基于 TMS320DM642 的 H.264 解码器优化

陈梅芳
(厦门大学　福建 厦门　361005)

摘　要:通过分析 H.264 软件解码器的结构和复杂度,确定了解码器在优化过程中的重点和难点,并结合 TMS320DM642 DSP 性能特点,详细讨论了在 TMS320DM642 DSP 平台上 H.264 解码器所采用的优化方法。这些方法主要涉及提高程序代码的并行性和增强存储器访问的效率,重点是运动补偿、IDCT 等关键模块的优化。通过实验结果表明,本解码器可以实现 CIF 格式视频流的实时解码。

关键词:TMS320DM642;H.264 标准;解码器;DSP
中图分类号:TN919.3　　**文献标识码**:B　　**文章编号**:1004-373X(2006)03-112-04

Optimization of H.264 Decoder Based on TMS320DM642

CHEN Meifang
(Xiamen University, Xiamen, 361005, China)

Abstract: The architecture and complexity of H.264 software decoder are analyzed, the emphasis and difficulties of optimization are figured out, then combining with the characteristics of TMS320DM642, this paper discusses the implementation and optimization of H.264 decoder on TMS320DM642. The optimization is involved mostly with the enhancement of the code parallelism and the access efficiency of memory. This paper emphasizes on the optimization of the IDCT module and moving compensation module. The experiments result shows that the H.264 decoder in this paper can implement the real-time decoding of the CIF video stream.

Keywords: TMS320DM642; H.264 standard; decoder; DSP

1　引　言

JVT 伴随着每一个 H.264 工作草案的版本都推出一个软件编解码器的参考实现——JM 系列,他基本上实现了对应工作草案中所定义的功能。但是由于他主要用于测试目的,程序代码和结构没有经过优化,不管在性能上还是结构上都远不能实现软件实时解码的要求,这就使得 H.264 的实时解码成为人们非常关注的问题。

通过对 JM8.6 程序结构进行分析,可以发现 JM8.6 测试代码在解码过程中读和解一个宏块时,没有事先区分宏块的类型,而只是分别调用 read_one_macroblock()和 decode_one_macroblock()函数进行处理,许多相关判断都在函数内部进行,而且对每个 4×4 块都要判断预测模式或预测方向,另外,插值模块中参考块是否超出图像边界、重构模块中需对每个象素进行限值等操作都存在着大量的判断、跳转指令。从而使得 DSP 的流水线经常处于断流状态,大大影响程序运行的速度。

文献[3]对兼容 H.264 基本规范的解码器进行了复杂度分析,发现解码器中计算密集型功能模块依次为插值、反变换和重建。插值是根据运动向量在参考图像中获得一个预测图像块,对应非整数精度的运动向量,首先要插值求出参考图像中非整数点的象素值,为求出一个 1/4 象素点,在最复杂的情况下需要计算 32 次乘法、44 次加法和 3 次移位操作。另外,我们对 JM8.6 程序进行了时间复杂度分析,发现插值模块是最耗时的模块,其次是整数余弦反变换和重建模块、主要码流元素的读取及熵解码。所以在对 H.264 的解码器代码进行速度优化时,应结合 DSP 特性,对上述 3 大部分进行重点优化,而对解码器中的其他模块中比较耗时的操作,如内存拷贝、大块数据赋值等进行了局部优化。

综合上述 H.264 解码器优化中的困难和目前各个型号 DSP 的性能特点,本文采用 TI 公司推出的最新一代高性能数字信号处理器 TM320DM642 作为 DSP 处理平台。其时钟频率达到了 600 MHz,处理能力为 4 800 MIPS,完全可以实现 H.264 视频实时解码。其主要结构特性有:

(1) 采用改进的 VLIW(甚长指令字)DSP 核;

(2) 增强的 DMA 控制器(EDMA),具有 64 个独立的 DMA 通道;

(3) 6 个 ALUs(32/40 b),支持单 32 b,双 16 b 或 4 个 8 b 的算术运算,2 个乘法器,支持 4 个 16×16 b 的乘法运算(32 b 结果),或 8 个 8×8 b 的乘法运算(16 b 结果),64 个 32 b 的 GPIO 寄存器,可字节寻址;

