分 类 号 _____ 密级 _____

学校代码 10542 学号 200810020031

H.264 视频编码的并行实现

Parallel Implementation of H.264 Video Coding

研 究 生 姓 名 胡 佳

指导教师姓名、职称 满家巨 教授

学 科 专 业 计算机软件与理论

研 究 方 向 视频处理

湖南师范大学学位评定委员会办公室

二零一一年五月

摘 要

视频处理技术是伴随着视频从模拟到数字化转变的过程中得到蓬勃发展的。随着人们对视频图像的清晰度、流畅度、实时度的要求越来越苛刻，使其成为了一项炙手可热的技术。1980 年以来，国际电信联盟（ITU）和国际标准化组织（ISO）先后颁发了一系列关于静止和活动图像的压缩标准，奠定了该技术的雏形。H.264被称为新一代先进视频编码标准，H.264相对以前的编码方法，在图像内容预测方面提高了编码效率，它采用帧内预测、可变块大小运动补偿、1/4采样精度运动补偿等算法改善了图像质量，增加了纠错功能和各种网络环境传输的适应性。但与此同时，计算复杂度也随之增加了。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H.263的2倍，这就限制了它在实时视频编码中的应用。因此，改进H. 264 的相关算法，降低它的计算复杂度是非常有必要的。

CUDA(Compute Unified Device Architecture)是一种由NVIDIA推出的通用并行计算架构，它包含了 CUDA 指令集架构（ISA）以及GPU（Graphic Processing Unit）内部的并行计算引擎。该架构是用于GPU计算的开发环境，它运用了GPU中大量闲置的流处理器的资源，进行大规模并行计算科学计算。GPU（Graphics Processor Unit）起初主要应用于3D图形渲染，受游戏市场和军事视景仿真需求的牵引，GPU 性能提供速度很快。目前，越来越多的软件开发人员正在使用CUDA 软件开发工具来解决各种专业以及家用应用程序中的问题。这些应用程序从视频与音频处理和物理效果模拟到石油天然气勘探、产品设计、医学成像以及科学研究，涵盖了各个领域。于是将GPU用于视频编解码已成了一大研究热点。

帧内预测算法在较大程度上增加了计算复杂度。为了找到一个宏

块的最佳编码模式，需要对亮度块与色度块的大量模式组合全部搜索一遍，率失真代价的计算量相当庞大，使得H.264 的编码复杂度大大增加。因此本文提出了一种改进的快速帧内预测算法，并将该算法并行实现。利用GPU的强大浮点计算能力和并行特性，提出了CPU+GPU的并行编码架构，总结CUDA技术对并行效率的影响，并将SAD计算部分并行实现，更高程度上提高编码效率。

关键词：CUDA，H.264，帧内预测，SAD

ABSTRACT

Video processing technology is accompanied by a video transition from analog to digital by the process of vigorous development. People require video images clear, smooth and real-time, which makes it a hot technology. Since 1980, the International Telecommunication Union (ITU) and the International Standards Organization (ISO) has issued a series of still and moving image compression standard, which established the prototype of the technology. Known as the next generation of advanced H. 264 video coding standard, H. 264 improves coding efficiency in image content prediction compared to the previous method. It uses intra prediction, variable block size motion compensation, 1/4 sample accuracy Motion compensation algorithm for improving image quality and increasing the error correction and various adaptive transmission network environment. But at the same time, computational complexity also will be increased. The computational complexity of encoding is 3 times compared to H. 263; decoding complexity is 2 times compared to H. 263, which limits its real-time video coding applications. Therefore, it is necessary to improve algorithm and reduce its computational complexity.

CUDA (Compute Unified Device Architecture) is a general purpose parallel computing architecture launched by NVIDIA, which includes the CUDA instruction set architecture (ISA) and the parallel computing engine within GPU (Graphic Processing

Unit). The architecture is a development environment for GPU computing. It uses large number of idle stream processors resources in GPU for large-scale parallel scientific computing. GPU (Graphics Processor Unit) at first mainly used in 3D graphics rendering, by the games market and the demand for military visual simulation of traction, GPU provides fast performance. Currently, more and more software developers are using CUDA software development tools to solve a variety of professional and home applications problems. These applications from video and audio processing and physics simulations to the oil and gas exploration, product design, medical imaging and scientific research, covers various fields. So the GPU for video encoding and decoding has become a major research focus.

Intra prediction algorithm increased the computational complexity to a greater extent. It requires searching all mode combinations of luminance block and chrominance block to find an optimal coding macro block mode, the rate distortion cost of computation is enormous, which making H. 264 coding complexity greatly increased. Therefore, this paper proposed an improved fast intra prediction algorithm and parallel implementation of the algorithm. This paper used the power of GPU floating-point computing and parallel features; proposed CPU + GPU parallel encoding architecture; summarized the impact on the parallel efficiency of CUDA technology; achieved SAD parallel computing; made a higher degree to improve coding efficiency.

Keywords: CUDA, H. 264, intra prediction, SAD

目 录

1 绪 论

1.1 引言

90 年代以来，随着 Internet 和移动通信技术的发展成熟，数字视频技术在 Internet 网络和移动网络中的处理和传输成为了当前我国信息技术化的热点。视频图像也越来越广泛的被应用在许多领域，例如很多家庭都装了数字电视、大小型公司用可视电话，会议电视会谈、医学检测，军事侦查勘探图像处理等等。视频信号由于信息量大，传输网络带宽要求比较高，于是将视频信号在传送前进行压缩编码，然后在网络上进行传输，以便节省传送带宽和存储空间成为当下最核心的问题。

