H.264 视频解码的 OpenMP 并行优化①

陈 玮，郎 涛

(上海理工大学 光电信息与计算机工程学院，上海 200093)

摘 要：随着 H.264 视频格式得到广泛应用，对 H.264 解码的效率要求越来越高，对 JM15.1 模型中的 H.264 解码过程进行了分析。采用共享存储编程的工业标准 OpenMP 对解码过程进行了并行区的设置。设置私有变量防止数据竞争和调整了负载。解码效率提高了 10%左右。

关键词：OpenMP；JM 模型；H.264 解码；负载平衡；并行优化

OpenMP Parallel Optimization of H.264 Video Decoding

CHEN Wei, LANG Tao

(School of Optical-Electrical and Computer Engineering,University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: With the H.264 video format widely used, the efficiency of the H.264 decoding has become increasingly demanding, this paper analyzes the H.264 decoding process in the JM15.1 model using the OpenMP which is industry-standard of shared memory to set parallel zones for decoding process. It sets private variables to prevent data conflicting and regulating load. The efficiency of decoding increases about 10%.

Keywords: OpenMP; JM model; H.264 decoding; load balancing; parallel optimization

1 引言

H.264[1]是目前最常用的视频编解码方式，由 ITU-T 的视频编码专家组和 ISO/IEC 的 MPEG 专家组联合制定出的视频编码标准。H.264 是在 MPEG-4 技术的基础之上建立起来的，其编解码流程主要包括 5 个部分：帧间和帧内预测(Estimation)、变换(Transform)和反变换、量化(Quantization)和反量化、环路滤波(Loop Filter)、熵编码(Entropy Coding)。H.264 标准分为 3 个档次：基本档次-baseline(视频电话，视频会议，无线通信等)；主要档次-main profile(应用了多项提高压缩比率和提高画质的措施，主要用于 HDTV)；扩展档次-extend profile(主要用于各种基于网络的视频流)。H.264 在混合编码的基本框架下，对关键模块做了重大的改进，如多模式运动估计、帧内预测、4×4 整数变换、1/4 像素精度预测、去波滤块等。H.264 性能的改进是以增加复杂性为代价而获得的。据估计，H.264 编码的计算复杂度大约相当于 H.263 的 3 倍，解码复杂度大约相当于 H.263 的 2 倍。

2 OpenMP标准

OpenMP 标准[2]是共享存储体系结构的线程级并行规范，它是为在多处理机上编写并行程序而设计的 C/C++ 和 Fortan 等的应用编程接口，包含编译制导(Compiler Directive)、运行库例程(Runtime Library)和环境变量(Environment Variables) ，支持增量并行化(Incremental Parallelization) 。OpenMP 主要是在循环级上的并行,指令以单一程序多数据结构、任务共享结构、同步结构以及提供对数据共享或私有的支持扩展了原有程序的顺序结构模型。OpenMP 遵循 Fork and Join 模式(如图 1)，编译制导指令#pragma omp 可以将建立并行域，在并行域内主线程将生成诺干个子线程并行执行，并行区结束，生成的子线程将同步，然后终止，消亡或者睡眠，主线程继续执行。

① 收稿时间:2010-06-27;收到修改稿时间:2010-08-21

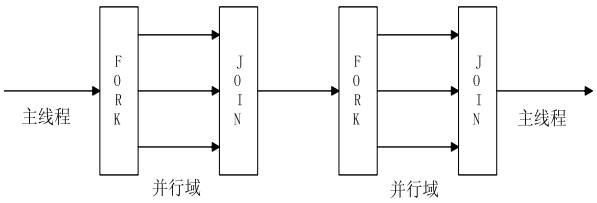

图 1 OpenMP 线程并行示意图

随着 CPU 多核化，意味着计算机体系结构的发展朝着更有利于 OpenMP 的方向发展，可以利用多核 CPU 的并行化来解决大规模计算的问题。

3 解码OpenMP并行优化

3.1 解码器的选择及分析

目前,H.264 开源解码器包括 X264 decoder, JM decoder,T264 decoder,ffmpeg libavcodec 等。JM 模型是 JVT 推出的 H.264 软件参考模型[3],此模型和 ITU-T 标准文档句法表解码流程完全对应，JM 中的 decoder 是一个基于 PC 的软件解码器，主要作用是解释 H.264 协议，测试 H.264 的编解码效率，其代码的功能十分齐全，但是代码的运行效率比较低，它的解码器工作流程图 2 所示：

