一种面向矢量视频处理器的并行存储结构

朱 笛 申功勋

(北京航空航天大学 宇航学院, 北京 100191)

摘 要: 视频编解码算法不仅复杂度非常高,对数据存储需要密集的矢量化访问.针对这一问题提出了一种线性斜移结构的二维存储方案,支持任意行或任意列的单指令存取,与SIMD(Simple Instruction Multiple Data)结构的矢量处理器相结合,能够成倍的加速视频计算的执行效率.给出了该方案的地址计算逻辑和存储结构,精简的地址计算逻辑和数据重排逻辑构成了视频处理器的矢量地址产生单元,解决了数据重排的高延迟和高复杂度问题.结合H.264视频编码标准对这一结构的性能进行了分析和对比,对该编码标准中的运动估计,分像素插值和去块效应滤波这3个计算复杂度最高的模块在该结构下的执行效率进行了分析和对比.

关 键 词: 视频编码; 存储结构; 超大规模集成电路

中图分类号: TP 303

文献标识码: A **文 章 编 号**: 1001-5965(2008)10-1177-05

2-D parallel memory architecture for video processor

Zhu Di Shen Gongxun

(School of Astronautics, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: Video codec has a very high computation complexity and features intensive vector accesses to memory. A 2-D parallel memory scheme based on linear skewing scheme was proposed. The memory scheme can be combined with simple instruction multiple data (SIMD) vector processor to address the computation challenge of video. The address generation logic and scheme's micro architecture were analyzed. The scheme uses a data rotation unit to permute data element to form a vector. This method simplified data permutation network which is the bottleneck of most parallel memory schemes. The performance difference between parallel memory and traditional memory were also compared. The kernel module of H.264/AVC such as motion estimation, de-blocking filter and interpolation were investigated based on the proposed memory scheme.

Key words: video coding; memory architecture; very large-scale integration (VLSI)

随着多媒体技术的广泛应用,要求DSP(Digital Signal Processing),嵌入式处理器或通用处理器执行更多的视频处理计算,比如MPEG-4, H.264, AVS等.这些视频处理不仅计算复杂度极高,而且需要更高的数据存取带宽和更加灵活的存储访问形式.传统的嵌入式处理器或DSP的哈佛总线存储结构具有诸多弊端,比如有限的1～2个存储器bank,一次只能访问1～2个字,地址必须与字的边界对齐,无法进行矢量存取等等.这些限制降低了处理器的对视频信号的处理效能.常用的处理器必须以非常高的时钟(功耗)才能执行这些越来越复杂的视频计算.

由于视频数据的采样都是8 bits的像素,因而在执行这些计算的过程中,处理器操作数绝大部分仅需8 bits字长,即一个字节.而传统DSP处理器操作数一般都是按字存储的,比如16位, 32位,自然计算也是以字为单位的,因此无法在一个指令周期内处理多个像素.可是在视频计算复杂度最高的运动估计部分,需要对尽可能多的像素进行SAD(Sum of Absolute Difference)计算,即希望一次对多个像素分别进行运算,同时对像素的访问需要跨越字的边界,即每次所取的数个像素

收稿日期: 2007-11-12

作者简介: 朱 笛(1977 -),男,山东嘉祥人,博士生, andy_zhudi@yahoo.com.cn.

可能前几个属于存储器中的某一个字,后几个属于临近该字的下一字.这种存储访问形式被称之为具有字节存取能力.