根据信源模型不同[1]，可将视频编码分为两大类，基于波形的编码和基于内容的编码。基于波形的编码是利用像素间的空间相关性和帧间的时间相关性，采用预测编码和变换编码组合起来的基于块的混合编码方法，减少视频信号的相关性，从而降低视频序列的码率，实现压缩编码的目标。基于内容的编码技术是对不同物体的形状、运动和纹理进行编码。例如人的脸部，已开发了一些预定义的线框对人的脸部特征进行编码，这时编码效率很高，只需少数比特就能描述其特征。除这两种编码模式外，立体（三维）视频编码是视频编码的发展方向之一，其平面信息外增加了深度信息，数据量非常庞大。

早在 20 世纪 40 年代，人们就开始着手视频编码技术的研究，至今已经取得了巨大的成就。视频编码技术基本是由 ISO/IEC 制定的 MPEG-x 和 ITU-T 制定的 H.26x 两大系列视频编码标准的推出。从 H.261 视频编码建议，到 H.262/3、MPEG-1/2/4 等都有一个共同的目标，即在尽可能低的码率（或存储容量）下获得尽可能高的图像质量。

H.264 视频编码标准与以往标准相比，不仅使视频压缩有明显提高，而且具有良好的网络亲和性，特别是对无线移动网、IP 互联网等易误码、易阻塞、QOS 不易保证的网络视频传输性能有明显的改善。然而 H. 264 性能的改进是以增加编解码的复杂性为代价的。我们对高清视频进行实时编解码时，CPU 的负担是很沉重的。因此现在我们利用 DSP 芯片、Intel 多媒体指令集、GA 硬件电路等方法，并且已经获得了较好的效果。而且国际两大显卡厂商 NVIDIA 和 ATI 生产的显卡芯片也都开始支持 H.264。

NVIDIA 公司在 1999 年发布 GeForce 256 图形处理芯片时首先提出 GPU 的概念。GPU 是显卡的“心脏”，也就相当于 CPU 在电脑中的作用。事实证明，在浮点运算、并行计算等部分计算方面，GPU 可以提高相比 CPU 数十倍乃至上百倍的性能；而且 GPU 价格便宜，如果把 GPU 编程用于 H.264 视频编码，采用 CPU+GPU 并行架构，必然能在更大程度上提高编码效率。

1.2 视频压缩编码标准发展史

近 10 年来，图像编码技术得到了迅速发展而且日臻成熟，其标志就是几个图像编码国际标准的制定。即国际电工委员会 IEC 和国际标准化组织 ISO 关于静止图像的编码标准 JPEG、国际电信联盟 ITU-T 关于电视电话/会议电视的视频编码标准 H.261,H.263 和 ISO/IEC 关于活动图像的编码标准 MPEG-1，MPEG-2 和 MPEG-4 等。这些标准融合了各种性能优良的图像编码方法，代表着目前图像编码的发展水平。下面简单介绍一下这些编码标准的特性：

(1)JPEG(Joint Photographic Expert Group)：JPEG 是 ISO/IEC 联合图像专家组制定的静止图像压缩标准，是针对静止图像的基于 DCT（离散余弦变换）的国际标准。JPEG-2000 是比以往的 JPEG 标准优势更明显的标准。JPEG-2000 是以小波变换为主的多分辨率编码方

式，压缩比 JPEG 高约 30%，JPEG-2000 能实现无损压缩。在实际应用中，有一些重要的图像，如卫星遥感图像、医学图像、文物照片等，通常需要进行无损压缩；误码鲁棒性好，相比更稳定，抗干扰性更强。而且能实现渐进运输，它可以先传输图像的轮廓，然后逐步传输数据。用户可先看到这个图片的轮廓，然后再决定是否下载。这在极大程度上方便了网络用户。

（2）MPEG-X 标准是由 ISO 所制定发布的专门为 CD 建立视频和音频的压缩标准。主要有以下五个：MPEG-1、MPEG-2、MPEG-4、MPEG-7 及 MPEG-21 等。H.264 是 MPEG-4 第十部分。MPEG-X 标准主要利用具有运动补偿的帧间压缩编码技术，减小图像的时间冗余度，利用 DCT 技术减小图像的空间冗余度，利用熵编码减小图像的统计冗余度。这几种技术的综合运用，令视听传播方面进入了数码化时代。

（3）H.261 标准是 1990 年正式发布的，该标准首次采用了运动补偿预测编码加 DCT（离散余弦变换)的框架，奠定了视频编码技术的基础。H.261 主要应用于会议电视，可视电话，窄带 ISDN 等领域。H.261 用于视频通信时，会产生多个国家的互通困难问题，我们知道不同国家采用不同的彩电制式，因此不能互通。H.261 采用一种公共中间格式（CIF）。不论何种彩色格式，发送方都先把自己国家的彩电制式转换成 CIF 这个格式，经 H.261 编码后再由 CIF 格式转换到接收方彩电制式。H.261 协议只允许使用 I 帧和 P 帧两种模式，运动估计采用整像素运动矢量。所以，H.261 的压缩码率不是很高。

（4）1995 年，ITU-T 总结当时国际上视频图像编码的新进展，针对低比特率视频应用在 H.261 基础上制定了 H.263 标准。H.263 标准特别适合在 PSTN 网络、无线网络等环境下的视频传输。H.263 已经被几种可视电话采纳为终端标准。

（5） H.264/AVC 是 ITU 与 MPEG 共同组成的 Joint Video Team

(JVT) 所制定的视频压缩标准。H.264 集中了以往标准的优点，并吸收了以往标准制定中积累的经验，仍然采用传统的混合编码框架，但是引入了多参考帧、多块类型、整数变换、帧内预测等新的压缩技术，使得编码效率显著提高。H.264 的颁布是视频压缩编码发展史上的一件大事，它优异的压缩性能也将在数字电视广播、视频实时通信、网络视频流媒体传递等各个方面发挥重要作用。

