文章编号:1001-2486(2011)03-0066-06

# 流化 H.264 编码的数据并行方法*

文 梅,任 巨,伍 楠,苏华友,荀长庆,张春元
(国防科技大学 计算机学院,湖南 长沙 410073)

**摘 要**:流模型是一种源于媒体处理的新型并行计算模型。然而 H.264 编码作为媒体处理领域中的重要应用,在与流模型适配时,却出现重要障碍,主要是由于相关性问题极大地限制了流化过程中数据级并行的开发。针对这个问题,首先根据编码算法分析了编码过程中遇到的相关性问题,然后为各主要编码模块分别提出解除相关性限制的数据并行编码方法。采用这些方法能够保证大数据并行粒度,最终达到高效并行流化编码的目的。实验表明采用数据并行流化编码方法,帧间预测、帧内预测、熵编码和去块滤波模块都获得了显著加速。

**关键词**:H.264 编码;流模型;并行计算;数据并行
**中图分类号**:TP391 **文献标识码**:A

# Data Level Parallel Method of Streaming H.264 Coding

*WEN Min, REN Ju, WU Nan, SU Hua-you, XUN Chang-qing, ZHANG Chun-yuan*
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

**Abstract**: Stream processing model is a newly emerging parallel processing model, which is derived from media processing. However, as an important application of media processing, H.264 coding encounters problems during its streaming procedure. The main reason is that dependency limits the data level parallel exploitations. To solve this problem, data level parallel methods for different coding modules were developed by the analysis of coding algorithm. By adopting the method, key coding modules of H.264 can attain large data parallel granularity. Experiment results show that the data level parallel streaming inter/intra prediction, CAVLC, and the deblock filter can achieve outstanding acceleration.

**Key words**: H.264 coding; stream processing model; parallel processing; data level parallel

ITU-T 视频编码专家组(VCEG)和 ISO/IEC 运动图像专家组(MPEG)于 2003 年共同开发出新一代视频编码标准 H.264[1]。该标准在充分吸取以往成功的编码标准的基础上,致力于在相同的重建图像质量的前提下,获得比其它标准更高的压缩效率,使之适合于无线或者互联网中的实时视频应用。在同等图像质量下,H.264 的压缩效率比 H.263 标准提高近一倍,与 MPEG 相比,则可以节省 50%~70%的比特率。但是,H.264 的编码性能是以计算复杂度为代价的。根据对 HDTV1024P(2048×1024, 30fps)规格的指令 profile 显示,H.264/AVC 解码需要 83GIPS(GIPS, Giga Instructions Per Second)的计算性能和 70GB/s 的访存性能。对于 HDTV720P(1280×720,30fps)规格的编码器,达到了 3600GIPS 和 5570GB/s 的性能需求[2]。H.264 编码的算法复杂度高,计算需求大,对于高清标准则更是如此。这对大多数现有的可编程处理器都是一个极大的挑战。

流模型[3]是一种新型的并行计算模型,在很多计算密集的领域特别是媒体/图像处理和科学计算领域,都展示了惊人的计算效率[4]。这为软件可编程处理器上加速应用处理提供了一个有效的方法。在一些新兴并行处理器(例如流处理器、GPU、cell 等)上,流化后的应用性能可与专用的 ASIC 相比,这大大避免了为专门设计 ASIC 而耗费大量的 NRE(Non-Recurring Engineering)成本和时间成本。

但是,在 H.264 编码算法的流化过程中,存在一个关键的障碍:那就是流模型以暴露数据级并行为主,采用 SIMD(单指令流多数据流)并行执行模式,而且要尽量避免短流效应,但是 H.264 编码中,相关性广泛存在于预测、熵编码和去块滤波等

* 收稿日期:2010-07-30
基金项目:国家自然科学基金资助项目(60703073,60903041,61033008)
作者简介:文梅(1975—),女,副教授,博士。

各个编码模块中,严重限制数据并行粒度(即流长),大大降低并行效率。针对这个问题,本文首先根据其算法分析了编码过程中遇到的相关性问题,然后为各主要编码模块分别提出解除相关性限制的具体方法,保证大数据并行粒度,最终达到并行编码的目的。

