文章编号:1671-4598(2018)07-0276-06 DOI:10.16526/j.cnki.11—4762/tp.2018.07.061 中图分类号:TP3 文献标识码:A

基于 GPU 的 H.264 并行解码优化

汪少锴[1], 李 伟[2], 金燕华[1]
(1. 电子科技大学 航空航天学院, 成都 611731;
2. 国家新闻出版广电总局五四二台, 北京 100866)

摘要: H.264 视频编码标准因其很好的压缩率而成为目前的主流标准之一; 针对 H.264 解码复杂度提高、计算量增大的现状, 根据 GPU 适合通用并行计算的特性, 提出其基于 GPU 的并行解码优化; 使用 GPU 对帧内预测与滤波器模块解码, CPU 负责控制 GPU 以及对剩余部分解码; 通过对帧内预测解码的分析, 提出一种优化的帧内预测并行算法, 经实验证明相比有优化前算法解码效率被提高 20%; 通过对滤波器模块的研究, 提出一种滤波强度并行求取算法以及并行滤波执行算法, 经实验证明滤波器的处理速度提升了 30%, 且相比原图像△PSNR 最大为 0.10, △SSIM 为 0.01; 最终通过实验证明, 使用 GPU 对视频解码的关键模块处理, 能大大提高处理效率。

关键词: GPU 并行解码、帧内预测、滤波强度求取、滤波执行

Parallel decoding optimization of H.264 based on GPU

Wang Shaokai[1], Li Wei[2], Jin Yanhua[1]
(1. University of Electronic Science and Technology, Chengdu 611731, China;
2. The State Administration of Press Publication Radio Film and Television of China, Beijing 100866, China)

Abstract: H.264 video coding standard has become one of the mainstream standards because of its good compression ratio. In order to deal with the increasing complexity and computation in H.264 decoding, it is proposed in this paper that the parallel decoding optimization based on GPU which has the characteristics suitable for general parallel computing. Using GPU to decode intra prediction and filter modules, CPU controls GPU and decodes the remaining part. Based on the analysis of intra prediction decoding, a parallel intra prediction algorithm is also proposed, the experimental results of which show that the intra prediction decoding efficiency is increased by 20%. Through the study of the filter module, a parallel filtering algorithm and parallel filtering execution algorithm are proposed. It is proved by experiments the processing speed of the filter is increased by 30%, and the maximum ΔPSNR is 0.10 compared with the original image, and the ΔSSIM is 0.01. Finally, it is proved by experiments that using GPU to process the key modules of video decoding can greatly improve the processing efficiency.

Keywords: GPU parallel decoding; intra prediction; filtering intensity; filtering execution

0 引言

2003 年, ITU (International Telecommunication Union) 与 ISO (International Organization for Standardization) 组成的 JVT (Joint Video Team) 制定了 H.264/AVC 标准。其中, 为了在保证其图像质量的同时尽可能降低码率, H.264/AVC 标准采用了的许多新创的编解码技术, 如改进的去块效应滤波器、块的多种划分方式、像素运动估计 (帧间预测) 以及根据块划分的多模式帧内预测等; 但是, H.264 解码时的计算复杂度是 MP4 的 2.2～4.1 倍, 并且 H.264 各个对应部分的计算量比 H.263 均高出了 1.2～2.1 倍。

在 H.264/AVC 标准中, 多模式帧内预测与去块效应滤波器两种技术是保证码率与图像质量的关键。在多模式帧内预测技术中, 率失真优化是帧内预测的本质; 为了更好的协调码率与失真, H.264/AVC 标准对所有预测模式进行计算, 选择率失真代价最小的模式作为最优解。其中, 文献 [1] 提出了 9 种不同处理顺序; 采用多级流水线方式, 改变预测块顺序, 使不相关的块可以并行执行, 实现帧内预测模块的并行执行, 但是并没有达到最好的效果; 为了提高并行效率, 文献 [2] 在其基础上改变了预测模式, 这虽然减少了处理时间, 但是所对应的影响则是很大程度上降低了图像质量。