数字电视的优越性已经是公认的，但是它的广泛应用还有赖于高效的压缩技术。例如，利用 MPEG－2 压缩的一路高清晰度电视(HDTV)，大约需要 20Mb/s 的带宽，如果利用 H.264 进行一路 HDTV 的压缩，大概只需要 5Mb/s 的带宽。可见 H.264 视频编码的优越性。欧洲的数位电视广播（DVB）标准组织于 2004 年采用 H.264/MPEG-4 AVC 进行数位电视广播；日本所采用的 ISDB 数位电视广播制式，提供的 ISDB-T SB 移动地面电视广播服务，同样也使用了 H.264/MPEG-4 AVC 编码；台湾公共电视台(PTS)，采用的也是 H.264/MPEG-4 AVC 视讯编码格式；而香港方面，无线电视与亚洲电视的高清频道与新增的标清频道，也采用 H.264/MPEG-4 AVC 作为编码制式。

H. 264 在视频通信中具有比以往编码标准更高的抗阻塞，抗误码的能力，而且具有良好的网络亲和性，现在基于 DSP 的用 H.264 编码的可视电话已经上市，表示 H.264 在视频通信中的应用价值越来越大。

H.264 还有一个重要应用，即网络的流媒体。流媒体技术广泛用于视频点播、远程教育、远程医疗、网络电台、 实时视频会议等互联网信息服务的方方面面。H.264 没有对所有权有限制，是一个公共的开放的标准。因此，增强了各个生产商的竞争，使得产品价格迅速下降，让这项技术能为更多的人服务。目前我国宽带上网用户已经突破 1 亿。由此可见，一个高质量的高压缩性能的视频编码技术给我们

的生活和工作带来了多么深远的影响。

1.3 国内外研究现状

1.3.1 H.264 视频编码算法研究现状

我们知道，H.264 的高效性是建立在高复杂度的算法基础上的，解码器复杂度相比以前的标准提高了 3 倍，而且编码器的复杂度高达 10 倍左右。因此，提高编码效率成为目前视频编码的核心问题。帧内预测和帧间预测占整个编码运算的 70%左右。所以对 H.264 编码器优化是突破点。目前国内外研究者提出了很多的快速算法：

减小帧内预测复杂度的方法大致是利用当前块与周围像素或者的相关性，预先排除可能性很小的预测模式，或提前终止模式预测，从而降低帧内预测的复杂度。基于时空相关性的快速算法有：文献[2-5]利用相邻模块之间的相关性，跳过一些概率比较小的模式，降低模式选择的复杂度。基于图像纹理的快速算法有：文献[6-9]基于像素块的纹理特性，利用离散交叉微分（DCD），Prewitt 算子，解拉普拉斯方程等方法来确定图像的边缘信息。根据宏块纹理方向提前终止模式选择。

针对 H.264 运动估计过程中 7 种运动估计模式计算量过大的问题，从最早的全搜索，到三步法及其改进算法[10-11]，钻石型搜索算法[12]，四步法[13]，六边形搜索法及其改进算法[14-16]，非对称十字交叉多层次六边形格点搜索法[17]以及由这些算法演变的快速运动搜索算法。这些快速算法是目前为止提出来的效果比较好的整像素搜索算法。例如文献[18]分别对 UMHexagonS 算法中搜索窗口大小的选择、小六边形(小钻石)和大六边形搜索搜索模式做了优化，既保持原有图象质量，运动估计的时间平均节省了 18.292%。文献[19]提出了一种基于大小正方形的运动估计算法，从预测点的选择、搜索模板的确定到自适应阀值上进行了优化，编码效率得到明显提高。

总之，H.264 编解码器的加速研究工作以及取得了不菲的成绩，给人们的工作生活带来了很多便利，但还没有得到普遍实用。当然国内外的无数视频研究工作者还会一直继续致力于此，相信未来一定会有更惊人的成就。

1.3.2 GPU 在视频编码中的应用现状

伴随着图形处理器 GPU 的高速发展，它不仅价格低廉，而且还有着高并行计算能力和高数据带宽，完全可以代替或协助 CPU 进行计算。GPU 不再仅限于 3D 渲染，通用计算也开始走向实用阶段。例如文献[20]利用 GPU 完成分子动力学模拟，使其突破计算速度慢这个制约其研究发展的瓶颈，具有重要的理论意义和现实意义。医学图像融合是目前的一个研究热点，然而过去的图像融合技术，在精度以及计算时间上都不太理想，从而限制了它的应用范围。文献[21]提出了基于 GPU 的多尺度算法——频域非下采样轮廓波变换，对医学图像融合技术研究，具有重要的应用价值和学术意义. GPU 在视频处理[22]，信息安全[23]等方面也取得了骄人的成绩.

最近几年，人们开始将 GPU 应用到视频编解码中。文献[24]将微软公司 WMV. 8 解码器中的运动补偿、重构、色彩空间转换这 3 个模块移植到 GPU 中，取得了较好的加速效果。文献[25]提出了许多新的技术实现 DCT / IDCT。文献[26-27]将 H. 264 中运动补偿、重构、色彩空间转换移入 GPU，实现了高清视频的实时解码。目前，GPU 在各个领域都得到了广泛应用，今后还会展示出了巨大的潜力和应用价值，为各个领域的发展做出贡献。

1.4 论文研究的主要内容和意义

这些年来，国内外研究人员一直致力于如何提高视频编码效率。H.264 是新一代视频编码标准，它提出了比以往标准性能更优的编码技术；CUDA 是一种通用并行计算架构，该架构能够利用 GPU 强大的

浮点计算能力和巨大的存储器带宽进行并行计算，GPU 可以提高相比 CPU 数十倍乃至上百倍的性能；而且 GPU 价格便宜，如果把 GPU 编程用于 H.264 视频编码，采用 CPU+GPU 并行架构，必然能在更大程度上提高编码效率。本文正是基于以上背景和思路研究 H.264 中视频编码的并行实现。