大部分具有多媒体指令处理器都实现了单指令多数据 SIMD(Simple Instruction Multiple Data)并行计算能力,即把操作数中的各个字节作为子操作数在同一指令周期内分别处理,这一技术也称为子字并行(subword parallel).这些处理器包括当今流行的 Intel Pentium MMX,Wireless MMX,另外流行的 DSP处理器象 TI C6X,Analog Device BlackFin[1],CEVA teaklite[2]也有这样的指令.然而对于跨越字边界的字节访问,这些流行的处理器并未直接实现,因此这些处理器都需要额外的指令开销来对像素在寄存器中的排列进行移位,合并,对齐等操作.当然,一些更新的处理器如 Intel P III SSE,AD I tigersharc, ARM V6等设计了专门的 align指令对此进行弥补,以减少这些额外的开销,但都没有直接解决这一存储访问的瓶颈问题,所以都未能达到和边界对齐的访问操作一样的指令执行效率.另外,对于图像处理中常用的列向量格式的访问,传统的一维存储结构就更加低效了.

1 并行存储处理器结构和应用背景

新一代的视频图像编码标准包括 H.264/AVC[3],中国的 AVS,微软的 VC-1.这些标准的算法流程都大致相似,其中计算复杂度最高的模块也相同,比如运动估计,整数 DCT变换,帧内预测,去块效应滤波等.其中尤其以运动估计和去块效应滤波在这些新标准中占据了最多处理器时间.传统的 DSP处理器由于存储结构的限制,在执行这些计算时非常低效.原因一方面是传统处理器是一种一维的线性存储结构,而图像处理要支持对图像进行二维的访问,比如对图像的某一列进行读写.另一方面是对于一次存储起访问所获得的多个像素需要进行矢量处理.因此,把并行存储结构和矢量处理器结合的架构对于视频处理是最有效的.

本文提出的并行存储结构是应用于如图 1所示的 SIMD矢量处理器的,图中,PCU为程序控制单元,同时也执行指令译码,ALU是一个标量处理单元,与传统的 RISC算术运算单元相似,它还拥有独立的数据存储器,用于标量数据存储.VPU是一个矢量处理单元,每次存储器访问可以获得两个 8个像素的 64 bits矢量,即 128 bits/Cycle.VPU包含 8个 8位运算单元,可以对 8个像素分别处理.VPU通过两个 64位的数据总线与 X存储器和 Y存储器相连.X存储器和 Y存储器各自拥有独立的 8个存储单元(bank),各单元都有独立的地址产生单元 AGU.由于每个指令周期可以处理高达 8个或 16个像素,VPU因此拥有巨大的数据吞吐能力和并行计算能力.最重要的是存储器不仅可以同时存取同一行的 8个像素,还可以同时存取同一列的 8个像素.

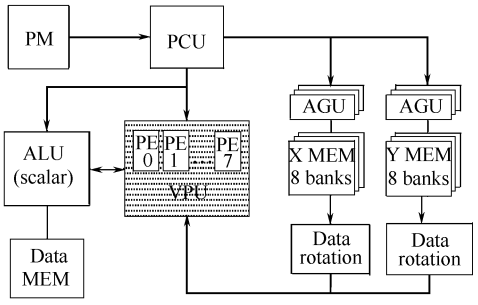

图 1 本文所采用的视频处理器结构

本文用 systemC编写了一个周期精确的视频 RISC处理器模型,并把它与所设计的二维并行存储结构相结合来进行访问和计算效能的分析,该处理器具有图 2所示的 7级流水线结构.

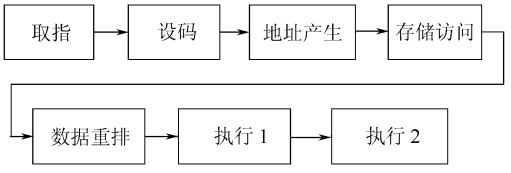

图 2 视频处理器流水线

同步 SRAM利用存储访问把数据驱动到总线上,数据在这一级的终点被锁存到“数据重排”级的寄存器.数据重排(data permutation)流水线产生正确顺序的 64 bits矢量操作数.执行 1和执行 2两级分别进行矢量算术计算.另外,对于普通的标量算术计算指令,流水线会少一个执行级.关于该处理器的微结构细节会另外发文详述.