## 1　流模型

流模型中的基本数据单位是流,流是一种有序的任意数据类型的数据集合。流模型里的操作都是对整个流的操作。包括 streamgather、streamscatter、形式为 kernel 的计算[5-6]。一个普通程序转化为流程序,需要将数据组织成流,并将计算过程划分为多个 kernel,称为流化。对流中的每个元素做的运算是以 kernel 为单位的 SIMD 处理,而不是简单的运算,因此流元素可以是一个数据块,甚至是一个宏块或一帧图像,这极大地暴露了数据局域性和数据并行性。但是,另一方面,高效的数据并行建立在数据存取与计算的解耦合基础上,即数据必须通过 streamgather/streamscatter 预先组织好。由于这些操作以及 kernel 的启动都需要一定的时间开销,如果流长过短,则无法摊销,造成短流效应[7],因此要保证一定的流长,即数据并行粒度。

## 2　相关性分析

本节主要关注 H.264 编码中影响流化数据并行的各类相关性因素,将它们归纳为数据相关、码流存储的优先约束、控制相关。

(1)数据相关。本节按产生相关的数据粒度来划分。

a. 帧间数据相关。例如图 1(a)所示,当前帧进行帧间预测时,需要使用参考帧中部分区域的像素数据作为输入,因此当前帧和参考帧之间具有数据相关。

b. 宏块间数据相关。例如图 1(b)所示,在帧内预测中,当前宏块的部分预测模式(如 16 × 16 亮度预测)需要以其左、左上、上和右上 4 个相邻宏块中的边缘像素数据作为预测的依据。

c. 块间数据相关。例如图 1(c)所示,帧内预测的某些预测模式(如 9 种 4 × 4 亮度预测)中,需要用到相邻块的边缘像素数据。去块滤波对某些块滤波时,需要使用左和上邻块的数据,在对某些块滤波时,则需要上、下、左、右 4 个邻块的数据。

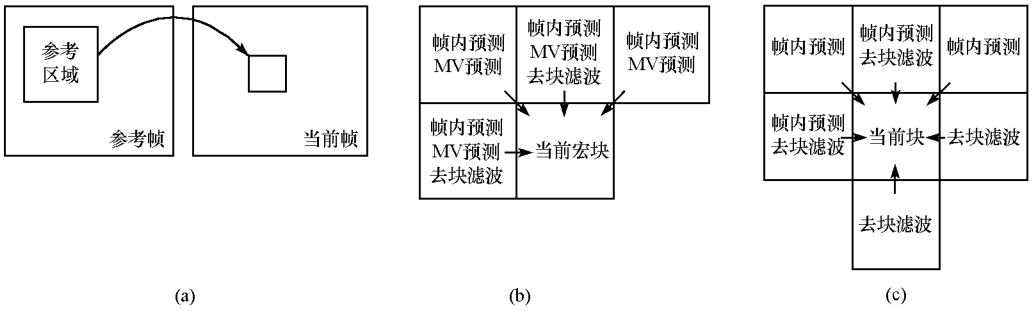


图 1　H.264 编码中的数据相关

Fig.1　Data dependency in H.264 encoding

(2)码流存储的优先约束。例如在基于上下文的自适应可变长编码 CAVLC(Context-based Adaptive Variable Length Coding)中,由于是变长编码,各个块中语法元素的个数和编码长度是不可预知的,因此每个块生成的码流长度也是不可预知的,而且块与块之间的码流是紧密衔接的。所以,在前一块没有生成码流以前,后一块因为不能预知写码流的起始位置而无法向码流结构中存储,只有等待前一个块的码流生成并存储结束以后,才能接着开始后面块的码流生成。