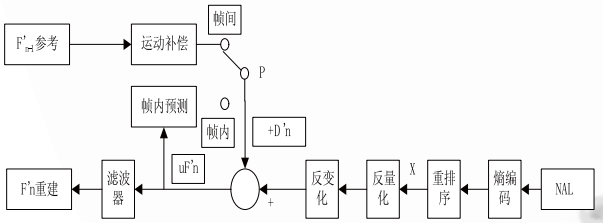

图 2 H.264 解码器框图

在解码过程中，H.264 解码器从网络提取层 NAL 中接收压缩的位流，经过熵解码和重排序产生量化系数 X。又经过反量化和反变换得到 D’n，使用解码的头信息，解码器创建预测块。预测块加上 D’n 产生未经滤波的宏块 uF’n，最后经过滤波器解码宏块 F’n，解出图像。

本文选取了 JM15.1 版本的解码部分并对其进行 OpenMP 并行优化，运用 Intel(R) VTune(TM) Performance Analyzer[4]对 JM15.1 解码过程进行了分析，图 3 给出了解码过程中各个解码模块所占用的时间。

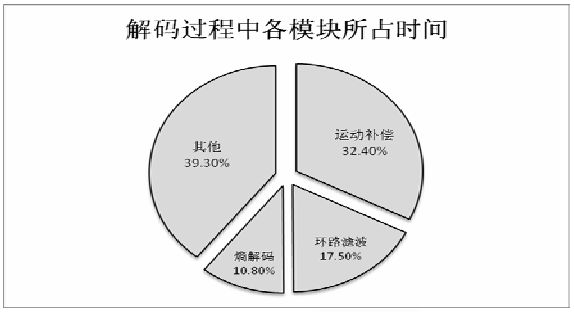

图 3 解码过程模块所占时间示意图

从图 3 可以看出运动补偿，环路滤波和熵解码模块是耗时最多的几个模块，占了总时间的 60.7%，因此，本文工作着重于运动补偿、环路滤波和熵解码模块的并行优化。

3.2 OpenMP 并行优化

3.2.1 并行区的选择，扩展和合并

分析运动补偿、环路滤波和熵解码模块，对其中计算复杂性大的部分设立并行区

扩展并行区，将并行区前面没有内存访问冲突且没有包含在其他并行区中的代码块合并到并行区中。

合并相邻两个并行区，形成一个较大的并行区，增大并行粒度，减少程序中并行区的数目，以减少并行执行和串行执行切换的开销。

3.2.2 共享、私有变量与数据竞争

OpenMP 在并行化循环结构中最重要的是区别共享和私有变量。其中共享变量可以被所有线程访问，而一个线程只能使用自己的私有变量,不能去访问其他线程的私有变量。

在并行区内[5]，共享与私有变量分配出现问题，数据可能同时会被并行区内外多个线程同时访问的情况，这时因为数据被并行区内外几个线程频繁的调度，从而造成堵塞，这种情况称为数据竞争。数据竞争的出现将极大的影响程序运行的效率，所以在并行化的过程中，发现并消除数据竞争是十分必要的。Intel Thread Cheaker 是 Intel 的一种线程检测软件，它能够检测多线程应用程序中存在的关于线程互操作的错误，能够发现看似功能正确的程序中所隐藏的一些问题,而且这些错误会不确定的出现。线程检测器它能识别的问题包括数据竞争,死锁,停止线程,丢失信号以及废弃锁。而且支持采用 OpenMP，POSIX，Windows API 开发的多线程应用程序的分析。用 Thread Cheaker 来