视频编码中计算复杂度最高的部分是运动估计,运动估计需要将当前要编码的像素块(current block)与参考区域的多个同样大的像素块(reference blocks)进行绝对值差(sum of absolute difference)的计算,寻找结果最小的那个.SAD计算是运动估计的计算核心,正是它的被大量调用,使得运动估计的算法复杂度很高.对于 $N \times N$ 像素块的 SAD计算方法是:

$$SAD(x, y) = \sum_{i=0}^{N}\sum_{j=0}^{N} |cur(i, j) - ref(i + x, j + y)| \quad (1)$$

式中,$cur(i, j)$和 $ref(i + x, j + y)$分别是当前块和

参考块对应位置的像素值,是一个 8位有符号数. 传统的存储方式把参考块和当前块按行做“之”字形扫描,然后线性的存储在一维 64 bits字长存储器中,一个字中会包括 8个像素,当参考块左上角像素是某个字的中间一个字节的时候,就出现像素块与字边界不对齐的情形.

传统存储结构这种弊端造成了运动估计计算十分低效. 由表 1可见,由于传统的按字访问的存储结构,使得在读取一组 (8个)跨越字边界的像素时,需要将相邻的两个字都读入寄存器,再拼合在一起,这样一次 8点的 SAD计算需要 7个周期,而在使用了并行存储结构的矢量处理器中,由于具有 7级流水线,平均仅需一个指令周期即可完成. 运动估计绝大部分的时钟周期都消耗在 SAD计算上,这使得整个运动估计的计算速度快了将近 4倍. 而传统的处理器要使用多达 7个时钟周期来完成一次 SAD计算,数据的重排占据了大量时钟资源. 下表给出了一个典型的传统 SIMD DSP进行 SAD计算时的指令片段.

表 1　传统存储结构 SAD计算

时钟周期	传统 64位哈佛总线结构下的一行 8像素点 SAD
0	LDX　r0, 0, 0 //从参考块 (0, 0)读 8个像素到 r0
1	SHIFT　r0, 8 //左移一个像素 (8位)
2	LDY　r1, 0, 8 //从参考块 (0, 8)读 8个像素到 r1
3	SHIFT　r1, - 56 //右移 7个像素 (56位)
4	OR　r0, r1 //r1 = r0 bit_or r1
5	LDX　r2, 0, 0 //从当前块 (0, 0)读 8个像素到 r2
6	SAD　a0, r1, r2 //计算 r1和 r2中 8个像素的 SAD

而采用了本文提出的并行存储结构之后,上面的例子仅需一个指令 (周期)就可以完成:

$$\mathrm{SAD}\quad x_0,\ x_1,\ y_0,\ y_1,\ a_0$$

指令中 x_0, x_1, y_0, y_1分别指向参考块和当前块的行的起点. SAD计算仅是这两种存储结构性能差异的诸例子之一.

2　一种二维并行存储结构

关于二维并行存储结构的研究已经有多年的历史. 很多研究一直在探讨如何提高并行访问的灵活性,即支持尽可能多的二维形状,比如,在一个二维图像中,能够同时访问一行,一列,或者某个对角线,或者两行,两列等等. 但是,这些支持多个形状的存储结构都需要非常复杂的数据重组网络 (permutation network)[4-6],因为并行存储需要将二维图像中的像素按照一定规则分别存储到多个存储单元 (bank)中,当进行不同形状的存取时,各个单元输出 (入)的像素需要重新排列组合 (permutation)成所需的矢量. 另一方面,这些支持多种形状的并行存储结构还需要更加复杂的地址产生单元,复杂的逻辑实现造成的时间延迟形成了视频处理器的时钟瓶颈,也不利于流水线间的延迟平衡. 因此关于访问形状灵活性的研究已经偏离并行存储在视频处理中的实际应用.

