# H.264 算法在 TI C64x 上的移植及优化

李 杰 蔡灿辉
(华侨大学信息科学与工程学院,福建泉州 362021)

**摘 要** 论文介绍了 H.264 标准代码 jm81a 从 VC6.0 移植到 TI 公司的 TMS320 C6416 平台上的过程及部分优化方法,讨论了移植过程中经常遇到的实际问题,并给出了优化前后的实验结果比较。

**关键词** DSP H.264 移植 优化

**文章编号** 1002-8331-(2005)33-0109-03 **文献标识码** A **中图分类号** TP391

# H.264 Algorithm Migration and Optimization Based on TI C64x

Li Jie Cai Canhui
(Institute of Information Science and Technology, Huaqiao University, Quanzhou, Fujian 362011)

**Abstract**: This paper introduces the migration of jm81a, a standard coding for H.264, from VC 6.0 to TI TMS320 C6416 platform and the methods of optimization. Difficulties generally encountered in the migration are discussed and the experimental results before optimized and after are given to verify the effect of the optimization.

**Keywords**: DSP, H.264, migration, optimization

## 1 概述

ITU-T 的 H.264 的标准化工作是由 ISO/IEC 的运动图像专家组 MPEG(Moving Pictures Experts Group)和 ITU 的视频编码专家组 VCEG(Video Coding Experts Group)共同成立的 JVT(Joint Video Team)负责完成的。jm81a 是一个 C 语言实现的实用的 H264 软件。由于 H.264 采用了许多以往标准中所没有的先进技术,因此,在相同的码率下用 H.264 标准进行编码能够获得比其他标准更高的主客观质量。H.264 的巨大成功表明视频编解码技术仍然具有较广阔的研究空间。尽管 H.264 具有很多以往标准无法比拟的优点,但该标准计算复杂度高,用一般的图像处理芯片难于达到实时编解码的要求。TI 公司生产的 C64 系列芯片具有很强的并行处理能力和信号处理功能,只要根据该 DSP 的结构特点对 H.264 算法进行优化,就可以在主客观质量损伤很小的情况下降低复杂度,实现 H.264 标准下的实时编解码。本文以 jm81a 为蓝本,讨论 H.264 代码在 TI 公司的 TMS320 C6416DSK 上的移植及部分优化过程。

## 2 H.264 的技术特点

与现有的视频编解码标准一样,H.264 标准也采用基于块的混合编码框架。即在编码前把每幅图像都划分成固定大小的宏块:16×16 像素的亮度宏块和两个 8×8 像素的色度宏块,多个宏块组成一帧。其主要技术特点如下:

(1)帧内预测

帧内预测用邻近块的像素(当前块的左边和上边)做外推对当前块进行预测,并对预测的残差进行编码,以消除空间冗余,用较少的比特数来表达像素块信息。

(2)帧间预测

帧间预测采用多帧参考和更小运动预测区域等方法,对下一帧精确预测,从而降低图像的时间冗余,减少传输的数据量。在其它标准中,处理的像素块大小均为 16×16 或 8×8,H.264 支持 7 种不同尺寸和形状的宏块和子宏块分割,可以达到 1/4 像素运动精度。

(3)环路滤波

H.264 标准中引入了基于内容的去块(效应)滤波器,降低高压缩比产生的明显的块失真效应。去块滤波基于 4×4 块边界,在块边界,滤波强度与块的编码模式、运动矢量和残差数值相关。解码端在反变换后、宏块重构前解去块滤波器,根据宏块中每一个 4×4 块的位置和量化参数的不同,对每一条块边界设置不同的滤波强度。

(4)变换和量化

以前的编码标准中基本都采用 8×8 的离散余弦变换,而在 H.264 中则采用了 4×4 的整数变换,具有效果好、计算快、反变换过程中不会出现失配问题等优点,也能够减少块效应和振铃效应。

(5)熵编码

H.264 使用了两种熵编码方法:

①基于上下文的自适应变长编码(CAVLC)与一致变字长编码(UVLC)相结合的编码;

②基于上下文的自适应二进制算术编码(CABAC)。

## 3 TMS320C64x 芯片简介

TMS320C64X 系列 DSP 是美国 TI 公司推出的新一代并行处理 DSP 芯片,属于目前高性能的定点处理芯片。TMS320C64x 片内有 8 个并行的处理单元,分为相同的两组。它的体系结构采用 VLIW(超长指令集)结构,每条指令字长为 32bit,8 条指令组成一个指令包,总字长为 8×32=256bit。芯片内部设置了专门的指令分配模块,可以将 256bit 的指令包同时分配到 8 个处理单元,并由 8 个单元同时运行。它具有高达 600MHz 的 CPU,

**基金项目**:福建省自然科学基金资助项目(编号:A0410018)

**作者简介**:李杰,硕士研究生,主要研究方向为视频压缩技术与 DSP 应用。蔡灿辉,教授,主要研究方向为图像编码与传输。

从而使得其运算能力达到4 800MIPS(600MHz×8条指令=4 800MIPS)。这些性能都确立了它在高端多媒体应用中的地位。

## 4 H.264源代码在DSP上的移植过程

TI公司提供了用于C语言开发的CCS(CODE COMPOSER STUDIO)平台,该平台包括了优化的ANSI C编译器,使之可以使用C语言开发DSP程序。这种方法不仅使DSP开发的速度大大加快,而且DSP程序的可读性和可移植性大大增加,程序修改也很方便。由于CCS是DSP的专用开发环境,因此CCS下的C与一般计算机使用的C又有所区别,主要表现在:

(1)DSP的C语言是标准的ANSI C,它不包括同外设联系的扩展部分,如屏幕绘图等。但在CCS中,为了方便调试,可以将数据通过prinf命令虚拟输出到主机的屏幕上。

(2)DSP的C语言的编译过程分两步:首先把C编译为汇编(asm)代码,再把汇编代码编译为可执行(obj)代码。因此,C和汇编代码直接对应,关系非常明确,便于人工优化。

(3)DSP的代码需要绝对定位;主机的C的代码由操作系统定位。

(4)DSP的C代码效率较高,非常适合于嵌入系统。

由于jm81a是在基于VC平台的应用软件,为了使jm81a能在CCS运行下有部分差异,必须做相应的修改。下面就jm81a的实际移植过程问题以及应注意事项给予说明。

(1)库文件的改动

因为CCS是对应DSP专用的开发环境,其支持库也和DSP硬件相联系,所以对VC的部分库函数包含的头文件要做相应的改变,使其适应CCS运行环境。例如VC中支持的malloc,calloc等动态存储分布函数,在CCS中都包含在stdlib库中,因此,必须对原来的include文件进行修改。

(2)变量的存取方式调整

在CCS中程序按段式存储,各主要段存放内容如下:

text 存放可执行代码
.cinit 存放初始化全局变量和静态变量表
.switch 用于存放分支跳转表
.bss 存放静态和全局变量
.far 用于存放声明为far调用的全局和局部变量
.stack 存放系统栈
.system 存放动态存储空间分配堆

CCS的C编译器支持两种内存模式:小模式和大模式。不同的内存模式影响对.bss段变量的访问。程序中的全局和静态变量都分布在.bss段内,小模式下,其总和不能超过32K。由于jm81a中的全局和静态变量已经超过32K,所以在编译选项(build options)中应采用-ml2,即对函数调用采用far调用,集合数据far类型存取。

(3)数据类型的调整

在CCS中没有定义long long类型,long表示40位长整型,double表示64位浮点型;而在VC中long和int都表示32位整型。因为C64系列通用寄存器均为32位,访问40位数据时,要对两个寄存器进行读写操作,从节约CPU处理时间角度考虑,应对其进行相应的数据类型调整。