本文首先简单介绍 H.264 和 CUDA 的基础知识和研究现状；总结国内外提出的快速帧内预测算法，提出一种基于纹理的快速帧内预测算法，并将该算法并行实现；总结国内外提出的各种快速运动估计算法，并将运动估计算法的 SAD 计算部分并行实现。通过实验比较，验证本论文提出的算法和并行思想的有效性。

1.5 论文章节安排

本文研究的是 H.264 视频编码的并行实现，首先简单介绍 H.264 和 CUDA 的基础知识和研究现状；总结国内外提出的快速帧内预测算法，提出一种基于纹理的快速帧内预测算法，并提出 CPU+GPU 架构，将该算法并行实现；总结国内外提出的各种快速运动估计算法，分析各种基于 GPU 的运动估计算法，并将 SAD 计算部分并行实现。

第一章 介绍视频压缩编码标准发展史，国内外研究现状，以及本论文主要研究内容和研究意义，并概括了论文的组织结构。

第二章 介绍 H.264/AVC 视频编码标准和 CUDA 基础知识。对其关键技术如帧内预测、帧间预测、整数变换和量化等进行了详细阐述；并简单介绍了 CUDA 的硬件映射和编程基础，为后面的论述作铺垫。

第三章 简单介绍 H.264 帧内预测算法的流程；总结国内外提出的快速帧内预测算法，在 Pan 算法的基础上提出一种快速帧内预测算法，并将该算法并行实现。将本文改进的基于 GPU 的帧内预测算法与 Pan 算法比较，得出相应的结论。

第四章 分析 CUDA 优化策略对并行效率的影响；并在前人提出的思想

上将运动估计算法中的SAD计算部分并行实现。并与串行的SAD计算所需的时间进行比较，得出相应的结论。

第五章 对本文进行总结和展望。

2 H.264 标准和 CUDA 基础知识简介

2.1 新一代视频压缩编码标准—H.264/AVC

目前，视频技术的应用范围十分广泛，如网上可视会议、网上可视电子商务、网上购物、网上学校；远程医疗、网上研讨会、网上展示厅、个人网上聊天等业务。但是传送包含视频信息的信号需要比较高的网络带宽，即为获得视频信息需要付出的代价。为了减少代价，我们必须对视频进行压缩编码。即把信息数据量以压缩形式存储、传输，既节约了不少存储空间，又提高了通信干线的传输效率，同时也可使计算机实时处理音频信息，以保证播放出高质量的音频节目。可见压缩编码的重要性。

2.1.1 H.264/AVC 编码器原理

由于视频内容时刻在变化，有时细节很多，有时大面积区域都很平坦。这种内容的多变性就必须采用自适应的技术措施；而且信道在不同环境下也是多变的，例如互联网有时畅通，有时阻塞，这就要求我们采取相应的自适应方法来对抗这种信道畸变带来的不良影响。这两方面的多变带来了自适应压缩技术的复杂性。由于大规模集成电路技术的迅猛进步，今天已完全具备了实现的可能性。

H.264 编码器的功能结构分为两层，即视频编码层（VCL）和网络提取层（NAL）。VCL 层主要包括帧内预测，帧间预测，变换量化，熵编码等单元。VCL 层主要利用帧内预测和变换技术去除数据的空间冗余；利用运动估计和补偿运动去除数据时间的冗余；然后对残差块进行量化和熵编码，进一步去除冗余。VCL 数据表示被压缩编码后的视频数据序列。在 VCL 数据传输或存储之前，这些编码后的 VCL 数

据，会先被映射或封装进 NAL 单元中。NAL 则负责以恰当的方式对数据进行打包和传送。

编码器采用的是变换和预测的混合编码法。如下图2-1所示，输入的帧或场Fn是以宏块(macroblock)为单位进行处理的。如果采用帧内预测编码，其预测值PRED（图中用P表示）是由当前片中前面已编码的参考帧（图中用F’ n-1）表示经运动补偿（MC）后得到。参考图像在过去或未来已编码解码重建的帧中选择，这样能提高预测精度。预测值PRED 与当前块相减产生一个残差块Dn，经块变换、量化产生一组变换系数X，再经熵编码，与解码需要的一些边信息（如预测模式的量化参数、运动矢量等）共同组成一个压缩后的码流，经NAL传输和存储。

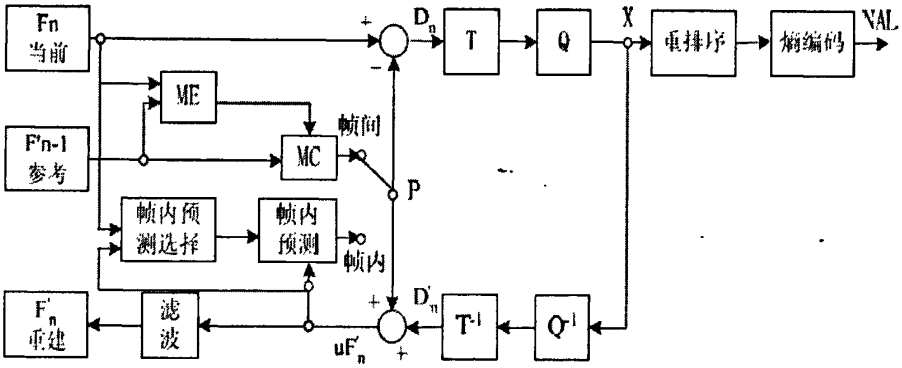


图 2-1 H.264 编码器

2.1.2 H.264/AVC 关键技术

H.264各种编码模式的选择模式可归结为：在所有的宏块类型中，遍历所有的可用编码模式，计算出在每种模式下其相应的重建图像失真度和编码比特数。

$$J(s,c,m|QP,\lambda_{\blacksquare}) = SSD(s,c,m|QP) + \lambda_{\blacksquare} R(s,c,m|QP) \quad (2\text{-}1)$$