而对于 H.264/AVC 标准中的滤波器, 其主要的作用, 是去除了编码过程中所产生的方块效应。其中, 对于去块滤波器的并行实现, 文献 [3] 直接提出了一种基于 CUDA 的并行实现方法, 文献 [4-5] 提出了通过局部减少分支条件来提高滤波器效率。上述两种算法在一定程度上提高了效率, 但在流程上并没有进行足够的改进, 且在执行顺序上没有很好的突破。

本文主要研究 H.264/AVC 标准中的帧内预测与滤波器模块的并行算法, 在文献 [2] 基础上对帧内预测模块进行优化处理, 对滤波器模块提出新的 Bs 值并行求取算法,

收稿日期:2018-05-04; **修回日期**:2018-05-18。

作者简介:汪少锴(1993-),男,湖南常德人,工程硕士在读,主要从事电子与通信工程 方向的研究。

另外在文献［3］的所提出的奇偶边界执行算法的基础上提出一种新的滤波执行并行算法。

1　GPU 解码框架

FFmpeg（Fast Forward Mpeg）使用的 H.264 解码器，主要通过对帧一级的图像解码来实现多线程并行工作：

1）从网络抽象层（Network Abstraction Layer (NAL)）获取码流数据，将其放入解码和重排序模块进行解析，可得一系列量化系数 X，继续将获得的数据进行反量化与反变换处理，得残差数据 D。

2）从 NAL 获取的码流，通过解析能得出帧内预测模式与帧间预测模式中所需要的 MV 等信息，将此信息作为帧内预测或帧间运动补偿参考数据进行帧内/帧间预测。

3）由残差数据与预测数据相结合得到重建的图像数据，因为图像中存在块效应，所以需要通过滤波器进行滤波，形成真正的解码帧，此解码帧也作为之后解码用的参考帧。

本文仅对帧内预测与滤波器做并行化处理，因此其框架如图 1 所示。

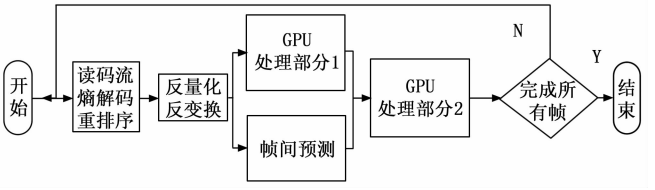

图 1　解码框架图

在 H.264/AVC 标准中，当图像中各个宏块经过反量化与反变换之后，所形成的图像根据规则被分为 I 帧与 P 帧，I 帧是指在编码过程中，预测所采用的是帧内预测，P 帧是指在编码过程中，预测所采用的是帧间预测（运动估计）。而在本文的框架中，主要对 I 帧进行讨论。

GPU 中处理的主要包括两个部分，其中，“处理部分 1”为帧内预测部分；“处理部分 2”为滤波器部分。具体处理流程如图 2 所示。

如图 2 所示，当前帧为 I 帧的时候才通过 CUDA 调用 GPU 进行帧内预测解码，否则当前帧为 P 帧，调用 CPU 进行帧间预测（运动估计）解码。当帧内预测与帧间预测都执行完成后，将两个模块的数据统一传到滤波器模块，调用 GPU 进行滤波模块的处理。

2　帧内预测并行算法优化

在 H.264/AVC 标准中，帧内预测主要包括四种方式：PCM 预测方式、4×4 亮度块的帧内预测方式、16×16 亮度块的帧内预测方式、8×8 色度块的帧内预测方式。

本文主要研究四种模式中的 4×4 亮度块的帧内预测方式，其传统处理顺序如图 3 所示。由图 3 可见，此方式以一个 16×16 亮度块为单位，其中 MB0、MB1、MB2、MB3、MB4 为 16×16 亮度块，每个 16×16 像素的宏块有 16 个 4×4 像素的子宏块，单个 16×16 亮度块内 4×4 像素子宏块的处理顺序如图中字母顺序所标示。