(4)存储空间的分配

在CCS中,存储空间的分配是通过配置.cmd文件实现的。存储空间分配前,必须了解芯片整个可用的内外存储空间大小。编译后的C程序跑飞一般是对不存在的存储区访问造成的。可以检查.MAP文件并与memery map对比,看是否超出了访问允许范围。对于jm81a的移植还要注意到下面问题:

首先,存放的所有的可执行代码的.text段要尽量放在片内,提高代码的执行速度。TI的C64系列有1M的存储空间,而jm81a编译通过后.text段大小只有430K左右,所以完全可以放在片内。

其次,要重新分配堆(heap)和栈(stack)的大小。通过在.cmd文件中设置-heap和-stack可以实现大小的配置。heap用于动态存储空间的分配,对应.sysmem段,stack用于保存函数的返回地址,对应.stack段。C编译器堆栈的默认大小为1KB,而且无法检查程序运行时堆栈能否溢出。在jm81a的视频解码过程中,存储参考帧及其它结构体需要很大的动态存储开销,应尽量多分配heap空间。由于C64片内存储空间只有1M,因此heap空间只能放在6416DSK片外的16M存储空间中。stack空间的分配可以根据使用情况而定,程序不正常跑飞时应注意检查是否堆栈溢出。

最后,必须对VC代码中的空间分配作必要的调整。因为jm81a是基于PC机开发的,空间分配都是按照最大帧尺寸进行分配。DSP存储资源有限,只能根据需要分配空间,因此要对代码中的一些空间分配做相应的改动。例如:defines.h中MAX_CODED_FRAME_SIZE原来定义为800 000,对于一般尺寸的视频图像解码,改为大约20 000就已经足够了。

(5)部分结构的初始化

在jm81a中dpb结构体各个成员值在VC中初始化为0,而在CCS中却没有对其初始化,因此,必须对一些结构进行必要的初始化。例如:dpb.ref_frames_in_buffer为解码缓冲区的参考帧数目,初始值应该为0,但CCS不对其初始化,所以通常会是一个很大的数,如不进行初始化,程序运行就会出错。

(6)其他注意事项

在C64x中,寄存器都是32位大小,一次可以同时处理32bit数据。如果程序中的存储地址不是合法的32位字地址,在程序装入数据的时候,LDW指令会对这个地址自动进行调整使之成为一个合法的字地址。

例如下面程序:

```
#include<stdio.h>
unsigned int b;
unsigned char a[10]={0x12,0x34,0x56,0x78,0x9a,0xbc,0xde,0xff,
0xc3,0xb4};
void main()
{
  b=*(unsigned int*)(a+2);
  printf("%x",b);
}
```

在CCS下运行结果为78 563 412,而VC下为bc9a7856。因为$a$+2在CCS中不是合法的32位地址,所以地址$a$+1,$a$+2,$a$+3都会在程序装入时校准到$a$。

也就是说在CCS中,上述程序地址为$a$,$a$+1,$a$+2,$a$+3会得到同样的结果。同理,地址$a$+5,$a$+6,$a$+7都会被校准到$a$+4。

## 5 代码的初步优化

(1)调整编译选项的内容

c6x的编译程序为"cl6x.exe"格式如下:

cl6x [options] [filenames]

编译选项是一个字母或者两个字母,对大小写不敏感。编译选项的前面需要有一个"-"号。一个字母的选项可以合并在一起。比如"-sgq"与"-s -g -q"相同。两个字母的选项如果第一个字母相同也可以合并在一起。比如"-mgt"与"-mg -mt"相同。

笔者在代码调试过程中采用下列优化选项 -gp -pm -op0 -o3 -mt -mh,具体作用说明如下:

-gp:装载时自动使能函数级分析,函数调用时代码性能微小降低。

-o3:对文件级别进行最强的优化,一般在编译时应该使用这个选项。但是在个别情况下使用这个选项优化程序可能会出现错误(-o2 有相同现象,-o0 和-o1 不会出现错误)。通常错误会在优化循环,组织流水线的时候发生。如果有这种现象出现可以同时使用-g 选项,程序优化就不会出现错误,但是优化效果会下降。