式中，$\lambda_{\blacksquare}$为拉格朗日系数；QP为宏块的量化参数，与QP有关，s，c分别表示原始图像和重建图像的像素值；R（s，c，m|QP）表示利用模式m

编码输出的比特数；SSD（s,c,m|QP）表示原始亮度块s与重建亮度块c的差值平方和，其意义是图像的失真度。计算拉格朗日函数值J就是率失真代价，比较各个模式的率失真代价后，选择其值最小的编码模式为宏块的编码模式。计算拉格朗日因子：

$$\lambda_{模}=0.85\times 2^{(QP-12^{*})/3} \qquad (2\text{-}2)$$

式中，$\lambda_{模}$表示拉格朗日因子；QP是编码后的帧及宏块的量化因子。

2.1.2.1 帧内预测

帧内预测实质是利用到了邻近像素间的空间相关性。如图2-2所示，4×4亮度块上方和左侧的相邻像素A-M是经过重建得到的，利用相邻像素A-M的像素值通过某种预测模式来实现对当前编码块（a-p）的预测，通过计算比较各种模式下的绝对误差和（SAE），选取SAE 值最小的模式作为最佳预测模式，这种预测模式能使预测帧更加接近原始帧。预测块和当前块相减得到残差，然后对进行变换、量化和熵编码。残差值很小，这样就达到了图像压缩的目的。

Q	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	I	j	k	l				
L	m	n	o	p				

图2-2 当前4×4宏块像素及周围像素

H.264视频编码分为亮度编码和色度编码。根据图像细节的变化程度，帧内预测分为4×4亮度块预测，16×16亮度块预测和8×8色度块预测。4×4亮度块有9种预测模式，适用于有大量细节的图像编码，预测像素及预测方向如图2-3所示。16×16亮度块有4种预测模式，适合平坦区域的图像编码，预测像素及预测方向如图2-4所示。对于色度分量8×8块也有DC（平均）、水平、垂直、平面4种预测模式。

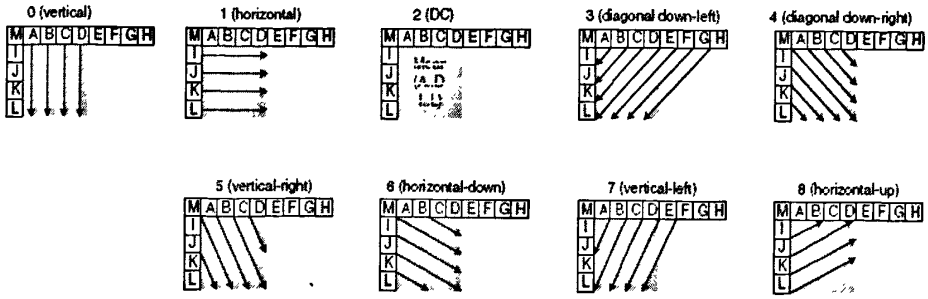


图2-3 Intra4x49种预测模式

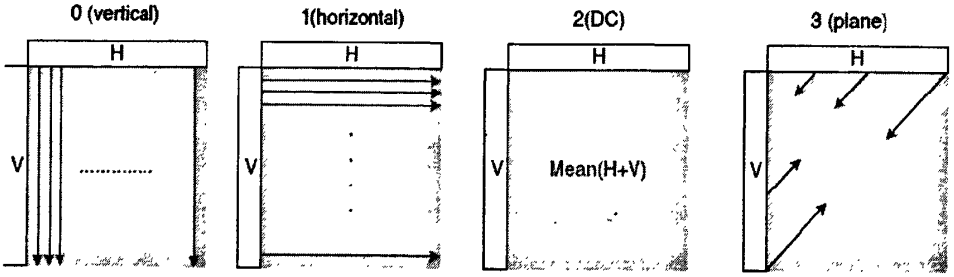


图2-4 Intra16x164种预测模式

我们知道帧内预测采用的是全搜索的方式，所以国内外很多学者都提出了基于各种理论减少帧内预测模式的帧内预测算法，具体会在后面章节重点讨论。

2.1.2.2 帧间预测

一般而言，帧间预测编码编码效率比帧内更高。在帧间预测编码中，运动估计是帧间预测编码核心，占整个系统运算量的60%-80%。运动估计实际上就是寻找最优或次优的运动向量的过程。它的基本思想是由于活动图像邻近帧之间存在着一定的相关性。我们可以将图像序列的每一帧分割成许多互不重叠的可变尺寸的宏块，比如16×16，16×18，8×4等等，我们称这些分割方式为树状结构分割。可变尺寸运动块的优点在于对视频序列中比较平坦的部分，采用较大尺寸块；对视频序列中细节内容比较多的部分，采用较小尺寸的块，这样可使预测更加准确。并认为宏块内所有象素的位移量都相同，然后根据一定的匹配准则（均方误差最小准则（MSE），绝对误差均值MAD，绝对误差和SAD）在邻近参考帧中搜索匹配块，得到运动矢量（匹配块与当前块的相对位移）。得到运动矢量的过程称之为运动估计。视频压

缩的时候，只需保存运动矢量和经过运动匹配之后得到的残差，然后将他们共同发送到解码端，从已经解码的邻近参考帧图像中找到相应的块或宏块，和残差相加后就得到块或宏块在当前帧中的位置。

高质量的运动估计算法是高效视频编码的前提和基础。其中块匹配算法是应用最广泛的算法。由于它算法简单且易于硬件实现。文献[28-29]都对块匹配算法进行了改进，各种算法的优缺点都会在后面进行分析和总结。