检测并行区内可能的数据竞争的情况。例如下面环路滤波中 EdgeLoopChromaNormal 函数部分代码中用来存放 P 块像素和 Q 块像素的*SrcPtrP，*SrcPtrQ，在滤波时，各线程对其访问很频繁，很容易出现数据竞争现象，因此有必要将其变为私有变量，防止并行区外的线程对其进行访问，消除因为设立并行区而造成的数据竞争现象：

```
#pragma omp parallel for private( yQ ,p_y ,p_pos_y , q_y ,q_pos_y , SrcPtrQ ,SrcPtrP )
//将 SrcPtrQ ,SrcPtrP 变量变为私有变量
num_threads(2)
{
…
    for( pel = 0 ; pel < MB_BLOCK_SIZE ; pel++ )
     {if( (Strng = Strength[pel]) != 0)
…
   SrcPtrQ = &(Img[q_pos_y][pixQ.pos_x]);
SrcPtrP = &(Img[p_pos_y][pixP.pos_x]);  //因为现在并行区内 SrcPtrQ ,SrcPtrP 在并行区内是私有变量，从而消除了数据竞争现象。
…
}
```

3.2.3 任务调度

OpenMP 中的任务调度策略有静态调度、动态调度、指数动态调度和运行时调度。默认情况下，OpenMP 在并行化中使用静态调度，静态调度是将迭代空间分为相等的子块，并按序将每个子块映射到处理器上执行。动态调度与静态调度有些类似，也是将所有循环迭代划分成若干块。但是使用一个内部任务队列，采用先来先服务的方式进行调度。当某个线程闲下来，就为其分配一个循环块。由此可见，动态策略可以极大地保证线程组的负载平衡，但是需要额外的开销。指数动态调度和动态调度有些类似，但是循环块大小刚开始比较大，后来逐渐减小，从而减少了线程用于访问任务队列的时间。运行时调度根据环境变量确定上述调度策略中的某一种，比较灵活，默认值是静态调度。

一般来说，不规则代码、恰当的调度策略都将导致负载分配的不平衡[6]，如果并行区内各个线程负载量相差较大，可能使得 OpenMP 并行的性能退化，所以有必要检测线程的负载，调整负载过重的情况。本文在源程序级对 OpenMP 中并行结构所隐含的同步操作用指令#pragma omp barrier 进行显示化，在并行结构中线程的同步点之后，负载分配之前开始时间的测量，在负载执行完成之后，下一个同步点来之前停止时间测量。期间线程负载的测量结果反映并行区内的各个线程的负载情况。

表 1　OpenMP 同步显示化及测量点代码插入

OpenMP 源代码	隐式同步显示化后代码	插入测量点代码

#pragra omp parallel
{…}
	#pragma omp parallel
	{
	…
	#pragma omp barrier
	}
		#pragma omp parallel
		{
		开始进行测量
		…
		结束测量
		#pragma omp barrier
		}

#pargra omp sections
{
#pragma omp section
{…}
}
	#pargra omp sections nowait
	{#pragma omp section
	{…}
	}
	#pragma omp barrier
		开始进行测量
		#pargra omp sections nowait
		{#pragma omp section
		{…}
		结束测量}
		#pragma omp barrier

#pragma omp parallel for {…}	#pragma omp parallel { #pragma omp for nowait {…} #pragma omp barrier}	#pragma omp parallel { 开始进行测量 #pragma omp for nowait {…} 结束测量 #pragma omp barrier}
#pragma omp single {…}	#pragma omp single nowait {…} #pragma omp barrier	开始进行测量 #pragma omp single nowait {…} 结束测量 #pragma omp barrier