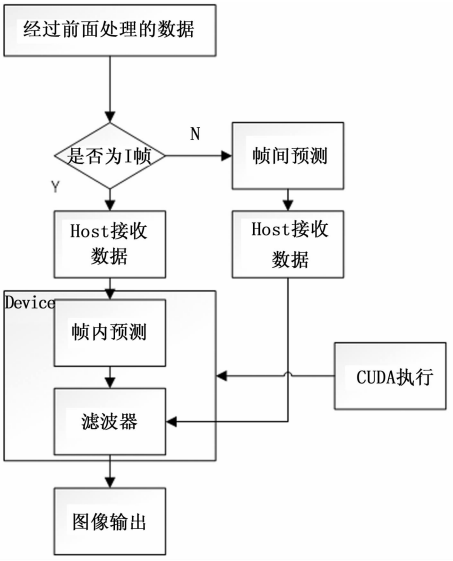

图 2　运行流程图

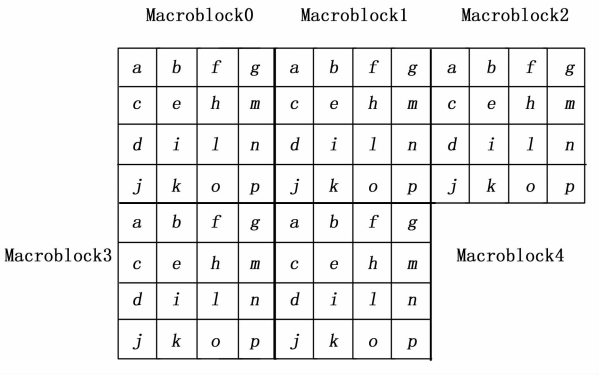

图 3　16×16 亮度宏块图

16×16 像素的宏块 MB4 中子宏块的帧内预测所依赖的全部子宏块分别为：块 1 的 j、k、o、p 子块；块 3 的 g、m、n、p 子块；块 0 的 p 子块；块 2 的 j 子块。以块 4 的 16 个 4×4 子宏块中，需要其他 16×16 块做支持的子块为例：a 子块的预测需要块 0 标记为 p 子块，块 1 标记为 j、k 子块，块 3 标记为 g 的子块；g 子块的预测需要块 4 标记为 f 子块，块 1 标记为 p 子块，块 2 标记为 j 的子块；j 子块的预测需要块 3 标记为 p 子块，块 4 中标记为 d、i 的子块。而对于 16×16 宏块的执行的顺序为 0～4，每一个 16×16 宏块中的 4×4 子块的执行顺序则是为 a～p。

若直接将传统的帧内预测解码顺序在 GPU 上实现，那么传统模式所采用的固定顺序会导致 GPU 的运行效率极低。为了解决该问题，文献［1］中提到了 9 种更改的处理顺序，文献［2］在其基础上从中选择一种处理顺序并在其基础上提出一种流水线解码方案，该方案的并行处理顺序如图 4，其中数字为并行处理顺序，数字相同的则是可以被并行处理的子块。

该方案在原先基础上将 4×4 块的处理顺序进行了较好的并行化处理，对一个 16×16 的宏块，总计需要 10 个计算单个子块的时间；本文在文献［1－2］基础上，提出了一

1	2	3	4
3	4	5	6
5	6	7	8
7	8	9	10

图 4 文献［2］中 4×4 块处理顺序图

种以 4×4 亮度块为最小单位、去除模式 3 与模式 7 的并行优化方案，如图 5，图中数字仍为并行处理顺序，数字相同的是可以被并行处理的子块；每行的间隔从 2 减少为 1。

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

图 5 本文 4×4 块处理顺序图

如图 5 所示，本文算法的并行粒度进一步提高，使得每行的宏块都能流水线的进行并行处理，对于一帧内的 16×16 块中属于同一行的 4×4 块进行顺序处理，对一个 16×16 宏块，总计仅需要 7 个计算单个子块的时间。

假定一帧图像的宽为 W、高为 H；对于所处理的块，其基础单位为 4×4 像素块。使用本文所提出的执行顺序，每一帧 I 帧至少需要 GPU 对其进行 M 次运算；而 N 是该执行顺序所能达到最大理论并行粒度。M、N 计算方式如式（1）～（2）：

$$M=(H/1-4)+W/4 \tag{1}$$

$$N=\min(W/4,2\times(H/4)) \tag{2}$$

每次运算所能同时进行的宏块数量由 $list[i]$ 表示。$list[i]$ 计算方式如式（3），其上限即为 N：

$$list[i]=\begin{cases} i+1 & 0\le i<N \\ (M-i-1)+1 & M-N\le i<M \\ N & N\le i<M-N \end{cases} \tag{3}$$