2.1.2.3 整数变换和量化

变换编码是一种间接的编码方法，其核心思想是在时域或空域进行描述时，数据间的相关性大，数据冗余度也大。变换编码首先将空域图像信号映射到变换域或频域，产生一批变换系数，然后对这些变换系数进行编码处理。在变换域中描述的数据相关性大大减少，数据冗余量减少，就样就能得到较大的压缩比。到达接收端后通过反变换回到样值，虽然会有一定的失真，但人眼是可以接受的。在视频压缩中，最常用的变换方法是 DCT 变换。

为了避免以往标准中通用 8×8 DCT 变换的逆变换出现失配问题，H.264 对图像或预测残差采用的是 4×4 整数 DCT 变换技术，这种新技术具有很多优点，例如：采用 4×4 整数 DCT 进行变换能够降低图像的块效应，整数运算代替了浮点运算提高了运算速度，避免了反变换误匹配的问题，也更有利于硬件实现。

量化过程在不降低视觉效果的前提下减少图像编码长度，减少视觉恢复中不必要的信息。H.264 采用标量量化技术，根据图像的动态范围大小确定量化参数，它将每个图像样点编码映射成较小的数值。既保留图像必要的细节，又减少了码流。文献[30]提出了三种变换矩阵(整数 DCT 正变换，整数 DCT 反变换，4 阶 Hadamard 变换)的快速实

现方法，并将量化和反量化过程集成到变换矩阵的硬件架构中；文献[31]把变换和量化分开来，设计了正向变换和反变换的并行架构。

在图像编码中，变换编码和量化从原理上讲是两个独立的过程。但 H.264 里这两个过程中的乘法合二为一。整数变换及量化具体过程如图 2-5 所示。如果输入的块是色度块或帧内 16×16 预测模式的亮度块，则将宏块中各 4×4 块的整数余弦变换的直流分量组合起来，进行 Hadamard 变换，进一步压缩码率。

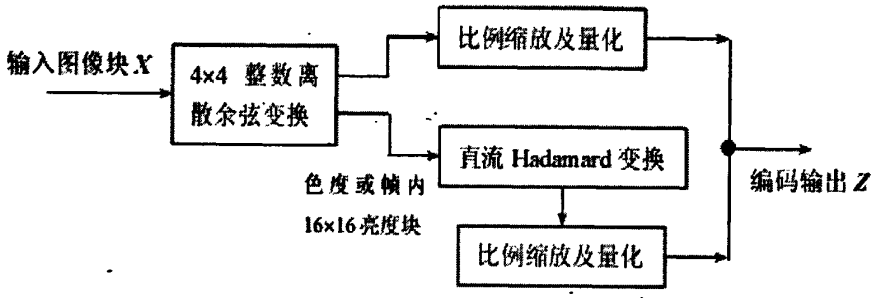


图 2-5 变换编码过程

2.1.2.4 去方块滤波

对 DCT 变换系数进行量化的过程相对比较粗糙，因而反量化过程恢复的变换系数会有误差，造成在图像块边界不连续。而且在进行运动补偿预测时，运动补偿块的匹配不可能是绝对准确的，所以也有可能会在复制块的边界上产生数据的不连续。尽管 H.264/AVC 采用 4×4 变换尺寸可以减小这种不连续造成的视觉误差，但仍需要一个去方块滤波器以最大程度提高编码性能。在视频编解码器中一般使用后置滤波器和环路滤波器区分真实的和人为的图像边界，并有效地滤除人为图像边界。在相同的 PSNR 下可以节省超过 9%的码流，从视觉上明显提高图像质量。文献[32-33]通过减少，求解微处理器 Intel Centrino 上边缘强度过程中，的分支判断次数加快滤波过程。

2.1.2.5 熵编码

熵编码即编码过程中按照熵原理不丢失任何信息的编码。在视频编码中，熵编码是最后一个步骤。基本原理是把一系列用来表示视频

序列的元素符号转变为用于传输或者存储的压缩码流.输入的符号包括量化的变换系数,运动向量,头(宏块头,图象头,序列头等)以及一些附加信息等.熵编码包括2种编码算法：CAVLC(基于上下文自适应的可变长编码)和CABAC（基于上下文自适应的二进制算术编码）。与CAVLC相比，CABAC节省了7%左右的码流，但增加了10%的运算时间。而且CABAC的抗误码率能力差，各有利弊。文献[34-35]对熵编码进行了优化，有效降低率失真优化的复杂度,节约编码时间,易于硬件实现。

2.2 CUDA编程简介

CUDA（Compute Unified Device Architecture）是由NVIDIA推出的一种集成技术。用户可利用这个技术实现GPU编程。以GeForce 8800 GTX为例，其核心拥有128个内置处理器。利用CUDA技术将这些处理器合理安排，将密集的数据并行起来计算。而各个处理器之间能够交换、同步和共享数据。CUDA从发布开始已经到了4.0版，从NVIDIA提出CUDA这个概念开始，已经经历了将近四年的时间，最近一年里CUDA发展迅速，特别是Fermi架构推出之后GPU可编程性得到急剧提升，目前CUDA除了能够利用GPU强大的浮点计算能力和巨大的存储器带宽进行图形渲染之外,还能应用于图像处理、视频传播、流体力学、生物计算、分子动力学计算、金融分析、石油勘探、天文计算等领域，并在这些领域中对CPU的加速效果比较显著。