收稿日期:2005-09-18

(4) L1/L2 DSP 高速缓存,128 kB(16 kB)L1P 程序 Cache,128 kB(16 kB)L1D 数据 Cache,L2 统一编址的 RAM/Cache(256 kB);

(5) 64 b 的外部存储器接口(EMIF)可实现与异步存储器(SRAM,EPROM)和同步存储器(SDRAM,SBRAM)无缝连接,外部存储器空间寻址范围达到 1 024 MB;

(6) 2 路视频输入,PAL/NSTC 制式;1 路视频输出,PAL/NSTC 制式,VGA,S 端子;

另外,TM320DM642 完全兼容 C64X 的软件资源,可充分利用 C64X DSP 提供的丰富指令资源,极大地扩展了运算处理能力,能够更方便快速地实现图像处理中的算法。其提供的强大包处理能力可同时处理 2 个 16 位数据和 4 个 8 位数据,并且还提供了 STDW(STNDW),LDDW(LDNDW)双字存取指令。

2 H.264 解码算法的 DSP 优化

本文在 JM8.6 测试软件的基础上,结合 DM642 本身的特点,对程序代码做了程序流程、算法和汇编语言等优化。

2.1 程序流程优化

(1) 优化程序的总体结构

首先将整个程序结构改为由主程序和 3 个线程 TSK 组成,主程序完成 DSP 初始化;取码 TSK 主要完成循环从缓冲区(已经存放了待解码的视频流数据)中读取一个 NALU;解码 TSK 主要完成对一个 NALU 单元进行分析、解码;输出显示 TSK 主要完成视频格式的转换和视频流数据的显示。

(2) 采用函数指针

根据宏块类型(帧内)/宏块划分方法(帧间),对各个宏块/分区设置独立函数分别读码流、解码。并根据子块/分区的预测类型,设置独立函数分别预测。并简化函数调用参数,清晰了程序流程、减少判断转移指令和方便下一级的汇编优化。

(3) 调整数据结构和内存分配

① 数据结构是指数据的类型及其在内存空间的分配方式,不同的数据结构对程序的性能有不同的影响。

C64X 只有 2 个 D 单元负责数据存取,并且从存储区取数据的装载(load)/送存(store)指令有 4 个时钟周期的延时。这种指令结构降低了 CPU 的执行效率,优化时,充分利用 C64X 提供的字/双字存取指令,减少从存储区 load/store 数据的次数;当然 C64X 也可以执行不符合边界调整要求的存储器访问,但每个指令周期却只能发出一个无边界调整限制的访问。因此优化时,应尽可能把需要同时处理的数据存放在具有字/双字边界的连续的内存空间。根据 DSP 的这个特点和一维数组更易于汇编优化的特点,在改进的程序中,调整了数据结构。对于图像数据、宏块预测数据、各个 4×4 残差数据等按行顺序存放在连续的一维数组中,并且数组分配的内存边界为 16 B,通过这样的数据结构调整,可以明显地提高程序的运行速度。

② 片内 RAM 与 CPU 工作在同一时钟频率,比片外 SDRAM 性能高得多,应尽可能在片内处理程序和数据。因此,把程序代码和图像参数、当前宏块数据(包括解码宏块参数、残差数据、预测数据和帧间插值需要的整数点象素数据等)、码流读取参数等所需内存都存放于片内 RAM 中,大大提高运行速度。

2.2 算法优化

在此仅讨论解码器中的熵解码优化。采用 CAVLC 进行熵解码时,需将码流中的码字与码表中的码字进行匹配,也就是进行码表查找。在读码字 coeff_token 时,原码表存储结构采用二维表结构,存储的内容为码字,二维下标分别代表解码后的两个语法元素 trailing_ones、total_coeff。而对于二维表结构,通过内容查找坐标,需要对整个表进行遍历,占用了大量的时间。本文更改码表的存储结构,各个表按码流中一定长度的码字的值划分为多个二维表,并通过码字值的大小进行判断,从而获取码字长度、trailing_ones,total_coeff 三个语法元素值。其中读取码流中的码字的长度由该码表中最长码字长度决定。

