H.264在DM6446平台上的移植与实现

■ 文/北京邮电大学信息与通信工程学院 张伟华 孙松林 景晓军

摘 要：视频编码系统对压缩率和实时性有很高的要求，X264算法的高压缩率和TMS320DM6446平台的高性能为最新的H.264算法的实现提供了可能。本文概述了DM6446芯片特点，X264算法的编码过程，X86与C64x+平台的差异性，讨论了基于X264模型的算法移植过程，并给出了QCIF图像序列的测试结果。

关键词：移植 DM6446 X264 双核 C64x+ DSP

一、概述

H.264/AVC是由ISO/IEC和ITU-T共同成立的联合视频小组JVT（Joint Video Team）制定的。它引入了多种宏块划分、多模式的帧内和帧间预测、高精度亚像素运动估计、多参考帧等技术，提高了压缩比；采用4×4整数变换，避免了精度损失；在编码结构中，提出了VCL层和NAL层的概念，提供了网络友好性。基于H.264实时编码器的设计成为通信领域的热点问题。

考虑到H.264标准实现的复杂度，本文采用X264算法进行移植，该模型较JM模型实现简单，摒弃了一些耗时但对编码性能提高微小的模块；采用TI公司的Davinci TMS320DM6446作为X264算法的实现平台，以提高硬件的处理速度和能力。

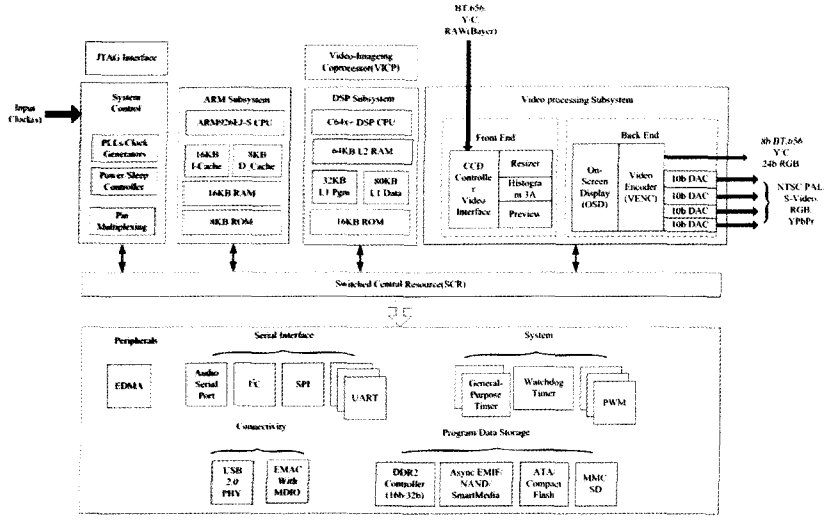

图1 DM6446功能框图

二、Davinci DM6446介绍

如图1所示，DM6446由内部结构和外设模块组成，内部结构分别由ARM子系统、DSP子系统、视频处理子系统、控制系统组成。ARM子系统由ARM926EJ-S核，16KB I-cache，8KB D-cache，16KB RAM，8KB ROM组成；DSP子系统由TMS320C64x+ DSP核，64KB L2 RAM，32KB L1 Pgm cache，80KB L1 Data cache组成；视频处理子系统由VPFE，VPBE组成，其中VPFE由CCD controller，resizer，histogram/3A，preview构成，VPBE由On-screen Display（OSD），Video encoder（VENC）以及DAC构成。DM6446所提供的外设接口有存储器接口、串行接口、系统接口等。

DM6446中ARM作为主控制器，通常负责设备的配置和控制，包括同外部接口做数据交换。特点如下：ARM926EJ-S RISC processor，ARMv5TEJ指令集，支持Little-Endian，Co-Processor 15（CP15），MMU，ARM中断控制器，PLL控制器，写缓冲等。

DSP核主要用作数据的处理和操作，包括8个功能单元，其中2个通用寄存器（A和B），每个都有32个32位寄存器组成，这些通用寄存器可以存放数据或者地址指针，加快DSP访问速度。每个功能单元在每个时钟周期内执行一条指令。由于C64x+TM内核主频高达594MHz，故指令数可达4752MIPS，运算速度很快。