因为编码阶段仍然会将模式 3 和模式 7 作为预测模式，若在解码阶段直接去除模式 3 与模式 7 会使得最终的图像质量受到影响。因此，为了保证最终图像质量，本文在进行帧内预测并行处理的同时，选择添加一个分支的方式，此分支可以在基本不影响并行粒度的同时，保证模式 3 和模式 7 的正常解码，其处理流程如图 6。

如图 6 所示，该分支主要作用是为了防止模式 3 和模式 7 的出现，等待前一个线程的宏块处理，使得当前模块所需要的左上模块得以被处理完成。

3 滤波器并行算法优化

3.1 滤波原理

1）在 H.264/AVC 标准中，图像边界分为虚假边界、

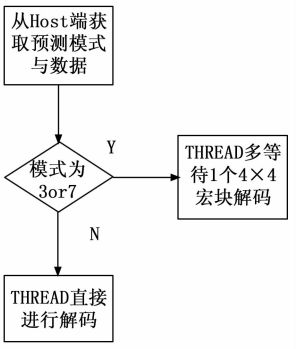

图 6 处理顺序分支流程图

真实边界，根据式（4）～（6）辨别图像边界伟哪种边界。其中，通过查 QP 表得出 α 和 β 的值。真实边界是指所求值不满足以下条件；虚假边界是值满足以下条件；滤波器将对存在虚假边界的宏块进行滤波。

$$|p_0-q_0|<\alpha \tag{4}$$

$$|p_1-q_0|<\beta \tag{5}$$

$$|q_1-q_0|<\beta \tag{6}$$

2）滤波强度 Bs 值是根据图 7 所描述规则来求取。在滤波的过程中，分别会对图像进行水平滤波、垂直滤波，因此要分别求取水平边界与垂直边界的 Bs 值。

Bs 值的求取规则如图 7 所示：

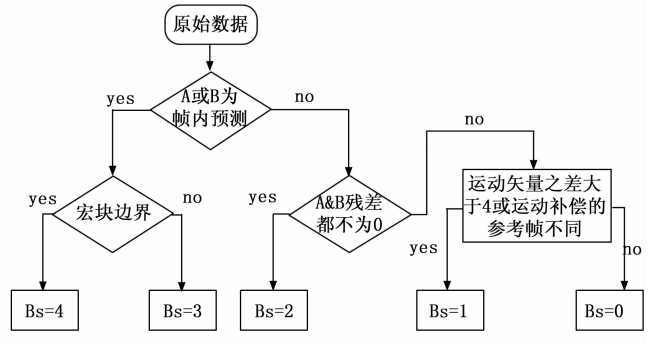

图 7 Bs 值确定规则图

3）虚假边界进行滤波操作，其中 Bs 值越大，滤波越强。

3.2 滤波强度 Bs 值求取的并行实现

3.2.1 Bs 值的并行求取

滤波强度 Bs 值求取的过程是以 16×16 块为计算单位，一帧图像的边数为（$W/H\times H/16$）×8=$W\times H/32$ 条，共有 $W\times H/32\times 16=W\times H/2$ 个边界点。这些边界点求取所需的值已经在之前的过程中获得，因此没有相关性，可以并行求取。

在滤波强度 Bs 值的求取过程中，总计需要的参数为：熵解码所解出的 CBP 值、运动矢量 MV、编码过程中的整数变换矩阵系数、反变换后所得残差数据、帧内预测模式等；在熵解码中，CBP 值与整数变换矩阵系数已经被转换为 CBP _ BLK 值。

本文将这些参数存放在全局存储器中，不需要额外调用 CPU 相 GPU 传递这些参数；另外需要额外处理的参数为 CBP _ BLK 数组，将其传入到 GPU 内的纹理存储器中，以便于减少读写访问；同时给 GPU 分配一定大小的空间用以存放各个边界点的滤波强度 Bs 值，以便于滤波执行时的直接调用。