2.2.1 CUDA硬件映射

GPU包括很多个流处理器（SM），即计算核心。每个SM又包括8个标量流处理器（SP），以及一些其他的计算单元。商业广告中提到的多核实际上是指SP的数量。CUDA中的Kernel函数是以block为单位执行的，一个SM中有多个block在等待执行，即存在多个block的上下文。同一个block中的线程必须在同一个SM中发射，且一个

block 中的每个 thread 被发射到一个 SP 上，如图 2-6 所示。这样他们就能共享数据了。

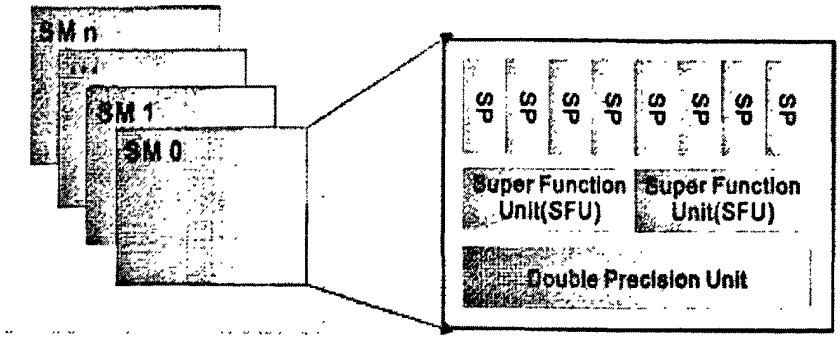


图 2-6 CUDA 硬件映射

目前，一个内核函数中只有一个 Grid，未来硬件也许会支持多 Grid.这样设备利用率会更高。一个 Grid 里面有多个 block,block 在运行时又被分割成线程束（warp）。线程束的多少由硬件的计算能力决定。如采用 Tesla 架构的 GPU 中，一个线程束由 32 个线程组成。Warp 中的线程只与线程号 threaID 有关，而与 block 的维度和尺度无关。即每发射一条 warp 指令，SM 中的 8 个 SP 会执行 4 遍这条指令，这就是一个 warp 为什么会有 32 个线程的原因。

执行内核的最小单位是线程。每个线程有自己的 blockID 和 threadID 来区分其他线程。一个线程块的所有线程都在一个流多处理器（SM）中执行。因此，同一个线程块中的线程可以通过共享内存来共享数据；也可以指定同步点，当线程块中的线程全部到达此同步点时挂起，以防止访问冲突。

2.2.2 CUDA 编程基础

CUDA 在整个 GPU 计算中充当的就是翻译的角色。它负责把 CPU 的计算指令翻译成 GPU 的计算指令,同时还负责显存和内存之间的数据交换。而 CUDA 程序负责把要处理的数据合理地分配然后并行计算。CUDA 程序本身还是要靠 CPU 来执行的。这种编程模式只要求开发人员对 GPU 进行一次编程，无论是针对处理器多还是少的 GPU 产品。当运行 GPU 计算程序时，开发者只需在 CPU 上运行程序，CUDA 驱动自

动会将程序载入到 GPU 上并执行。GPU 运算功能的启动、数据的传输以及其它一些 CPU 和 GPU 之间的交互都可以调用驱动中的专门操作来完成。这些高级操作把程序员摆脱了手动管理 GPU 运算资源的烦恼。

CUDA 架构提供的是用 C 语言编写设备端代码的编码方式。CUDA 编程模型将 CPU 作为主机（Host）端，GPU 作为协助处理器或者设备（Device）端。CPU 和 GPU 各司其职，协同工作。在进行 CUDA 编程时,Device 端程序为 GPU 上执行的部分，该程序又称 Kernel（内核函数）。Kernel 以线程网格（Grid）的形式组织，每个线程网格由若干个线程块(block)组成，而每个线程块又由若干个线程(thread)组成。一个 Kernel 函数中两层并行模型是 CUDA 最重要的创新之一，即 Grid 中的 block 间并行和 block 中的 thread 间并行。

一旦确定了程序中的并行部分，Host 端就会将数据准备好,拷贝一份到显卡的内存中，由 GPU 执行 Kernel 程序，完成后再由 Host 端程序将结果从显卡的内存中拷回。一个 Kernel 函数并不是一个完整的程序，而是整个 CUDA 程序中的一个可以被并行执行的部分。

一个典型的CUDA 程序一般包括以下过程:

(1) 在CPU 上初始化数据;

(2) 将数据拷贝到GPU;

(3) 运行kernel函数处理数据;

(4) 把结果传回CPU。

2.3 本章总结

H. 264 作为新一代视频编码标准，性能确实比以往标准更好。而 GPU 作为并行计算的硬件新宠，能很好的运用到 H。264 视频编解码中。本章简单介绍了关于视频信号的特点，视频压缩编码标准发展史，H.264/AVC 编码器原理以及它的关键技术；CUDA 的硬件映射以

及编程基础，并给出了一个完整的CUDA程序实例，为后面的CUDA在视频编码中的应用作铺垫。

3 基于 GPU 的快速帧内预测算法

基于空间域的帧内预测是 H.264/AVC 采用的一项新技术，它充分利用图像的空间相关性，根据已解码重建的相邻块的信息来预测当前块的信息。H.264/AVC 在率失真优化 RDO 模式下进行帧内预测模式选择，定义率失真代价最小所对应的预测模式为最佳的预测模式。我们知道 H.264/AVC 采用的是全搜索模式。对于亮度分量，帧内 4×4 块有 9 种预测模式；帧内 16×16 有 4 种预测模式。而对于色度分量，每个 8×8 色度块也有 4 种预测模式。因此，确定一个宏块 MB 的帧内预测模式，需要进行 592 次率失真代价(RD_Cost)计算，导致编码复杂度相当的高。如何减少运算时间是帧内预测算法优化的最终目标。

3.1 经典的快速帧内模式选择算法

目前大致可以从以下两个方面减小帧内预测复杂度:一方面是简化代价函数;另一方面可以缩小预测模式选择的范围，或预先排除某些可能性小的预测模式，或提前终止某些可能性小的模式的代价计算，从而降低因全搜索各种帧内预测模式造成的高复杂度。