DM6446中的ARM核和DSP核之间的通信主要是通过相互的中断来实现，二者可以共享外设资源和各自内部的RAM资源。

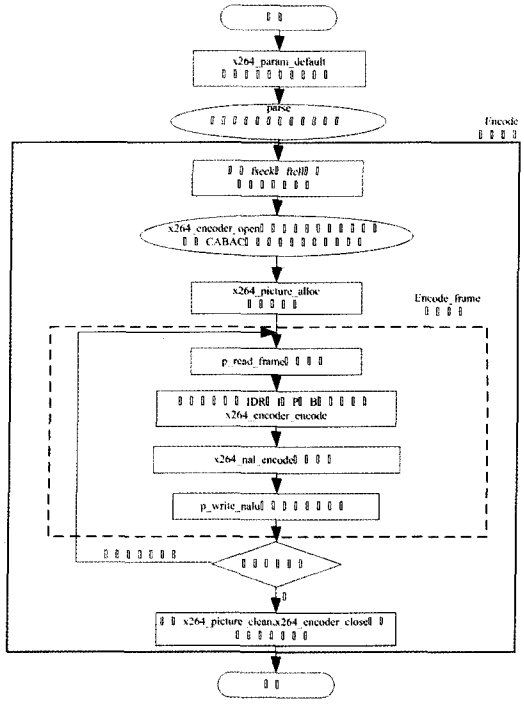

图2 X264编码流程图

三、H.264的算法移植

1.X264算法分析

基于上述编解码模型的特点，本文采用X264算法（20081113版）进行移植，算法的编码流程如图2所示。

整个算法分为视频数据的读入，压缩编码，编码后数据的存储三个部分。其中最关键的是数据的压缩编码，可以分为如下两个步骤：

（1）判断帧类型，初始化相关参数

如果当前编码帧是IDR帧，意味着一个新切片的开始。为了防止编码错误扩散，H.264规定当前切片不以本切片以外的其它帧为参考，因而遇到IDR帧需要重置参考帧区域。同时将SEI（Sequence Enhancement Information）、SPS（Sequence Parameter Set）和PPS（Picture Parameter

Set）信息编码写入各自的NAL单元。这三个参数集集合了编码的核心参数，是解码端解码的依据。如果不是IDR帧，依据帧类型设定当前NAL单元的类型参数。

然后进行一系列参数初始化操作，包括建立参考帧序列list0和list1，初始化量化参数QP、MV、CABAC等结构体，初始化比特流的起始位置和大小，准备编码。

（2）以宏块为单位对视频数据进行压缩编码

H.264中宏块在进行预测编码之前必须得到相邻块的信息。X264模型中开辟了intra4x4_pred_mode和non_zero_count区域用来存放当前宏块左侧和上侧相邻块的预测模式和非零值个数。在编码开始之前通过x264_macroblock_cache_load（）函数将相关数据载入缓存，通过x264_macroblock_analyse（）函数确定宏块的最佳编码方式。I帧采用9种4×4模式和4种16×16进行帧内预测；P帧可采用帧间预测模式下的16×16、16×8、8×16、8×8、8×4、4×8、4×4这7种宏块分割方式，也可使用帧内预测。

接着调用x264_macroblock_encode（）函数对宏块进行编码，编码流程按照预测，变化，zigzag扫描和熵编码的过程以先亮度块后色度块的顺序执行。同时，SKIP模式调用x264_macroblock_encode_pskip（）和x264_macroblock_encode_skip（）函数进行单独处理。

2.算法裁剪

（1）去除优化指令

原始的X264算法利用了基于X86的MMX、SSE等技术，针对SAD、DCT、Quant、Deblocking Filter等模块进行优化，具有很高的编码速度。在PPC和SPARC平台也作了一些优化工作，由于DSP与这些平台的体系结构不同，这些技术在DSP上是不可用的，故删除了这部分优化的内容，同时将HAVE_MMX预编译选项屏蔽。

（2）去除界面显示信息

X264代码中包含了help提示信息、文件操作、用户界面操作和Debug信息等，为了提高编码效率，简化程序结构，去掉了这些功能模块。

（3）去除解码模块

由于DSP上存储空间有限，目前关心的只是H.264编码器端的性能，所以将解码模块去除。为了方便以后测试编码器性能，将编码的4个档次都进行了移植：Baseline Profile，Main Profile，High Profile和High444_Predictive Profile，保留了B帧，CABAC等模块，这样可能会降低编码效率，但是有助于分析。

（4）工程的整合

原始的X264代码包含libx264库和x264应用程序两个工程，为便于向DSP平台移植，将源程序中的众多文件夹和文件结构简化为一个文件夹，一个工程，进行工程整合。

四、H.264的平台移植

X264模型裁剪完成后，即可移植到DM6446平台上。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，故可以使用C语言开发DSP程序，该平台大大方便了DSP程序的开发与调试。移植过程中，在VS2008和CCS3.3中建立X264工程，同步修改，这样方便跟踪代码调试效果。

