结束事件去触发另外一个通道的转移, 当一个转移的源地址是另外一个转移的目的地址时, 仍能保证两个数据转移任务的正确结果, 由于共用了相同的存储器, 所以减小了存储器需求. 两个数据块的转移具有确定的先后顺序, 后一个转移的源地址为前一转移的目的地址, 或者后一个转移的目的地址为前一个转移的源地址时, 当 DMA 控制器不具有链表能力时, 这两个数据转移任务的上下文就不能同时初始化.

4 利用数据局域性

文[4]提出了片外存储器与片上存储器之间的数据转移成本模型, 并以此模型为基础, 讨论了矩阵分集的方法. 多媒体数据的处理可以分成一些基本数据对象, 处理基本数据对象后, 会得到与基本数据对象相对应的最后输出, 计算过程中可能会利用已处理数据对象的输出. 将对应基本数据对象的处理时间可以称为一个一级时间片, 每个一级时间片根据需要可进一步分成若干个二级时间片, 每个二级时间片对应于数据处理任务. 每个同级的时间片并不代表时间的绝对相等, 而只代表这个时间片中含有类似的数据处理任务或者相同数量的数据处理任务. H.263 编码器和 H.264 编码器中, 每编码一个 16×16 的宏块, 就能产生对应该宏块的最终输出码流, 因而把宏块看成是 H.263 和 H.264 编码器中的基本数据对象, 称宏块的处理为一级时间片. 占用较多存储器的是前面数据对象的处理输出将要用作后面处理输入的矩阵数据. 对各个基本数据单元的处理, 经常会出现不同程度的数据重复利用, 这通常表现为对相邻几个数据单元的处理存在相同的输入数据, 或者一个单元的输出用作另一个单元的输入. 对于图像而言, 相邻的数据单元包括水平方向或者垂直方向的相邻. 规定基本数据对象处理的时间间隔为 1, 该次数据被 CPU 读入与相邻前一次 CPU 读写该数据的间隔称为该数据的间隔. 例如在 H.263 编码器中, 重建宏块(x, y)用作下一帧中以该宏块为中心的 9 个宏块的运动搜索参考图像, 假设图像大小为 QCIF, 该重建宏块的 9 次访问间隔分别为 87、1、1、9、1、1、9、1、1. 需要采用 DMA 转移该数据的时刻为访问间隔较大的时刻, 如在第一次读访问前(间隔为 87), 或者在第 1、4、7 次读访问前(时间间隔分别为 87、9、9). 若仅在第一次读访问前转移该宏块数据到片上, 该宏块数据仅需从片外到片上转移 1 次, 需要片上存储器至少(1+ 1+ 1+ 9+ 1+ 1+ 9+ 1+ 1)共 25 个宏块; 若在第 1、4、7 次读访问前前转移, 需要转移 3 次, 存储器至少需要(1+ 1+ 1)共 3 个宏块. 若全部重建图像存储在片上, 不需要转移, 至少需要存储器(87+ 1+ 1+ 9 + 1+ 1+ 9+ 1+ 1)共计 111 个宏块. 实际应用中, 算法对访问数据格式有一定的要求, 需要的存储器比上面结果稍大一些. 这需要 CPU 硬件特点及指令寻址方式作具体分析. 上述问题也可这样分析: INTER 帧编码中相邻两个宏块(水平相邻或者垂直相邻)的搜索窗仅有三个宏块不同, 若仅分配 9 个宏块片上存储器用于存储参考图像, 编码每个宏块至少需要新转移三个宏块重建图像到片上, 每个重建宏块至少需要 3 次片外到片上转移. 若分配三个 GOB, 能将已经转移到片上的重建宏块保存在片上直到它不再使用为止, 这时每编码一个宏块只需新转移一个宏块到片上, 大大缩减重建图像从片外转移到片外的次数.

5 变字长编码输出

变字长编码(VLC)是一种广泛应用的熵编码方法, 对于大概率事件采用较短的码字编码, 小概率事件采用较长的码字编码, 以使得平均码长最短. 每帧图像的编码比特数一般要求为整数个字节, 若整帧图像的有效信息不是 8 比特的整数倍, 最后添加一定的冗余比特, 构成 8 比特的整数倍. 目前代码中普遍使用的 VLC 码流输出方法是每次处理 1 个比特, 每达到 8 比特就写入存储器, 需要较多的条件判断和存储操作. 这种方法没有充分利用 CPU 移位寄存器一次移位操作可左移或者右移多个比特这一特点.

现在 CPU 寄存器普遍是 32 比特, 甚至 64 比特(如 AMD Athlon 4), 现在的码流输出方法也没有充分利用 CPU 寄存器字宽. 改进的算法流程见图 7, 假设 CPU 为 big endian 工作模式, 寄存器字宽是 N, N 为 2 的幂, VLC 码字的最大长度 n 不超过 CPU 寄存器字宽 N. 为了充分利用移位寄存器的移位能力, 每次写存储器的单位为 N 比特, 则未能输出到码流中的比特数 nLeft 不大于 $N-1$ 比特, 输出新码字时, 总的信息长度 $n+$ nLeft 不大于 $2N-1$, 可分成两种情况: $n+$ nLeft 大于等于 N 或者 $n+$ nLeft 小于 N.