(3)控制相关。指不同的输入数据导致编码程序进入不同分支执行的情况。例如去块滤波中,不同边界强度对应不同滤波过程。

这些相关与流模型采用的 SIMD 并行方式相矛盾。SIMD 要求在不同数据上执行相同的指令,达到并行处理的目的。然而,数据相关和码流存储的优先约束都可以视为当前计算指令所需的并行数据不能同时准备就绪;而控制相关则由于执行路径的不同导致有待并行处理的数据面临不同的指令。另外一方面,这些相关性在处理顺序上形成了优先约束,将 H.264 编码限制在串行执行模式内,数据处理粒度低,例如 x264 的主要编码核心循环体均以块为基本数据处理粒度。而低数据处理粒度则导致了短流效应。

## 3　数据并行方法

本节主要针对上一节分析的各类相关,讨论通过解除相关性的约束来挖掘数据并行的方法。

由于 H.264 编码算法本身很复杂,有的模块同时包含多种相关,难以面面俱到,因此我们分别以主要模块预测编码、熵编码和去块滤波的典型数据并行算法来说明。

### 3.1 预测编码

(1)帧间预测的宏块级并行

帧间预测存在宏块间的数据相关,出现在运动矢量预测部分。当前块的运动矢量预测方式有三种形式,如图 2 的上半部分所示,每一种预测方式中当前块的 MV 预测值都依赖于邻近宏块的 MV 值。只有邻近宏块的 MV 值确定以后,当前宏块才可以继续后面的计算。这导致宏块间难以并行,例如图中的宏块 MB-P0 和 MB-C 之间存在数据相关,因此不能并行处理。帧间预测的并行思路是,通过改变相关关系的方向,消除对并行的限制,形成宏块级并行。具体解决方法是将计算所需的当前宏块左边的宏块的运动矢量 MV0 都转变为左上角宏块的 MV3。如图 2 的下半部分所示,在第一种方式里,MV 的预测值由原有的 MV0、MV1、MV2 的中值变成了 MV3、MV1 和 MV2 的中值,显然 MB-P0 和 MB-C 能够被并行处理,这样图像中同一行的宏块之间都可以并行地进行帧间预测。

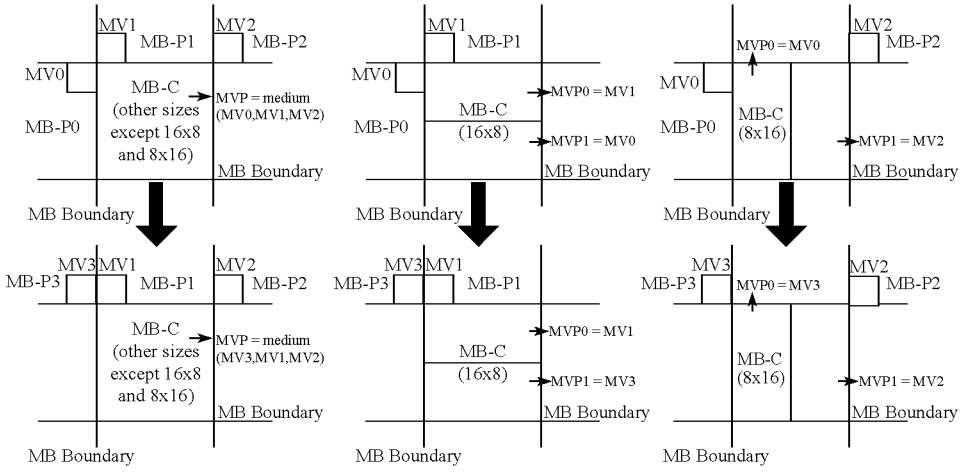


图 2 帧间预测宏块间相关性消除示意图

Fig.2 Eliminating MB dependency in inter prediction

这种方法只改变了针对运动估计补偿因子的计算,仍然符合 H.264 标准。对 Claire QCIF 10Hz 序列的测试表明,改进后的帧间预测的 PSNR 与原算法相比平均只降低了 0.4dB。

(2)帧内预测的块级并行

帧内预测包含多种预测模式,其中 4×4 亮度帧内预测过程存在块间数据相关。虽然 4×4 帧内预测的 9 种模式使每个块同 4 相邻块有相关关系,但是并非每两个 4×4 块之间都会产生直接相关,因此帧内预测的并行思路是并行执行不存在相关关系的块。