通过测量结果得出在并行区内各个线程负载所用时间的情况，假设所优化的源程序有 n 个同步段（记为 Si，i∈[1,n]），串行执行第 i 号同步段所花时间为 st(i),m 个线程并发执行第 i 号同步段所花时间为 pt(i,m),则可以得到：

同步段并行加速比：

$$SpeedUp\_S(i,m) = st(i) / pt(i,m) \quad (1)$$

$pt(i,m) = \max_{k=0,m=1} pt(i,k)$，pt(i,k)为第 k 号线程执行第 i 号同步段所用的时间。

同步段的并行效率：

$$Eff\_S(i,m) = SpeedUp\_S(i,m) / m \quad (2)$$

由(2)式可以得到各个并行块的并行效率高低，考虑到各个并行块执行时间，如果一个并行块并行效率低，但所占运行时间较少，对其进行负载平衡所花的额外时间开销可能比负载平衡优化后缩短的时间还多，本文对公式(2)作了一些改进。

同步段加权剩余并行效率：

$$WREff\_S(i,m) = (pt(i,m) / \sum_{i=1,n} pt(i,m)) \times (100\% - Eff\_S(i,m)) \quad (3)$$

其中：$pt(i,m) / \sum_{i=1,n} pt(i,m)$ 为权值，表示第 i 号并行同步段占所有同步段之和的比值，100%-Eff_S(i,m)为剩余并行效率，为最大并行效率也就是 100%与第 i 号并行同步段的差值。

通过理论上的最大并行效率与现时并行效率的差值与时间的乘积的结果，由大到小排序，就可以看出哪些并行块有提高并行效率的潜力，然后由大到小采用动态调度或者指数动态调度后在进行测量，检查运用动态调度或者指数动态调度后的额外的线程时间开销是否比负载不平衡时的运行时间少，如果运用后的并行效率有所提高，变换调度策略。

4 测试结果

本文所用的测试平台：系统 Windows XP sp3，CPU 酷睿双核 E8400,主频 3.0GHz,内存 4G,显卡 NVIDIA GeForce GTX 280 显存 1G 的 PC 机,MicroSoft VS2005 开发平台。将优化后的 JM15.1 在 VS2005 上编译，运行，解码 H.264 不同格式的视频流，结果如表 2 所示。

表 2 OpenMP 并行优化前后比较表

格式	分辨率	优化前 (fps)	优化后 (fps)	调整负载后 (fps)	节约百分比 (%)
QCIF	176×144	186.67	201.33	204.54	9.57
CIF	352×288	45.35	48.26	49.82	9.86
D1	720x480	9.28	10.20	10.32	11.21
D4	1280×720	3.61	3.99	4.06	12.53

表 2 可以看出，不同常用视频格式下再经过 OpenMP 并行化和调整负载后，解码效率有了 10%左右的提高。

5 结论

本文通过对 H.264 解码过程的分析，运用 OpenMP 技术对 JM15.1 模型 H.264 解码过程中占用 CPU 时间较多的模块进行了并行区划分，设置私有变量防止数据竞争，调整负载等并行多线程优化操作，使得解码效率有了比较大的提高。

参考文献

1 张杰.视频编解码新标准 H.264/AVC 中的重要技术.现代电子技术,2004,27(6):101－103.

2 OpenMP-Version2.5-Specification.[2010-05-11].http://www.openmp.org/drupal/mp-documents/spec25.pdf

3 Wiegand T, Sullivan GJ, Bjontegaard G. Overview of the H.264/AVC video coding standard, IEEE Transactions Oll Circuits and Systems for Video Technology,2003,13(7):560－576.

4 Intel. Vtune analyzer. [2010-05-07]. http://www.intel.com/software/Products/vtune

5 廖永红.H_264 运动估计中块模式选择的并行设计.计算机科学,2008,35(9):133－135.

6 李建江.OpenMP 源程序级同步段负载监测方法与均衡策略.电子学报,2005,33(5):852－856.