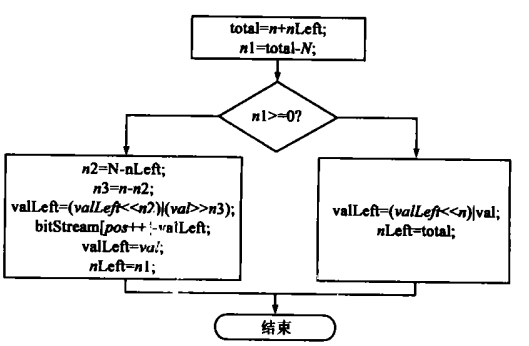

图 7 改进 VLC 输出

当 $n+$ nLeft 大于等于 N 时(图 7 中左边分支), 码流输出见图 8, 首先将剩余信息(a)左移到寄存器的最左边(c), 再将

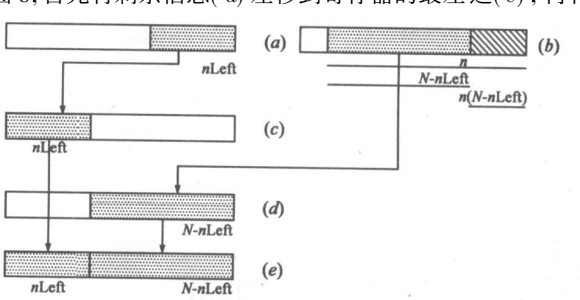

图 8 改进算法中 $n+n$Left>=N 分支

当前码字(*b*) 右移到(*c*) 的后面(*d*), (*c*) 和(*d*) 占据了整个寄存器, 将寄存器内容写入存储器. 这次编码未写入存储器的信息位于新的码字中. 当 $n+ n\mathrm{Left}$ 小于 N 时(图 2 中右边分支), 码流输出见图 9, 将剩余信息(*a*) 左移 n 比特到当前码字的左边(*c*), 再与当前码字或, 得到最后的剩余信息(*d*) . 图 3 是原始算法和改进算法的运算复杂性对比. 只要平均码长大于 2 比特, 改进方法的运算次数都少于原始算法.

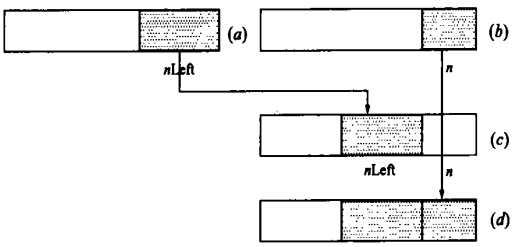

图 9　改进算法中 $n+n\mathrm{Left}<N$ 分支

		条件判断	运算
原始算法		3n	4n
改进算法(最大码长小于 N)	分支 1	1	7
	分支 2	1	4

图 10　算法复杂性对比

6　实验结果

实验: 为了验证本文提出的 VLC 码流输出算法的有效性, 测试了 H263 编码器算法 TMN30 的码流输出原始算法与建议算法所需时间对比. 测试序列为 miss. qcf, 编码 6 帧, 编码模式为 IPPP…, 量化参数为 8, 调用次数 4584 次. 在 intel Pentium 4 平台上原始算法和改进算法所花时间分别为 0. 350 毫秒和 0. 109 毫秒, 时间缩短了 68%. 在 TI OMAP1510 中 C55x 处理器上, 原始算法和改进算法所消耗 CPU 周期分别为 1776104 毫秒和 285442 毫秒, 时间缩短了 83%.

7　结论

本文提出的减少存储器需求的方法已经成功应用在 OMAP 平台上 H263 优化中. 在 H. 263 编码器中, 运动估计等模块采用了快速算法及优化后, 变字长编码输出由于有较多的条件判断, 数字信号处理器中的条件操作需要消耗更多的周期, 因而减少变字长编码输出的时间对于运算能力有限的移动媒体处理器仍是十分重要的.

参考文献:

[1] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M Balakrishnan, Peter Marwedel. Scratchpad memory: a design alternative for cache on-chip memory in embedded systems[A]. 10th International Workshop on Hardware/ Software Co-Design (CODES' 02) [C]. New York, NY, USA: ACM press, 2002. 73- 78.

[2] Jamil Chaoui, Ken Cyr, Sebastien de Gregorio, et al. Open multimedia application platform: enable multimedia applications in third generation wireless terminals through a combined RISC/ DSP architecture[A]. 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2001 (ICASSP' 01) [C]. Washington, DC, USA: IEEE Computer Society, 1997. 1009- 1012.

[3] Anand Ramachandran, Margarida F Jacome. Xtream-fit: an energy-delay efficient data memory subsystem for embedded media processing[A]. Proceedings of the 40th conference on Design automation[C]. New York, NY, USA: ACM press, 2003. 137- 142.

[4] Kandemir M, Ramanujam J, Irwin M, et al. Dynamic management of scratch-pad memory space[A]. Proceedings of the 38th conference on Design automation[C]. New York, NY, USA: ACM press, 2001. 690- 695.

作者简介:

温淑鸿　男, 1969 年生于湖北钟祥, 清华大学电子工程系博士研究生, 中国传媒大学信息工程学院教师, 主要研究方向为图像编码、嵌入式系统、嵌入式多媒体处理器结构. E-mail: wenshuhong@ sohu. com.

崔慧娟　女, 1945 年 11 月生, 清华大学电子工程系教授, 主要专业及研究方向: 语音编码、图像编码、多媒体终端等.

E-mail: cuihj@ ee. tsinghua. edu. cn.

唐　昆　男, 1945 年 10 月生, 教授, 博士生导师, 主要研究方向为语音编码、多媒体终端、多媒体通信等.

E-mail: tangkun@ tsinghua. edu. cn.