目前已经完成了DM6446上DSP端X264算法的移植，将继续进行ARM端应用程序和二者的通信架构的开发。下面将重点介绍X264算法在C64x+ DSP上的实现。

1.修改使之符合CCS开发环境

由于X264是基于VS平台的模

数据类型	VS2008	CCS（DM6446）
Bool	8bit	无此类型
Char	8bit	8bit
Short	16bit	16bit
Int	32bit	32bit
Long	32bit	40bit
Float	32bit	32bit
Double	64bit	64bit

表1　两个平台的基本数据类型对比

型，为使之在CCS下成功编译，必须对代码做相应的修改。下面就X264的实际移植过程遇到的重点问题给予说明。

（1）数据类型的不同

C64x+是定点DSP，只支持定点的计算（乘、加）和定点数据类型。如表1所示列举出CCS和VS平台上编译器的基本数据类型之间的差异。

故在CCS中要避免bool类型的使用，如果必须要用，可以用unsigned char代替，本文所使用X264模型没有碰到这个问题。另外需要注意的是long型数据的处理，在编译过程中会报错：int64_t，uint64_t没有定义，X264模型对于数据类型的定义放在头文件stdint.h中，因此对该头文件作如下修改：

typedef long long int64_t;

typedef unsigned long long uint64_t;

（2）数据对齐的问题

数据对齐随处理器和编译器的不同而不同，处理好数据对齐可以可以提高程序的时空效率（节省空间，提高速度），特别对于视频编码这种计算密集型的应用更是如此。在X264中原有的数据对齐是通过下面的宏定义实现的：

#define DECLARE_ALIGNED(var, n) __declspec(align(n)) var

其实这是X86体系中的用法，其中"__declspec"用来指示变量的扩展属性，"align"是VS编译器中的关键字，通过这种方法定义出来的变量都是按指定的方式字节对齐的。但在DM6446开发环境中并不支持这种做法。在CCS中数据对齐需要通过伪指令"DATA_ALIGN_pragma"来实现。因为宏指令和伪指令不能嵌套使用，所以需要在每个数据对齐处重新定义。

如将：DECLARE_ALIGNED_16(int16_t fenc_dct8[4][64]);

修改为：

int16_t fenc_dct8[4][64];

#pragma DATA_ALIGN(fenc_dct8[4][64], 16);

2.修改CCS开发平台的设置

在对X264算法修改后进行编译之前，要正确设置CCS的编译选项（Build Options），有些选项不能直接使用默认设置。

（1）变量的存取方式调整

编译器的内存模式选项（Memory Models）默认选项为Near，此时内存模式为小模式。在小模式下，要求.bss段小于32KB，.bss段用于全局变量和静态变量的存取。但X264代码中的全局变量和静态变量以及视频数据远远大于32KB。这时可将内存模式选项改为Far Aggregate。

（2）"序"的差别

"序"的差别指的是"Big-Endian"或是"Little-Endian"。数据在内存中是按字节存储的。大端模式指的是高位字节先存储，反之，小端模式是指低位字节先存储。DM6446使用的是小端模式，故CCS平台上Endianness选项应该设为little endian。

（3）存储空间的分配

在X264模型中，经常使用malloc函数进行动态内存申请，应该尽量修改为静态分配，例如用数组的形式，方便管理。若必须使用动态分配，一定要注意malloc分配空间之后要使用free函数释放。本文使用的X264版本在运动估计的分析函数x264_mb_analyse_load_costs中使用如下函数：

p_cost_mv[a->i_qp] = x264_malloc((4*4*2048 + 1) * sizeof(int16_t));