图 3(a)示例了一个宏块内各个 4×4 块之间的相关关系,图中标号代表相应的块,箭头表示相关关系。如图所示,显然存在很多不直接相关的“块对”,例如由块 0 和块 4 组成的块对(0,4)等,但是由于依赖的传递性,无直接相关的块对仍然可能具有间接相关,例如,块对(0,4)的两个块因为块 1 的传递而具有间接相关关系。

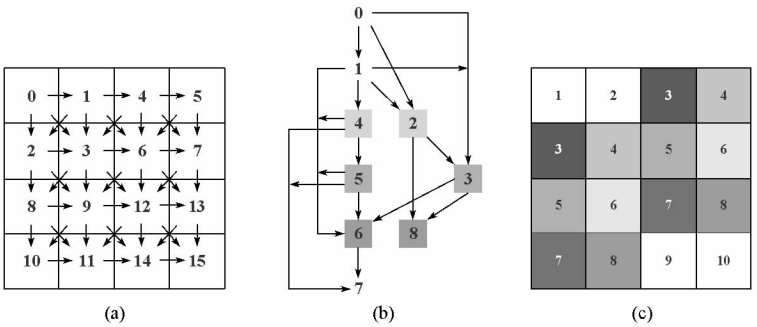


图 3 帧内预测宏块内部各 4×4 块间的相关关系及十阶段块并行方法

Fig.3 4×4 block dependency and 10 stages parallel intra prediction

块级并行的基本思想就是将不存在相关关系的块并行执行,这里的相关关系包括直接和间接

两种相关关系。因此,块级并行的关键问题是寻找无相关关系的块。为了明晰地表示宏块中各块的相关关系,按照“点火方式”进行适当的变化,可以得到图 3(b)。这里的点火方式是指一旦某个块的相关关系全部得到满足,立刻执行该块。图 3(b)示例了块 0 至块 8 的相关关系,可见块对(2,4),块对(3,5)以及块对(6,8)都是无相关关系的块,这些块对中的两个块可以在同一时刻开始执行预测,即处于图中相同水平层次的块可以并行执行预测。按照上述方法,一个宏块中可以并行执行的块对共有 6 对,分别是块对(2,4)、块对(3,5)、块对(6,8)、块对(7,9)、块对(10,12)和块对(11,13)。

按照这些块对形成的块级并行预测方案如图 3(c)所示。此方案经历 10 个阶段,因此称为十阶段块并行法。图中的数字标号 $n$ 表示该块在第 $n$ 个阶段执行。其中阶段 1、2、9、10 这四个阶段将只执行一个块的预测,而阶段 3 至阶段 8 则有 2 个块并行预测。此方案的执行时间是原来的 16 阶段串行时间的 62.5%。这种并行方式能开发的数据并行度为 2。

### 3.2　熵编码

在熵编码算法 CAVLC 中,采用上下文自适应的算法,计算过程与输入数据的特征相关;另外各块间的码流紧密连接,使各块的码流存储形成优先约束型相关。图 4(a)示例了宏块间的码流拼接情况。设宏块 MB0 至 MB2 的码流长度分别是 18bit、12bit 和 14bit,因此 MB0 码流的最后一个字节只写 2 个比特位,那么宏块 MB1 的码流首先要填充该字节中剩余的 6 位,然后再从下一个字节开始存储剩下的编码,结果又在码流的第 4 字节余下 2 个空位。宏块 MB2 以前述方式继续按比特位完成码流存储。