例:当解码 ChromaDCLevel 即 nC == −1 时,码表改造如图 1 所示。

trailing_ones	total_coeff	nC==-1
0	0	01
0	1	0001 11
1	1	1
0	2	0001 00
1	2	0001 10
2	2	001
0	3	0000 11
1	3	0000 011
2	3	0000 010
3	3	0001 01
0	4	0000 10
1	4	0000 0011
2	4	0000 0010
3	4	0000 000

```
coeff4_0[][3]={
{6,0,2},//0001 00
{6,3,3},//0001 01
{6,1,2},//0001 10
{6,0,1},//0001 00
};
coeff4_1[][3]={
{7,3,4},//0000 000(0)   {7,3,4},//0000 000(1)
{8,2,4},//0000 0010     {8,1,4},//0000 0011
{7,2,3},//0000 010(0)   {7,2,3},//0000 010(1)
{7,1,3},//0000 011(0)   {7,1,3},//0000 011(1)
{6,0,4},//0000 10(00)   {6,0,4},//0000 10(01)
{6,0,4},//0000 10(10)   {6,0,4},//0000 10(11)
{6,0,3},//0000 11(00)   {6,0,3},//0000 11(01)
{6,0,3},//0000 11(10)   {6,0,3}};//0000 11(11)
```

图 1 码表改造

2.3 汇编代码级的优化

汇编级的优化包括 2 部分:直接用汇编语言进行优化和采用线性汇编语言进行优化。对于需要重点优化的插值模块、反变换和重建模块等必须采用汇编语言进行优化。

2.3.1 直接用汇编语言进行优化

使用汇编语言进行并行编程难度比较大,但是在有些情况下,程序中数据有非常强的承接关系使得无法并行运行,并且该程序体逻辑关系清楚,使用的寄存器不超过 64 个,这时可以考虑直接使用汇编语言实现。

(1) 插值模块是解码器中最耗时的模块,因此有必要详细介绍该模块在 DSP 中的优化方法。插值模块的重点在于计算非整数象素点的插值,由于 1/4 象素点的值是由

整数象素点和 1/2 象素点的采样值平均得到,所以主要针对采用 6 抽头滤波器[1,-5,20,20,-5,1]进行插值的 1/2象素点进行分析。一个 4×4 预测块需要 4×9 个整数象素点进行插值,因为插值一排 1/2 象素点需要 9 个同方向的整数象素点,同时垂直方向插值所需的整数象素点可通过 C64X 提供的 PACK2,PACKH2 以及 PACKL4,PACKH4 指令进行转换到水平方向,所以本文仅介绍水平方向的插值优化方法。

A18				A19				A20
E	F	G	H	I	J	K	L	M
			1	-5	20	20	-5	1
		1	-5	20	20	-5	1	
1	-5	20	20	-5	1			
	1	-5	20	20	-5	1		

图 2 1/2 象素点插值

首先通过 LDNDW 和 LDBU 指令可把所需的 9 个整数象素点存放在三个 32 位寄存器中,然后根据图 2(紧凑)中从上而下的顺序,由 DOTPUS4、SHRMB,SHLMB 指令计算各个 1/2 象素点的 4 抽头[-5,20,20,-5]的点积值,存放在寄存器 SUMX(3,2,0,1)中,并把 E,F,G,H 和 J,K,L,M 这 8 个象素点存放在 2 个寄存器中,用 UNPKHU4,UNPKLU4 和 ADD2 指令计算各个 1/2 象素点的 6 抽头滤波中另 2 抽头[1,1]的和值,同时各加 16,存放在寄存器 SUM1X(0,1)中。由指令 PACK2 对 4 抽头的点积值进行打包后,用 ADD2 指令与 SUM1X 相加,并算术右移 6 位(SHR2),再由 SPACKU4 指令进行饱和打包便可快速获得限值到[0,255]范围的该排的 4 个预测值。同时为了增加指令并行,可同时对 2 排 8 个 1/2 象素点进行插值,分别在 A,B 两侧的寄存器组中进行。实验结果表明可在 30 个时钟周期内完成一个 4×4 预测块的 1/2 象素点插值。

(2) 在 H.264 标准中,整数变换的基本单位是 4×4 点,具有只用加法和移位就可以实现的优点,并可完全在 16 b 精度下进行,结合 C64X 提供的半字打包、加、减、移位操作指令,可方便、快速优化 IDCT,并极大提高运算速度。整个汇编优化过程比较简单,基本思路框图如图 3 所示。