在 GPU 中的每个 block 所处理的最小单位为 16×16 像素块，即 1 个 block 需对 128 个边界点进行处理；本文为每个点分配一一对应的 thread 进行计算，以保证读写的效率与精准度。线程调度方式为：

$$grid(W/16,H/16),block(16,4,2)$$

即将一帧分为 $W/16\times H/16$ 个 *block*，每个 *block* 负责 16×4×2 个 *thread*。

3.2.2　优化的并行求取算法

对于同一个 4×4 像素块中的运动矢量 MV、帧内预测模式、反变换后所得残差数据以及 CBP _ BLK 等参数是一样的，因此根据同样参数所计算出的 Bs 值也应该相同。基于该情况，本文对进一步优化 Bs 值求取算法，其所需的存储空间与时间都为原来的 1/4，计算效率为原来的 4 倍。优化后的线程调度为：

$$grid(W/16,H/16),block(4,4,2)$$

即将一帧分为 $W/16\times H/16$ 个 *block*，每个 *block* 负责 4×4×2 个 *thread*。算法执行时，仅需要对同一个 4×4 像素块的多个边界点进行一次计算，所计算出的 Bs 值就是该宏块所有全部边界点的滤波强度 Bs 值。

3.3　环路滤波执行的并行优化

对于传统的滤波执行如图 8 所示，滤波首先对从水平方向对各个垂直边界进行滤波，执行的顺序从 1～4；然后从垂直方向对各个水平边界执行滤波，执行顺序为 5～8。

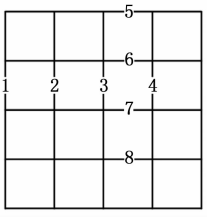

图 8　传统滤波执行顺序图

从 H.264/AVC 标准中的滤波原理所知，块与块间的滤波是没有关联性的，其所需要的像素点都已经在之前的过程中被计算出，官方所给出的 H.264/AVC 标准给出顺序是为了提高数据访问的精确性，以便于串行操作；受文献［3］中对并行滤波执行的启发，彻底的打破原先的执行顺序，本文提出一种优化环路滤波并行算法。

该算法在奇偶边界并行滤波算法基础上进一步的提高并行粒度：首先从水平角度执行，对一个 16×16 块的全部垂直边界同时进行滤波；然后从垂直角度执行，对一个 16×16 块的全部水平边界同时进行滤波。因为是先水平后垂直的执行顺序，并不会对存储器的读写造成访问冲突。

在本文提出的并行滤波算法中，以 16×16 块为单位进行滤波，1 个 block 对应处理一个 16×16 块；由于从水平角度到垂直角度的滤波顺序是串行的，所以每次滤波过程仅并行的处理 4 条边界，对于一个 16×16 块的每条执行边界有 16 个像素点，为得到更好的并行粒度，为每个边界一一对应的分配 thread。其 thread 与边界点所对应的关系如图 9。

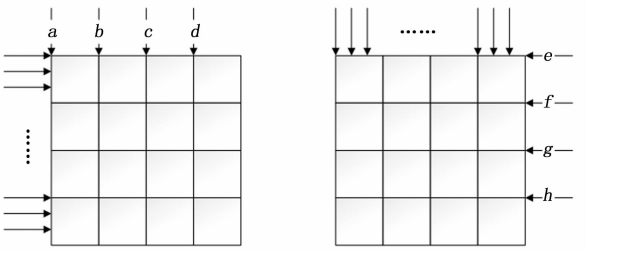

图 9　边界滤波中线程和边界点对应关系图

以图 9 为例，边界 *a*、*b*、*c*、*d* 子边界的滤波所分配的 64 个线程进行处理，当完整的一轮滤波执行完后，再由同样的 64 个线程对边界 *e*、*f*、*g*、*h* 子边界滤波进行处理。这样能保证各个边界点之间为线性的一对一存储访问，从而提高读写的精准度。

在 GPU 上的内核函数线程调度如下：

```
for（i=0；i<2；i++）{
filter<<<blocks（W/16，H/16，1），threads（16，4，1）>>>
}
```

在本文算法执行前，所需要的参数已经经 Bs 求取模块处理完成并存放在了 GPU 的本地存储器中，因此没有额外的参数传递时间损耗。当执行完全部滤波后，调用 GPU 将最终所计算出的图像传回 CPU 中。由于本文提出的算法完全打破了原有的执行顺序，因此，最终的图像结果可能会与串行解码所得的图像结果有略微不同。