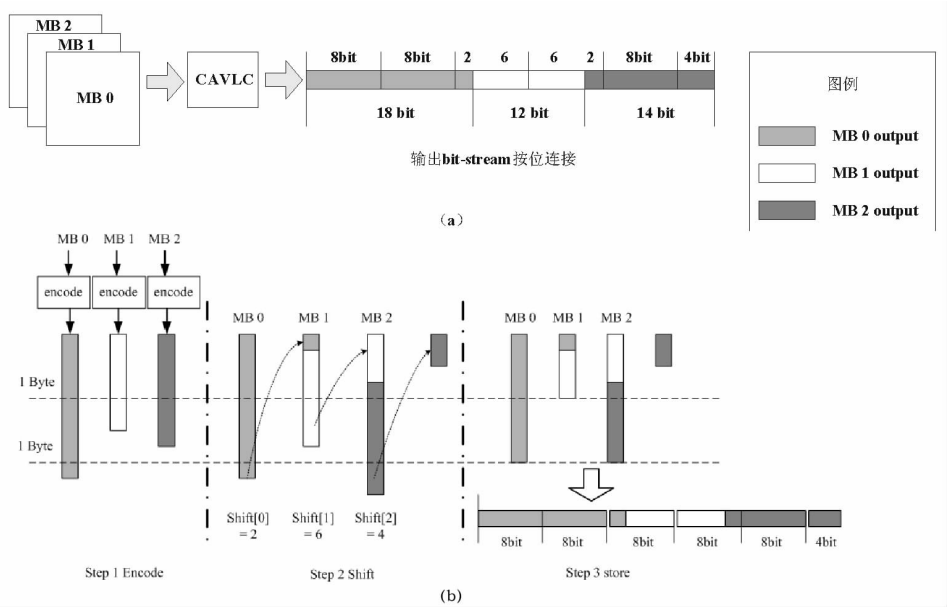


图 4　CAVLC 的并行码流生成法

Fig.4　Parallel bit-stream generation in CAVLC

针对 CAVLC 这种具有优先约束型存储问题的应用,我们提出一种并行码流生成法,通过预先计算存储长度、并行移位等技术将存储位置的不可预知变成可预知,从而解决优先约束型存储的相关问题。该方法使用 3 个步骤完成多宏块的并行码流存储,如图 4(b)所示。

Step 1　并行编码。各宏块被分配到不同的处理单元中并行编码,每个宏块内的 27 个块依次编码,为每个宏块生成一个拼接好的码流。这个码流可能是字节对齐的,也可能不是。在码流的生成过程中为每个宏块记录生成的码流长度 $L$,该长度以比特位为单位。

Step 2　并行移位。这一步的目的是将每个宏块的码流按字节对齐。宏块码流的结尾如果不是字节对齐的,则将多余的不足 8 位的码流填充至下一个宏块的码流首字节,如图中 MB0 末尾的 2 位将保存至 MB1 码流的开始。为了与新移入的码流拼接,下一个宏块则需要将原来的码流向后移位,如图中 MB1 需要向后移 2 位。然后下一个宏块根据新的码流长度计算移出码流的长度,继续形成整字节对齐的码流。上述码流的移出、填充和移位过程仍然是一个串行过程,因为前面的不移位完成,后面的无法移位,则实际上仍然没有解决优先约束存储问题。为了将这一过程并行起

来,必须首先计算每个宏块的移位长度。既然在 step 1 中已经为每个宏块计算出码流长度 $L$,则在 step 2 的一开始就可以精确计算出每个宏块最终需要移出的码流长度和整体的移位长度。因此各个宏块可以实现并行的移出、移位和填充。另外,每批并行宏块中的最后一个宏块,其移出码流作为变量传递给下一批将要被并行处理的宏块,填充进其中第一个宏块的开始字节即可。

Step 3 码流输出。现在每个宏块中的码流都是整字节对齐的,并且长度已知。这些码流不需要任何调整即可直接输出。

采用此方法的 CAVLC,图像中同一行的宏块都可以被并行处理。

### 3.3 去块滤波

在去块滤波过程中,当前宏块的滤波过程虽然与其相邻的左边宏块和上面宏块的重建数据相关,但这部分数据是由 IDCT 产生的,在去块滤波之前已经执行完毕。因此相关的数据可以事先计算和预取,这就使得宏块的并行处理成为可能。