-pm:在程序级别进行优化。可以将所以文件联合在一起进行优化,主要有去掉没有被调用的函数、总是常数的变量以及没有使用的函数返回值。有可能应尽量使用。

-op0:说明有外部变量引用和函数调用。

-mt:表示在程序中没有使用 alaising 技术,例如两个指针指向同一目标。这使得编译器可以进行比较好的优化。

-mh[n]:去掉流水线的 epilog,减小程序的大小。这个选项的作用比较明显。但是有可能出现读取地址超出有效范围的问题,所以要在数据段的开始和结尾处增加一些 padding,或者在分配内存时保证数组的前面和后面一段范围内都是有效的地址。可选的参数 $n$ 给出这种 pading 的长度字节数。

(2)设置 cache

DSP 芯片中的片内存储空间可以设置为 L2cache。使用 cache 可以提高片内外数据的交换速度。在 C64x 中,支持 4 种 4 路 L2cache 的大小设置,分别为 32K,64K,128K 和 256K。

cache 的设置可以通过调用 CCS 的 CSL (芯片支持库)中的函数来实现。在.cmd 文件中也要相应地改变 ISRAM 的大小。

例如要设置 32Kcache,可以调用如下 CSL 函数

```
CSL_init();
CACHE_enableCaching(CACHE_EMIFA_CE00);
CACHE_setL2Mode(CACHE_32KCACHE);
```

.cmd 文件中相应设置 ISRAM 的大小为 0xF0000, 防止代码被写入 cache 造成存储访问错误。

(3)使用 DSP/BIOS

DSP/BIOS 是 TI 公司推出的一个实时操作系统。DSP/BIOS 与 TI 的 CCS(Code Composer Studio)集成在一起。应用 DSP/BIOS 可以大大简化 DSP 应用程序的开发和调试过程。DSP/BIOS 实际上是一组可重复调用的系统 API 模块的集合。它只占用 DSP 很少的资源, 就可以满足 DSP 实时运行时的调试性能分析,编写高效的程序。例如,在 TMS320C6211 上运行 printf()函数需花费 4 000 个周期,而运行 LOG_printf()只花费 36 个周期,即 DSP/BIOS 的 LOG _pfrintf()比 printf()要节省 100 倍以上的时间,其编程效率,可见一斑。

利用 DSP/BIOS 的数据获取(data capture)、统计(statistics)和事件记录功能(event logging),在软件调试阶段与主机 CCS 内的分析工具 BIOScope 配合, 可以完成对应用程序的实时探测(probe)、跟踪(trace)和监控(monitor)。

用 foreman_cif.yuv 序列测试,解码 5 帧 cif baseline 序列经上述改动后,处理时间从改动前的 19s 减少到改动后大约只需 3s 左右,速度提高了 5 倍多。

此外,CCS 中没有 timeb.h, 只能用 DSP/BIOS 中的 CSL 库函数 CLK_gethtime 实现原有的计时系统。

(4)优化存储分配

由于 DSP 芯片本身存储空间较小,如何优化内存管理,高效地利用存储空间是实现基于 DSP 的视频编解码中的一个重要问题。在 jm81a 中, 解码输出缓冲的帧数 dpb.size 由 getDpbSize()函数读入,解码器根据序列参数设置级别 active_sps->level_idc 和图像的大小 pic_size 分配缓冲区。其中,level idc 参数是在编码器配置文件 encoder.cfg 中设定的,解码器配置文件 decoder.cfg 不能改变其大小。jm81a 中默认的 level_idc 为 30,解码 cif 格式的图像帧时,decoder 中 dpb.size 默认大小为 20。