4　仿真实验

实验的运行环境采用的 GPU 是 NVIDIA GeForce GTX 950 显卡，CPU 的型号为 Inter　Core™ i5－4590 处理器，操作系统是 windows 10，编译环境为 CUDA8.1。

实验中，选取了四种不同尺寸的 H.264 视频码流，分别为 Foreman _ QCIF 序列（176＊144）、Mobile _ 625SD 序列（720＊576）、Parkrun _ 720P 序列（1280＊720）、Tortoise _ 1080P 序列（1920＊1280）。在实验过程中取每个序列 200 帧亮度帧进行 50 次实验，计算平均每帧耗时。

4.1　帧内预测并行算法的并行优化实现

本文算法、文献［2］中算法以及 FFmpeg 算法对比如图 10 所示。

由图 10 可知在四种序列下，序列的像素点越多，并行加速效果越明显。其中本文所提出的帧内预测与文献［2］中并行帧内预测相比有 1.2～1.4 倍的加速效果，这是因为在编码中去掉了模式 3 和模式 7，使得并行架构更紧凑合

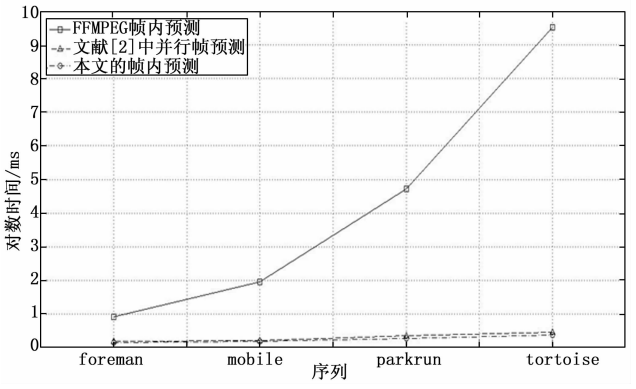

图 10 帧内预测解码对比图

理；且因加入了分支判断，对图像质量没有影响。

4.2 滤波强度 Bs 值求取

本文算法、文献［4］中算法以及 FFmpeg 算法对比如图 11 所示。

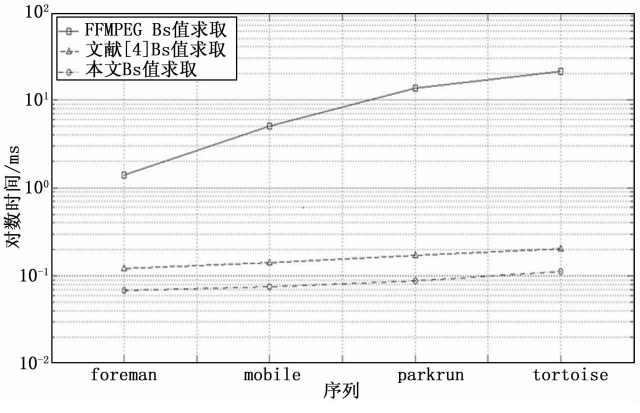

图 11 滤波强度 Bs 值求取对比图

由图 11 可知，随着被处理图像尺寸的增加，加速效果也随之提高。本文所提出的对滤波强度 Bs 值并行求取算法的优化表现出了明显的效果。虽然原并行算法的加速比已经很明显，本文算法比文献［4］中并行算法也有 1.8～2 倍的明显加速比。

4.3 滤波执行

本文算法、文献［5］中算法以及 FFmpeg 算法对比如图 12 所示。

由图 12 可知，本文所提出的对全并行滤波执行算法有很明显加速的效果，虽然原并行算法的加速比已经很明显，但是经过优化后的算法比文献［5］中并行算法也有 1.2～1.5 倍的明显加速比。对比 1280×720 的 parkrun 序列，1920×1280 的 tortoise 序列，可以看出尺寸更小的 parkrun 序列的加速比反而比尺寸更大的 tortoise 序列更高，这是因为 parkrun 序列中的运动过程更多；具体原因是在 CPU 串行执行时候，所有的边界处理时间之和才是环路滤波所消耗的运算时间，对运动越剧烈的图像序列，所需要的边界点数规模更大。