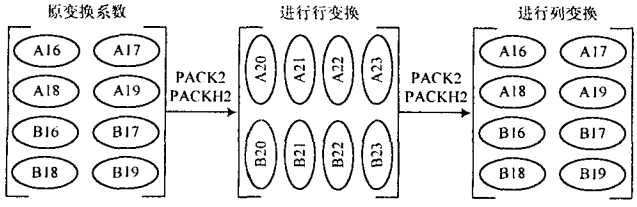

图 3 IDCT 变换优化框图

2.3.2 使用线性汇编语言进行优化

在程序中影响性能的主要代码通常是循环,可对于某些以循环为主的函数或者抽出复杂函数中的循环体使用线性汇编语言进行改写,经过汇编优化器进行优化之后,效率是非常高的。在此介绍了图像重建模块代码的线性汇编优化过程,图像重建就是在重建的残差数据上加上预测数据后限值到[0,255]范围。

其中预测数据是 8 位无符号整型,残差数据为取值范围为[-255,255],是 16 位有符号整型。对于各个 4×4 块重建的 C 语言代码如下:

```
for(i = 0 ; i < 4 ; i ++){
   for(j = 0 ; j < 4; j ++){
     Image[j] = max(0,min(idct[j] + pred[j],255));
                        }
                idct += 4;
                pred += predWidth;
                Image+= PicWidthInSample;
}
```

对于此段代码,需要对每个象素点的重建值进行判断,从而影响编译器优化,对此主要利用 C64X 提供的带饱和的打包指令 spacku4 进行算法改进,大大降低了函数的运行时间。内循环的线性汇编优化流程如图 4 所示。

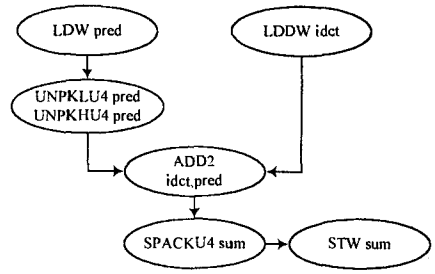

图 4 内循环的线性汇编优化流程图

3 优化结果

本文为了测试解码器的效率,首先采用文献[6]中提供的通用编码参数用 JM8.6 编码器对一系列标准测试系列进行编码。通用编码参数除了保证码流符合基本规范的参数如不支持场编码和自适应帧场编码、不支持基于上下文的自适应算术编码、无 B 帧等,其他主要参数的设置是:ProfileIDC 为 77,LevelIDC 为 3.0, IntraPeriod 为 10,搜索范围为 16,参考帧为 2,使用 7 种运动搜索块,使用哈达玛变换,无分区,不分段,不使用受限的帧内预测模式,不使用差错避免机制等,并且通过调整量化参数的值可以调整编码后的码流码率。分别用优化后的解码器和未经优化的解码器在 DM642 上对编码器生成的码流进行解码后,可以得到在不同比特率下的平均每帧解码时间,如表 2所示。

从表 2 中可以看出,在各种比特率条件下,解码速度都可以获得大幅度提升,并且在比特率较低时比比特率较高时速度提高更为明显。上述实验结果直接说明了本文所采用的算法效果是十分卓越的。优化后的解码器基本能够实现 30 f/s 的 CIF 视频的实时解码,满足可视电话、视频会议等领域的实时要求。

表 2 解码器平均每帧解码时间比较

测试系列(CIF 120 帧)/30f/s	QP	比特率 kb/s	优化前 ms/f	优化后 ms/f
vectra _ color	24	2 196.83	12 708	31.69
	28	1 385.00	11 883	27.89
	32	811.94	11 175	24.54
	36	461.14	10 642	21.83
paris	24	1 344.60	10 158	27.27
	28	866.23	9 592	25.09
	32	553.77	9 242	23.04
	36	341.16	8 951	21.23

4 结 语

在实际的优化过程中,还采用了:利用编译器选项优化——根据系统的要求,不断地对各个参数(- mx - pm - 03 - mt 等)进行选择、搭配、调整,选择合适的优化选项,提高程序执行的并行性和通过 EDMA 技术搬移数据进行优化、参考图像边界扩展优化等方法,表 2 的测试数据也包含了这些优化方法的结果。当然,基于本文开发的 H.264 解码器在性能方面还有待于进一步的优化改进,以达到 D1 格式视频的实时解码,满足大部分 H.264 视频解码业务的要求。

参 考 文 献

[1] Iain E G Richardson. H.264 and MPEG - 4 Video Compression[M]. The Robert Gordon University, Aberdeen, UK. 2003.