去块滤波的并行流化的思路是,在不改变编码正确性的条件下更改滤波顺序,将它分为两个部分处理。将滤波过程分割成两个 kernel(Fdec _ Deblock _ V 和 Fdec _ Deblock _ H),如图 5 所示。首先以列优先的顺序处理宏块,在对 Fdec _ Deblock _ V 进行调用时,一列的所有宏块被处理,并且每个宏块只处理垂直边界。这样一列的所有宏块的滤波过程都可以同时进行。类似地,然后以行优先的顺序处理宏块,在对 Fdec _ Deblock _ H 进行调用时,一行的所有宏块被处理,并且每个宏块只处理水平边界,这时候一行的宏块同样都可以同时进行滤波计算。通过这样的操作,滤波过程更加有效地发挥出了 SIMD 处理的优势,挖掘出了更高的并行性,提高了处理性能。对于去块滤波过程中存在的较多的分支控制语句,采用冗余分支执行的方法达到 SIMD 并行的目的。

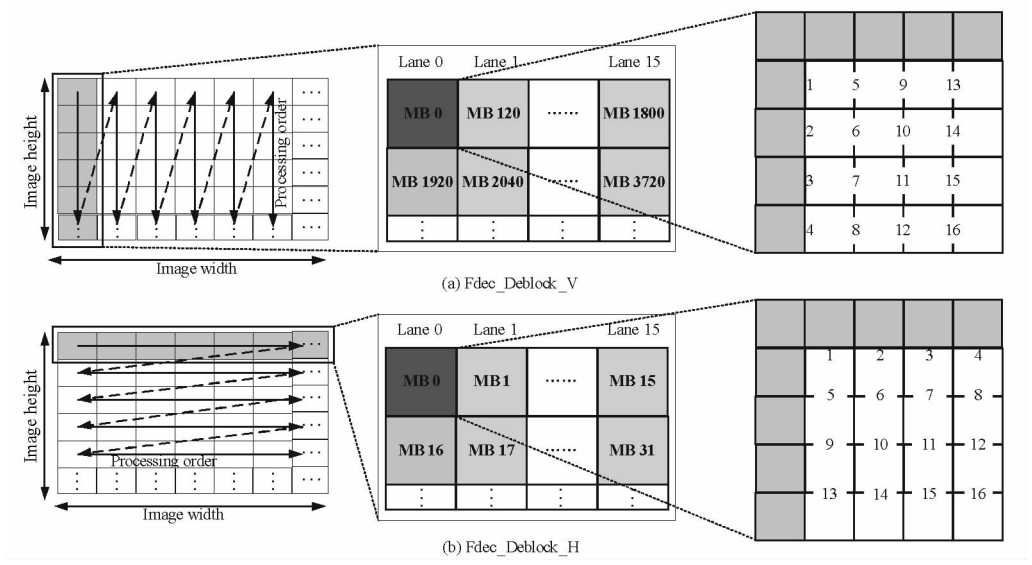


图 5 去块滤波的 kernel 划分和流数据组织

Fig.5 Kernel and stream organization in deblock filter

## 4 实验评测

本文在流数字信号处理器 STORM-SP16 G220[8] 上对采用原始串行方法和数据并行方法的 H.264 编码模块进行了性能测试和对比分析。SP16 G220 包括 16 个计算通道,以 SIMD 的方式处理数据。

本文选取 HD-VideoBench 高清视频测试序列作为实际性能评测的视频测试序列,它包含 4 个高清视频测试序列,参数如表 1 所示。采用前文提到的数据并行方法,各编码模块中的相关性问题得以解决,数据处理粒度明显上升,对于 1080p 的视频而言,帧间预测和 CAVLC 并行潜力数据集大小上升到 30KB,为流化 H.264 实时编码打下了重要的基础[9]。

**表 1 高清视频测试序列参数**

Tab.1 HD VideoBench configuration

| 测试序列 | 分辨率 | 帧数 | 格式 | 视频数据量 |
|---|---|---|---|---|
| Blue _ sky | 1080p | 217 | YUV 4:2:0 | 644MB |
| Pedestrian _ area | 1080p | 375 | YUV 4:2:0 | 1112MB |
| Riverbed | 1080p | 250 | YUV 4:2:0 | 742MB |
| Rush _ hour | 1080p | 500 | YUV 4:2:0 | 1483MB |