为了对最终图像质量进行比较分析，本文采用两种评价标准 PSNR 与 SSIM，对形成的图像进行比较分析。

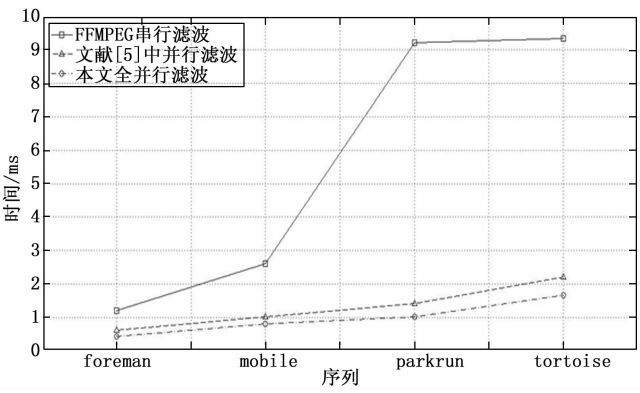

图 12 滤波执行对比图

表 1 串并行成像效果对比

序列		串并行△PSNR	串并行△SSIM
foreman	Y	−0.01	−0.01
	U	+0.01	0.00
	V	+0.01	0.00
mobile	Y	0.00	−0.01
	U	−0.02	0.00
	V	−0.01	+0.01
parkrun	Y	0.00	0.00
	U	−0.10	−0.01
	V	−0.02	0.00
tortoise	Y	+0.03	0.00
	U	+0.04	0.00
	V	−0.02	0.00

由表 1 可知，PSNR、SSIM 两种标准中，四种不同尺寸的序列中，仅存在十分微小的差异；所以本文提出的并行滤波算法在有很大加速比的同时，也不会对成像效果有很大影响。

5 结束语

本文基于 GPU 分别提出了一种的帧内预测并行算法与一种滤波器并行算法；帧内预测算法主要对文献［1，2］算法进行了优化，实验中优化后的算法是原算法加速比的 1.2～1.4 倍，加速效果明显；本文将滤波器分为两个部分，对于 Bs 值求取是在文献［4］基础上做出了优化，对滤波执行在文献［3］基础上提出一种全并行算法，使得滤波器加速优势更明显。通过实验可以看出 GPU 非常适合进行视频解码处理。

参考文献：

［1］Smaoui S，Loukil H，Atitallah A B，et al. An efficient pipeline execution of H.264/AVC intra 4×4 frame design［A］. International Multi－Conference on Systems Signals and Devices［C］. IEEE，2010：1－5.

[2] Orlandic M，Svarstad K. A low complexity H.264/AVC 4×4 intra prediction architecture with macroblock/block reordering [M]. 2013.

[3] 梁　霞. 基于 GPU 的 H.264 并行解码器设计 [D]. 大连：大连理工大学，2010.

[4] 刘　虎，孙召敏，陈启美. CUDA 架构下 H.264 快速去块滤波算法 [J]. 计算机应用，2010，30 (12)：3252－3254.

[5] Su H，Zhang C，Chai J，et al. A efficient parallel deblocking filter based on GPU：Implementation and optimization [J]. 2011，42 (4)：280－285.

[6] Tonfat J，Kastensmidt F，Reis R. Energy efficient frame－level redundancy scrubbing technique for SRAM－based FPGAs [A]. Adaptive Hardware and Systems [C]. IEEE，2015，1－8.

[7] Tung D M，Dong T L T，Anh T T. An efficient parallel execution for intra prediction in HEVC Video Encoder [C]. International Conference on Computing，Management and Telecommunications：IEEE，2014，233－238.

[8] Garland M，Grand S L，Nickolls J，et al. Parallel Computing Experiences with CUDA [J]. Micro IEEE，2008，28 (4)：13－27.

[9] Benmoussa Y，Senn E，Derouineau N，et al. Joint DVFS and Parallelism for Energy Efficient and Low Latency Software Video Decoding [J]. IEEE Transactions on Parallel & Distributed Systems，2018，PP (99)：1－1.

[10] Pastuszak G，Trochimiuk M. Architecture design of the high－throughput compensator and interpolator for the H.265/HEVC encoder [J]. Journal of Real－Time Image Processing，2016，11 (4)：663－673.