[2] Horowitz M, Joch A, Kossentini F, *et al*. H.264/AVC Baseline Profile Decoder Complexity Analysis[J]. IEEE Trans. on Circuits and Systems for Video Technolog, 2003, 13(7): 704 - 716.

[3] Lappalainen V, Hallapuro A, Timo D H. Complexity of Optimized H.26L Video Decoder Implementation [J]. IEEE Trans. on Circuits and Systems for Video Technology, 2003, 13(7): 717 - 925.

[4] 魏芳,李学明. 基于 MMX 技术的 H.264 解码器优化[J]. 计算机工程与设计,2004,25(12):2 218 - 2 221,2 224.

[5] H.264 测试模型 JM8.6. http://bs.hhi.de/~suehring/tml/download/jm86.zip.

[6] Sullivan G. Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low - reolution Progressive - scan Source Material [EB/OL]. ftp://standards.pictel.com/video - site/0109 _ San/VCEG - N81.doc.

作者简介 陈梅芳 男,1979 年出生,厦门大学 2003 级硕士研究生。研究方向为 H.264 视频标准研究及 DSP 应用。

(上接第 109 页)

表 1 为实例属性。表 2 是微机局域网内运算环境的部分实例直流分析运行结果,运行环境 Windows 2000,开发环境 VC++6.0。表 3 是单机多 CPU 运算环境的部分实例直流分析运行结果。运行环境为 SunUltra80 4CPU (450 MHz) Memory 4 G OS:Solaris7。

表 1 待分析实例统计属性

	通孔数	AL2 层节点数	AL3 层节点数	AL4 层节点数	总节点数
Data1	568 252	117 234	1 229 817	1 112 337	3 027 640
Data2	1 808 849	139 633	1 226 545	1 093 937	4 268 964

表 2 微机局域网内运算环境部分实例

直流分析运行结果 (时间单位 h:m:s)

	数据读取时间	子网计算时间	总网计算时间	总时间
(单台 P4 1.4 G)				
Data1	00:01:05	00:31:00	00:08:25	00:40:30
Data2	00:01:00	02:08:15	00:00:00	02:09:15
(两台 P4 1.4 G)				
Data1	00:01:05	00:16:00	00:07:30	00:24:30
Data2	00:01:00	01:10:20	00:00:00	01:11:20
(两台 P4 1.4 G,一台 P3 933)				
Data1	00:01:05	00:11:20	00:07:20	00:19:45
Data2	00:01:00	00:55:04	00:00:00	00:55:59

表 3 单机多 CPU 运算环境的部分实例

直流分析运行结果 (时间单位 h:m:s)

	1CPU	2CPU	3CPU	4CPU
Data1	00:40:12.8	00:23:59.9	00:19:42.9	00:17:45.4
Data2	03:09:31.4	01:38:47.4	01:05:34.2	00:51:21.4

参 考 文 献

[1] Dharchoudhury A, Panda R, Blaauw D, *et al*. Design and Analysis of Power Distribution Networks in PowerPC Microprocessors [A]. Proc. 35th Design Automation Conference[C]. 1998. 738 - 743.

[2] Steele G, Overhauser D, Rochel S, *et al*. Full - chip Verification Methods for DSM Power Distribution Systems [A]. Proc. 35th DAC[C]. 1998. 744 - 749.

[3] Min Zhao, Rajendran V Panda, Sachin S Sapatnedar, *et al*. Hierarchical Analysis of Power Distribution Networks[A]. Proc. of 37th Design Automation Conference[C]. 2000. 150 - 155.

[4] 都志辉. 高性能计算并行编程技术——MPI 并行程序设计[M]. 北京:清华大学出版社,2001.

作者简介 张 敏 女,1978 年出生,2001 年毕业于浙江工业大学计算机系,获工学学士学位,现为杭州电子科技大学 CAD 研究所 03 级硕士研究生。主要研究方向是集成电路 CAD。