实际上,面向视频处理的存储对灵活性的需求却是有限的. 在视频编码算法中不需要对角线形状的访问,而且对于多行多列的访问可以用单行单列进行代替. 支持任意坐标起始的单行单列的矢量访问是各种最新的视频编码标准的普遍且唯一的需求. 因此,本文提出的这种并行二维存储结构对于视频处理器是高度优化的一种结构. 而且,这种存储结构具有简单的地址计算和简单的数据重组逻辑 (data permutation),因而地址计算流水级的延迟也非常短,如图 3所示,本文提出一种循环式的存储结构来适应于视频处理器的存储需求.

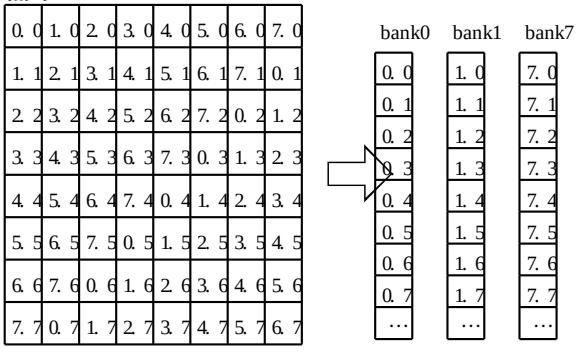

图 3　本文提出的线性循环二维存储方案

图中,每个小方格代表一个 8位的像素,以一个 8×8的像素块为例,存储器被分为 8个单元 (bank0…bank7),每个单元都有独立的地址和 8位数据总线,每个方格中的前一个数字表示该像素所在的存储单元,后一个数字表示该像素在相应的存储单元中 (bank)的地址. 由图可见,二维图像数据中,所有对角线的像素全部存储在同一个存储单元 (bank)中. 对于位于 (i, j)的像素,对应的存储单元为

$$B = (i + j) \bmod 8 \qquad (2)$$

这是一种线性斜移并行存储方案,每行比上一行斜移一个元素. 这种存储方案具有以下特性:

1) 从任意位置开始访问 8个像素或小于 8

个像素的一行,那么这 8个像素都不在同一个存储单元(bank),从而可以无冲突地并行访问;

2) 从任意位置开始访问 8个像素或小于 8个像素的一列,那么这一列中的各个像素都不在相同的存储单元,从而可以无冲突的并行访问从任意位置任取的一行或一列;

3) 不同位置的矢量中各像素所在的存储单元的顺序不同,这一点从图 2中可以看到,从图 2中任取一个行矢量,该矢量第一个元素可能位于 8个 bank任意一个之中.但是不同矢量的存储单元的顺序之间是一种循环移位的关系,利用这种关系可以简化数据重排的逻辑,使用简单的循环移位器(data rotation)就可以实现数据的重排(data permutation).

整个存储访问的结构如图 4所示. I和 J为矢量在二维图像中的起点坐标,坐标以图像左上角为原点.经过 8个 AGU计算出该矢量中的 8个像素在各自存储单元的地址,各存储单元输出的数据,经过循环移位后(data rotation)得到真正顺序排列好的 64 bits矢量.

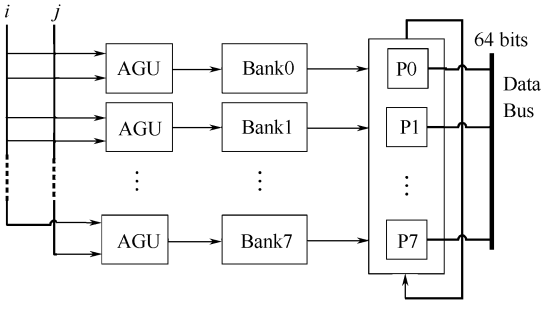

图 4 二维存储结构的实现形式

各个存储器单元都是 8位的双端口同步 SRAM存储器,以 64 ×64的二维图像块为例,那么对于二维图像块中从(i, j)坐标起始的矢量,它包含的 8个像素在 8个 bank中的地址可由下列等式算出:

$a = i\%8 + j\%8;$

bank0_addr = (((8 - a) % 8 + i) > > 3) + (j < < 3);

bank1_addr = (((9 - a) % 8 + i) > > 3) + (j < < 3);

bank2_addr = (((10 - a) % 8 + i) > > 3) + (j < < 3);

bank3_addr = (((11 - a) % 8 + i) > > 3) + (j < < 3);

bank4_addr = (((12 - a) % 8 + i) > > 3) + (j < < 3);

bank5_addr = (((13 - a) % 8 + i) > > 3) + (j < < 3);

bank6_addr = (((14 - a) % 8 + i) > > 3) + (j < < 3);

bank7_addr = (((15 - a) % 8 + i) > > 3) + (j < < 3);

地址产生单元 AGU的计算逻辑见图 5,为了减少地址计算的逻辑时延,这一计算可以采用 ROM查表法实现.每个 AGU存储一个 8 ×8的地址查找表即可.

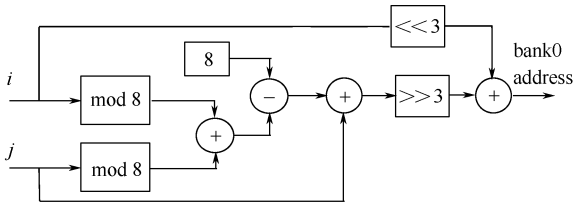

图 5 二维存储结构的地址计算单元逻辑(以 bank0为例)

3 二维并行存储的应用与性能分析

本文使用 SystemC系统建模语言对所设计的存储结构进行了验证和仿真,并创建了一个 SIMD(单指令多数据)视频微处理器模型与这个存储模型一起进行周期精确的性能分析.同时还把 H.264/AVC中最复杂的计算片段写成机器指令在这个仿真平台上运行.得到的性能结果比没有采用二维并行存储结构的 SIMD DSP(ADI BF533[6])改进很多,如表 2所示.

表 2 二维存储结构的对视频处理的性能改进

计算点	Haward总线存储结构下的 MIPS消耗/(百万周期·s^{-1})	二维并行存储结构下的 MIPS消耗/(百万周期·s^{-1})
运动估计	183	45
像素插值	28	5.8
去块效应滤波	55	36

表 2是以 CIF 30fps的 foreman序列作为样本统计的结果,参照对象是 ADI公司的 BlackFin533 SIMD DSP[6],而且其表中数据都是该 DSP上进行汇编优化后的结果.对于这 3种核心计算模块,在使用二维并行存储结构时的计算平台上的优化结果如下.

3.1 运动估计

在运动估计中像素矢量与字边界不对齐的概率是 75%,也就是如表 2所示的情况,传统的按字访问的 DSP需要 6个周期去进行 8点的存储访问和 SAD计算,在另外 25%的概率下像素矢量恰好与字边界对齐,这时需要 3个周期去进行存储访问和计算.这样对于 8点 SAD的运算,未采

用二维存储结构的传统 DSP平均需要: 75 ×6 + 0.25 ×3 = 5.25 cylces而结合了二维存储结构的视频处理器仅需一个周期,无论像素矢量是否与字边界对齐.因而运算执行速度是传统存储结构 DSP的 5.25倍.由于运动估计占了整个视频编码的至少 50%以上的计算量 (比例因算法而异),因而新的存储结构成倍的提高了视频编码的性能.