[11] 毕厚杰，王　健. 新一代视频压缩编码标准：H.264/AVC（第 2 版）[M]. 北京：人民邮电出版社，2009.

（上接第 275 页）

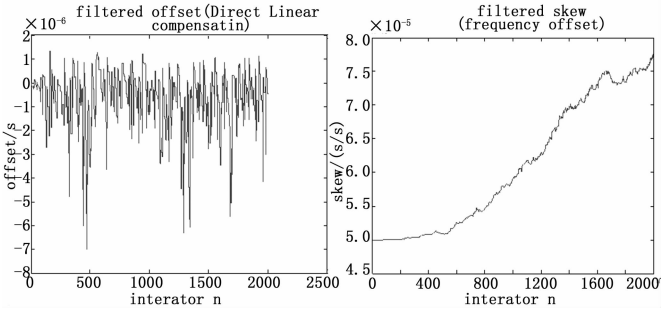

图 10　线性渐进补偿结果

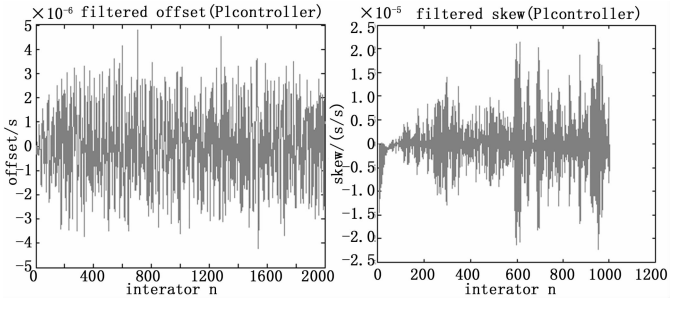

图 11　PI 控制器的补偿结果

5　结论

本文在深入分析 TTE 时钟同步协议的基础上，总结了影响时间同步的因素，分析了时钟同步过程中存在的时间不确定性，建立了时钟噪声模型，根据噪声的统计特性采用了 kalman 滤波等三种滤波算法，得到了较好的效果。最后在滤除噪声的影响之后，采用了两种渐进时钟补偿方法对于时钟进行补偿，并通过仿真分析验证了算法的可行性。

参考文献：

[1] SAE. ARINC 664 Aircraft data network part 7：Avionics Full Duplex Switched Ethernet (AFDX) network [S] [s. l]：ARINC 2005.

[2] CHARARA H，SCHARBARG J，ERMONT J，et al. Methodsfor bounding end－to－end delays on an AFDX network [C]. IEEE 18th Euromicro Conference on Real－Time Systems 2006：192－202.

[3] Abubakari H，Sastry S. IEEE 1588 Style Synchronization over Wireless Link [A]. International IEEE Symposium on Precision Clock Synchronization for Measurement [C]. ISPCS，2008：22－26.

[4] SAE AS6802. Time－triggered ethernet [S]. Society of Automotive Engineers，2011.

[5] Zucca C，Tavella P，The Clock Model and Its Relationship with the Allan and Related Variances [J]. IEEE Trans. Ultrason. Ferroelect，Freq. Contr，2005，52 (2)：289－295.

[6] LEMMON M，GANGULY J，XIA L. Model－based clock synchronization in networks with drifting clocks [A]. Proc. Of the Pacific Rim International Symposium Dependable Computing [C]. 2000：177－184.

[7] 任丰原，董思颖，何　滔，等，基于锁相环的时间同步机制与算法 [J]，Journal of Software，2007，18 (2)：373－380.

[8] Liu L，Qing Y Z，GPS Analysis of noise type of satellite clock [J]，Global positioning system，2005，30 (2)：27－29.

[9] Zhong H D，Zhou C X，Kong R Y. Error analysis of time synchronization algorithm for Wireless Sensor Networks [J]. ensors and instruments. 2008，(24)：151－512.

[10] 章翠枝. 基于 WIFI 的时钟同步技术研究 [D]. 杭州：浙江大学，2012.

[11] Hill J D. Culler A Wireless Embedded Sensor Architecture for System－level Optimization [D]. Technical report，U. C. Berkeley，2001.

[12] Jamalipour A，Zheng J. Wireless Sensor Networks：A Networking Perspective [M]. Wiley－IEEE Press，2009.