考虑到 DSP 本身存储空间较紧张, 如果解码 baseline,把 dpb.size 改为 1,level_idc 改为 10 已经足够。这时解码只用前一帧作为参考,每当检测到缓冲区满时,就会调用调度算法输出一帧。必须指出,由于 h264 支持多帧参考预测,jm81a 的默认参考帧为 5, 即 encoder.cfg 中的 NumberReferenceFrames 取值为 5,必须作相应的修正,把 NumberReferenceFrames 改为 1,才不会因为编解码参考帧数不同,影响解码效果。

用 foreman_cif.yuv 序列测试,解码 5 帧 cif 格式的 baseline 序列,改动后 heap 可以减少大约 3M~4M,大大节约了存储空间。

(5)其他优化改动

在 jm81a 中有很多代码参与到函数调用但是并没有实际作用。例如:find_snr 函数只是和参考帧比较求 SNR,如果解码正确,则显示结果应为 0,在完成程序移植后,可以去掉这类代码。

## 6 进一步优化方法

其它的优化方法还包括直接调用 C6000 的 intrinsic 内联函数,用 inline 关键字内联小函数,把一些小的循环进行展开,尽可能减少条件分支语句,避免流水线被做空,用移位运算代替除法,调整程序的执行顺序,给变量合理分配存储空间等等。这些措施可以使编译器能根据 C64 的硬件结构,充分利用内部的 8 个处理单元进行并行和流水处理。通过上述方法处理,可使代码执行速度有很大的提高。

用上述部分方法对 jm81a 中瓶颈函数 vlc.c 的 showbits 优化结果如表 1。

表 1 优化结果

| 优化前 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Areas | Code Size | Count | Incl.Total | Incl.Maximum | Incl.Minimum | Incl.Average | Excl.Total | Excl.Maximum | Excl.Minimum | Excl.Average |
| ShowBits | 176 | 121723 | 17727951 | 433 | 40 | 145 | 17727951 | 433 | 40 | 145 |
| 优化后 | | | | | | | | | | |
| Areas | Code Size | Count | Incl.Total | Incl.Maximum | Incl.Minimum | Incl.Average | Excl.Total | Excl.Maximum | Excl.Minimum | Excl.Average |
| ShowBits | 188 | 121723 | 5620799 | 342 | 46 | 46 | 5620799 | 342 | 46 | 46 |

(下转 142 页)

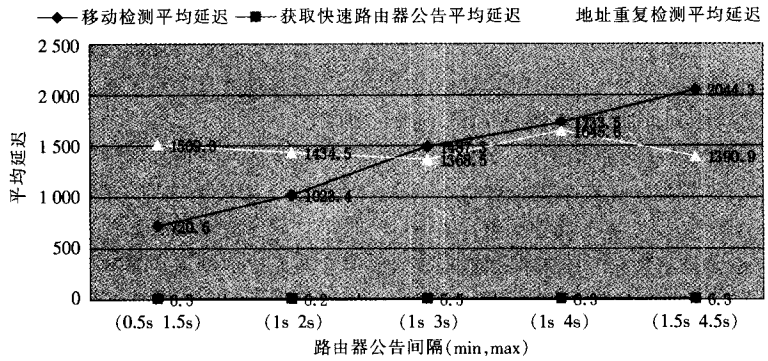


图5 不同路由器公告间隔的移动检测,地址重复检测和获取快速路由器公告的平均延迟

的路由器公告实现，将随机延迟0到MAX_RA_DELAY_TIME(缺省为500ms)后响应路由器请求。表1是采用基于隧道的快速切换方法时,对使用接入路由器信息缓存机制和不使用接入路由器信息缓存机制两种情况下测试的移动IPv6切换平均延迟。如表所示,当使用接入路由器信息缓存机制时,切换平均延迟由14.8ms减少到7.7ms,减少了7.1ms,这个时间正好为移动节点发送路由器请求而获取快速路由器公告的时间,并且移动节点在切换后不需要立即发送路由器请求,节省了无线带宽。