3.2　像素插值

在进行运动估计的时候需要对半像素和 1/4像素的位置的像素通过插值动态的生成,这些插值计算使用了若干抽头的滤波器,图 6仅以半像素的插值为例:图中 A, B, C, D, E, F以及 D, H, C, I, J, K分别是列方向和行方向的整像素点, x是 C和 I之间的半像素点, y是 C和 D之间的半像素点.这两个半像素通过下面的插值滤波器生成:

$$x = (A - 5 \times B + 20 \times C + 20 \times D - 5 \times E + F + 16) > > 15$$

$$y = (G - 5 \times H + 20 \times C + 20 \times I - 5 \times J + K + 16) > > 15$$

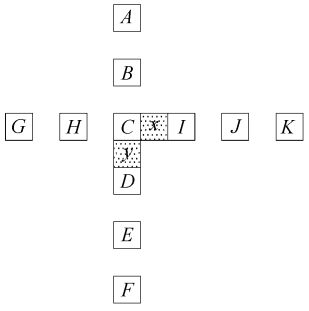

图 6　插值滤波的存储访问格式

这种插值计算如果用传统存储方式的 DSP计算将是非常烦琐和低效,限于篇幅,仅给出对于上面例子的统计结果,见表 3.

表 3　不同存储结构做插值计算时所需时钟周期

计算点	传统 Harvard总线存储结构的 DSP所需周期	采用本文的二维并行存储所需周期
x点插值滤波	19	3
y点插值滤波	25	3

可见采用本文的并行存储结构仅需 3个周期,这是因为, A, B, C, D, E, F 6个像素作为一个列向量仅用一周期载入到矢量单元的操作数寄存器,而传统的存储结构需要将 A, C, B和 D, E, F分别载入,再用多个周期将其移位对齐并放入到矢量单元的操作数寄存器中.在解码器中插值占了计算量的 25%,而在编码器中插值计算占了 6%.因而应用本文的存储结构通过加速插值计算进而高效地加速了视频处理的速度.

3.3　去块效应滤波

去块效应滤波是视频编解码计算量最为庞大的一部分.一方面,需要对 4 ×4块的边界强度进行密集的计算,一方面又需要对块边界进行不同强度的滤波.这些滤波计算都是多抽头的线性滤波,与插值计算类似,主要通过对边界像素点垂直或水平方向上多个像素做加权平均.使用二维存储结构,可以直接访问一行或列的像素,使用一个周期即可生成滤波后的像素点,这可以使用与插值计算相同的指令.由表 4可见采用了新的存储结构后,滤波所需周期急剧减少.

表 4　不同存储结构做去块效应滤波时所需周期

计算点	传统 harward总线结构下滤波操作所需周期	采用本文的二维并行存储所需周期
强亮度滤波	28	4
强色度滤波	18	4
基本滤波 $p_0 + q_0$	16	3
基本滤波 p_1, q_1	8	3

4　结　论

本文提出的一种二维的并行存储结构能够成倍的减少视频处理中的冗余存储访问,基于 SystemC的模型验证表明该结构成倍地减少视频计算所需的时钟周期,提高视频矢量处理器的算法执行效率,从而降低整个多媒体系统的成本和功耗,与现在的商业级高性能 DSP相比,视频处理性能获得了成倍的增加.

参考文献 (References)

[1] Analog Devices Inc. ADSP-BF533 Blackfin processor hrdware reference [EB/OL]. Norwood, MA: ADI, 2004 [2008 - 04 - 06]. http://www.analog.com

[2] CEVA Inc. ceva portable multimedia whitepaper [EB/OL]. San Jose: CEVA Inc, 2007 [2008 - 02 - 03]. http://www.ceva-dsp.com

[3] ISO/IEC 14496-10, Advanced video coding for generic audiovisual services[S]

[4] Tanskanen J K, Sihvo T, Niittylahti J. Byte and modulo addressable parallel memory architecture for video coding [J]. IEEE Transactions on Circuits and Systems for Video Technology, 2004, 14(11): 1270 - 1276

[5] Park J W. An efficient buffer memory system for subarray access [J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(3): 316 - 335

[6] Kant S, Mithun U, Pssbk G. Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications [C]// IEEE Consumer Electronics. ICCE '06 Digest of Technical Papers. Las Vegas: IEEE Consumer Electronics, 2006: 93 - 94

(责任编辑:张　嵘)