该函数分配了64KB左右的空间，应该补充释放内存的代码。如果不释放，会导致堆空间耗费完毕，程序跑飞。

同时在连接命令（cmd）文件中，应该注意堆栈分配的大小，因为X264代码中有很多动态内存的分配，

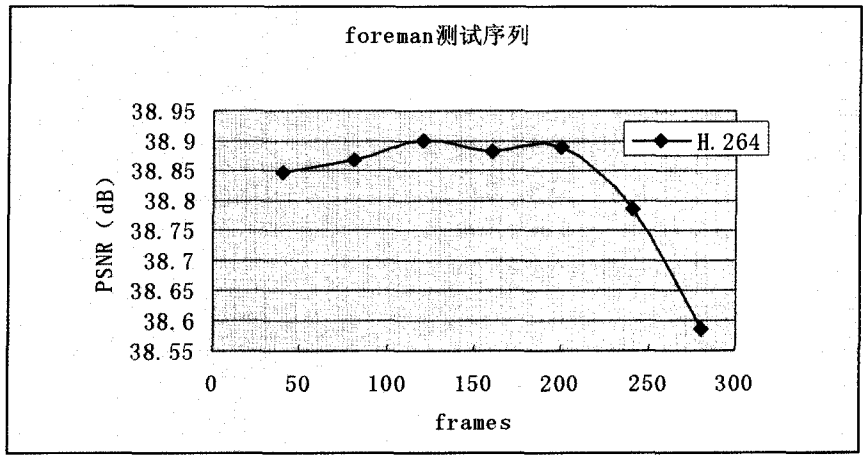

图3 PSNR随编码帧数变化测试结果图

故堆大小应该设置的稍大一点，本文的堆栈配置如下：

-stack 0x0000f000

-heap 0x00800000

另外，在移植阶段最重视的是编码的正确性，所以可以先将CCS中的段分配到DDR2中，这样会降低编码速度，但优化时可以作修改。

（4）编译优化选项

在编译器中提供了分为若干等级和种类的自动优化选项，比如-o2、-o3、-pm和-mt等。根据实际编译的程序，选择合适的优化选项，进行源程序的项目级优化。

五、仿真测试

X264算法移植到DM6446开发板的DSP端后，对标准的foreman QCIF（176x144）序列进行测试，编码采用Main Profile，熵编码同时采用CABAC和CAVLC方式，PSNR随编码帧数和码流变化的效果分别如图3、图4所示。

同时，针对运动强度不同的container，foreman，mobile，stefan QCIF序列采用Main Profile进行编码，固定I帧量化参数为23，P帧量化参数为26，平均编码一帧Encode_frame函数所耗费的CPU周期数如表2所示。

在移植过程中，为便于分析，对编码的4个档次都进行了移植。针对foreman QCIF序列，固定I帧量化参数为23，P帧量化参数为26，测试了不同档次平均编码一帧的编码效率。测试结果如表3所示。

对于上述测试结果均采用JM模型的解码端对编码出来的.264文件解码，验证了编码码流的正确性。

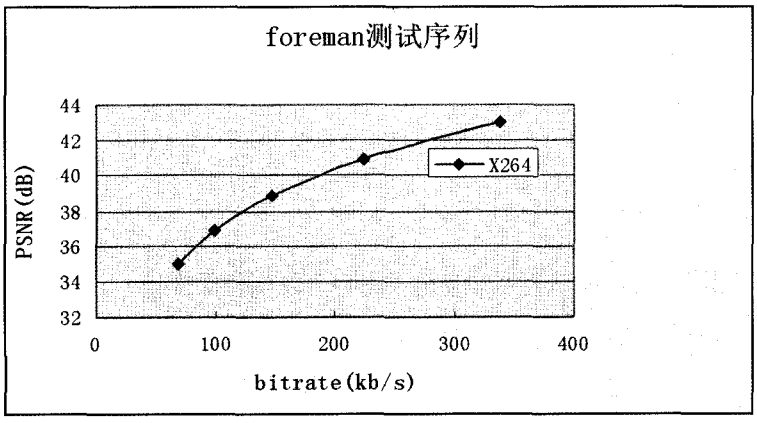

图4　PSNR随帧率变化测试结果图

表2　运动强度不同的序列测试结果

序列	运动强度	PSNR (dB)	Bitrate (kb/s)	CPU.Total.Incl.Total（Encode_frame函数）
Container	Low	39.447	583.15	1424171891
Foreman	Medium	39.820	610.77	1601938580
Mobile	Medium	35.885	595.23	1734151182
Stefan	High	36.334	934.23	1796240764

表3　不同编码档次的测试结果

Profile	PSNR (dB)	Bitrate (kb/s)	CPU.Total.Incl.Total（Encode_frame函数）
Baseline	40.226	706.94	1606955367
Main	39.820	610.77	1601938580
High	40.274	708.36	1344791847
High444_Predictive	40.274	708.36	1078466513

六、结束语

本文从应用的角度出发，讨论了H.264算法实现模型X264在DM6446上的移植过程及应注意的具体事项，并给出了仿真测试结果。由于CCS和VS开发平台的差异以及C64x+自身硬件结构特点，算法移植需根据这些特点进行相应修改。

DM6446的高速的处理能力，丰富的多媒体接口以及完善的双核处理机制为高性能的视频编解码提供了必要条件，如何充分利用这些硬件资源及Codec Engine通信机制，使基于该平台的视频编解码达到所期望的性能仍是值得大家关注的问题。CMMC