## 6 结论

本文对移动IPv6切换性能进行了分析，提出了一种在WLAN中实现基于隧道的移动IPv6快速切换的方法。这种方法通过结合L2-UP触发器和快速路由器公告，实现了将移动节点连接的无线接入点链路地址快速映射成其连接的接入路由器的IP信息。同时,用户可以使用灵活的接入路由器信息缓存机制,加速快速切换过程,并且节省无线带宽。实验结果表明,与基本的移动IPv6切换相比,本文提出的实现方法获得了很好的延迟性能。下一步研究工作是考虑当移动节点切换时,如何将其在接入路由器上的QoS参数,头标压缩参数,安全参数等的转移与本文的快速切换方法相结合,使得当移动节点切换到新的子网时,不用再重新协商这些参数。

(收稿日期:2005年9月)

## 参考文献

1.S Deering,R Hinden.Internet Protocol,Version 6(IPv6)Specification[S].RFC2460,1998-11

2.D B Johnson,C Perkins.Mobility support in IPv6[S].draft-ietf-mobile-ipv6-15.txt(work in progress),2001-07

3.G Krishnamurthi,R Chalmers,C Perkins.Buffer Management for Smooth HandOvers in Mobile IPv6[S].draft-krishnamurthi-mobileip-buffer6-01.txt(work in progress),2001-03

4.G Dommety.Handoff Optimization with no prediction and minimal L2 Trigger information[S].draft-dommetymobileip-min-handoffv4and6-01.txt(work in progress),2001-03

5.K El Malki,H Soliman.Simultaneous Bindings for Mobile IPv6 Fast Handoffs[S].draft-elmalki-mobileipbicasting-v6-02.txt(work in progress),2001-06

6.Shim E,Wei H,Chang Y et al.Low Latency Handoff for Wireless IP QOS with NeighborCasting[C].In:Proceedings of ICC 2002,2002:3245~3249

7.J Kempf,M M-Khalil,B Pentland.IPv6 Fast Router Advertisement[S].draft-mkhalil-ipv6-fastra-01.txt(work in progress),2002-05

8.Caceres R,Padmanabhan V N.Fast and Scalable Wireless Handoffs in Support of Mobile InternetAudio[J].ACM MONET Journal,1998;3(4)

9.A J Tuominen,H Petander.MIPL Mobile IPv6 for Linux in HUT Compues Network MediaPoli[C].In:Proceedings of Ottawa Linux Symposium 2001,Ottawa,Canada,2001-06

10.李津生,洪佩琳.下一代Internet的网络技术[M].北京:人民邮电出版社,2001:253~263

(上接111页)

表1为foreman_cif.yuv序列测试,解码1帧cif格式的baseline序列的profile结果,编译优化选项为-pm -op0 -o3 -mt -mh。

可以看出,经过优化的函数需要的指令周期比优化前降低了一个数量级。

## 7 结束语

本文从应用的角度出发，讨论了H.264算法实现程序jm81a在TI C6416DSK上的移植过程及应注意的具体事项,并给出了初步优化的方法和优化后试验结果。由于VC和CCS开发平台的差异,以及DSP自身硬件结构特点,代码移植必须根据这些特点进行相应的变动。C64系列DSP的硬件资源为高性能的视频编解码提供了必要的条件,如何充分利用这些硬件资源及开发工具，使基于DSP的H.264编解码器达到所期望的性能,满足实时性的要求仍然是值得大家关注的问题。

(收稿日期:2005年3月)

## 参考文献

1.Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification(ITU-T Rec.H.264|ISO/IEC 14496-10 AVC)[S].7th Meeting:Pattaya,Thailand,2003-03

2.Code Composer Studio User's Guide.Texas Instruments Incorporated.2000

3.TMS320C6000 Optimizing Compiler User's Guide.Texas Instruments Incorporated,2002

4.TMS320 DSP/BIOS User's Guide.Texas Instruments Incorporated,2003

5.TMS320C64x DSP Two-Level Internal Memory Reference Guide.Texas Instruments Incorporated,2004

6.李方慧,王飞,何佩琨.TMS320C6000系列DSPs原理及应用[M].电子工业出版社,2003