表 2 是各编码模块采用 x264 串行编码和数

据并行的流式编码的编码时间对比。

**表 2　各编码模块性能**

Tab.2　Performance of different coding module

| | 帧间预测 | 帧内预测 | 熵编码 | 去块滤波 |
|---|---|---|---|---|
| Blue _ sky stream | 1.61s | 7.81s | 1.03s | 0.62s |
| Blue _ sky x264 | 20.67s | 11.79s | 3.60s | 3.04s |
| Pedestrian _ area stream | 2.79s | 13.02s | 1.70s | 1.06s |
| Pedestrian _ area x264 | 36.08s | 20.42s | 6.00s | 5.27s |
| Riverbed stream | 1.89s | 8.89s | 1.47s | 0.84s |
| Riverbed x264 | 26.07s | 14.01s | 4.78s | 4.18s |
| Rush _ hour stream | 3.97s | 17.85s | 1.90s | 1.44s |
| Rush _ hour x264 | 50.27s | 26.93s | 6.90s | 7.48s |
| 平均加速比 | 13.04 | 1.54 | 3.47 | 4.98 |

表中数据显示,4 个视频测试序列在各个编码模块中的表现一致,数据并行方法的性能比串行方法有显著提升:帧间预测在采用数据并行方法后,平均加速比高达 13.04;而帧内预测、熵编码和去块滤波的加速比也分别达到了 1.54、3.47 和 4.98。可见,帧间预测的加速最为明显,这是因为帧间预测计算量大、访存次数多,是视频编码中最耗时的部分,但是其并行性和数据粒度也最大,因此可以很容易组织长流和高密度的 kernel。帧内预测的加速效果最差,这主要是因为其并行度仅为 2,没有充分发挥数据并行的优势。熵编码的并行度虽然高达 16,但是受到码流生成过程中大量访存操作的影响,其加速比较低。而去块滤波的计算量较小,使得 kernel 计算密度很难提高,另外其中的大量控制相关带来额外的冗余操作也降低了性能,因此加速比也较低。

## 5　结 论

本文归纳了 H.264 编码中存在的主要相关性问题:数据相关、码流存储的优先约束和控制相关,并针对这些相关性限制提出了帧间预测宏块并行、帧内预测块级并行、熵编码多宏块的并行码流生成、去块滤波宏块级并行等。实验结果表明,采用上述数据并行方法后,提高了流化 H.264 编码的数据并行粒度,为 H.264 高效流化并最终达到实时编码打下重要基础。

## 参 考 文 献:


[1] Wiegand T. Draft Text of Final Draft International Standard (FDIS) of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496 – 10 AVC) [C]//7th Meeting of JVT, Pattaya. Mar. 2003.

[2] Chen T C, Lian C J, Chen L G. Hardware Architecture Design of an H.264/AVC Video Codec [C]//Yokohama, Japan: In Proceedings of the 2006 Conference on Asia South Pacific Design Automation, 2006: 750 – 757.

[3] Rixne S. Stream Processor Architecture [M]. Boston: Kluwer Academic Publishers, 1985: 1 – 6.

[4] Mattson P. A Programming System for the Imagine Media Processor [D]. Stanford: Stanford University, 2002.

[5] Buck I, Foley T, et al. Brook for GPUs: Stream Computing on Graphics Hardware [J]. ACM Transactions on Graphics, 2004, 23: 777 – 786.

[6] 张春元,文梅,伍楠,等. 流体系结构技术发展探讨[J]. 国防科技大学学报, 2009,31(5): 1 – 5.

[7] 文梅.流体系结构关键技术研究[D].长沙:国防科技大学, 2008.

[8] Stream Processors Inc. 2008. SPI software Documentation [Z]. http://www.streamprocessors.com, 2008.

[9] Wu N, Wen M, Wu W, et al. Streaming HD H.264 Encoder on Programmable Processors [C]//Beijing, China: Proceedings of the 17th ACM International Conference on Multimedia, 2009: 371 – 380.