首先从简化代价函数上考虑，该类方法主要对 RDcost 算法本身进行优化。常用的方法是寻找可以替代 RDO,同时对编码质量影响不大的简化函数,通常我们会将 RDO 公式中的差值均方和(SSD)公式,用绝对变换差值和(SATD)或绝对差值和(SAD)替代。文献[36]基于以上考虑，简化了代价函数，减小了计算复杂度；同时 Intel 公司推出了多项适合于 RDcost 计算的 SMID(单指令多数据流)技术，该技术可由一条单一指令完成多条操作，这种并行技术使运算速度得到显著提高。

另一方面，保留标准中的代价函数公式,从缩小预测模式选择的范围或预先排除某些可能性小的预测模式，或提前终止某些可能性小

的模式基础上考虑，已成为国内外研究的热点。例如：文献[37-38]通过设置阈值法或者概率预测等方法,目的都是减少候选模式的数量；Meng Bojun 等提出了 EIPMS 算法，利用代价函数和多阈值的方法提高了 4×4 子块的编码速度[39]；文献[40]利用下采样方法，用 RD 模型替代率失真方法选择预测模式。文献[41]根据相邻 4×4 块的预测模式之间的强相关性，即当前 4×4 块的左边块和上边块的预测模式可以预测当前块的最可能模式（MPM），从而减少预测模式的选择数目。此外文献[42]利用宏块的 MAD(平均绝对误差)信息及时空相关性，结合自适应多阈值的方法，对于特征明显的宏块，采用自适应阈值的方法从两种预测模式中选取一种；同时在 Intra 4×4 的 9 种预测模式中利用阈值判断进行快速选择。对于那些特征不是很明显的宏块，采用标准算法进行判断，以确保选取的是最佳预测模式模式。文献[43]选取方向性比较强的几个模式使用简化的代价函数 SATD 计算相应的 RD，取值较小的两个和 DC 模式加入到预测范围中，然后根据这两个模式与周围预测模式的相关性判断是否有新的模式加入到最优模式中来；最优模式选取完了之后，接着进行 RDO 代价计算，取 RDCost 最小值对应的预测模式为最佳预测模式。最优的帧内预测模式与视频图像块的纹理方向是紧密相关的，由于边界方向能够很好的反映出图像的纹理方向,据此方向减少候选模式的数量,降低计算复杂度,节省运算时间.利用这个特点,Pan 等人在 JVT 会议中提出了一种基于边缘方向直方图的快速算法[37]。在对 4x4 块进行帧内预测时，先使用 Sobel 算子对整幅图像做卷积运算,求出每个像素亮度和色度梯度的幅度以及方向角,统计得出图像块的累计方向直方图,据此判定图像纹理最强的方向,选取这个方向，和它最接近的两个方向，DC4 个方向预测模式作为最优模式。 对于 16×16 或者 8×8 大小的块,也用类似的方法,选取这个方向模式和 DC 模式作为最优模式。该算法在极大程度上

减少预测模式，减少了大约 50%的运算时间,同时保证了在同样的码率水平下恢复出来的图像具有相近的 SNR（峰值信噪比）。

另外，各种基于 GPU 的并行帧内预测算法也相继被提了出来。文献[44]经统计后发现，在对当前 4×4 块进行帧内预测时，有些预测模式如模式 0，3，7 并不存在数据依赖，即不需要等待临近块编码重建完就能进行帧内预测。临近块在编码重建的同时，能进行模式 0，3 和 7 的预测。等临近块编码重建完后，再将其他模式 1，2，8 和 4，5，6 并行预测。总的来说，将帧内 4×4 的 9 种预测模式分成三组，每组三种预测模式。这种并行架构和 H.264 帧内预测算法相比，虽然都是全模式搜索，但是每个 4×4 块的处理时间由 $16\times T_p+16\times T_r$ 降到 $17\times(\frac{1}{3}T_p+T_r)$。$T_p$ 是当前块的预测时间；T_r 是临近块的重建时间。文献[45]同样也分析了帧内模式选择中的数据依赖性，提出了一种基于贪婪编码顺序的并行算法。文献[46]在微电子设备 FPGA 和 ASIC 上并行实现帧内预测算法，在不降低图像质量的前提下，这种并行处理方法能有效的降低运算复杂度。

3.2 基于 GPU 的快速帧内预测算法

3.2.1 快速帧内预测算法

本论文在 Pan 等提出的基于纹理的快速算法的基础上，提出一种基于 GPU 的快速算法。Pan 用 Sobel 算子提取边缘，仅仅考虑了水平和垂直两个方向的边缘信息。但是 Kirsch 算子有 M0-M8 个方向模板来确定梯度幅度值和梯度的方向，如下图 3-2 所示。每个方向模板对应的 8 个方向如下图 3-3 所示。因此能得到更细致的边缘纹理信息。在 Intra4×4 预测时，先计算 8 个方向梯度幅度值，其中最大值作为该像素的边缘强度。最大梯度幅度值的序号对应的即该像素点的边缘方向。最大幅值对应的模式，与此相邻的两个方向模式和 DC 模式作为候选模式。

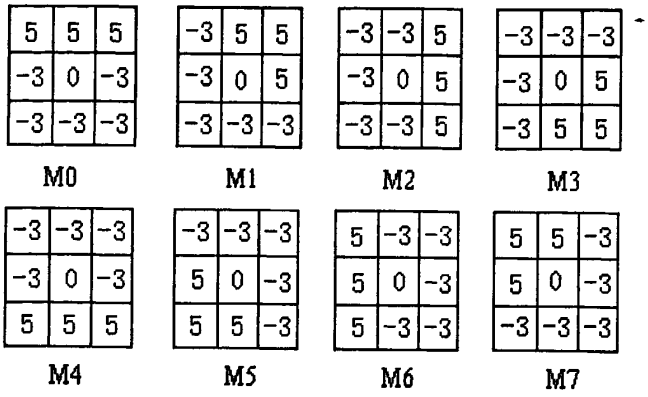


图 3-1 Kirsch 算子 8 个方向模板

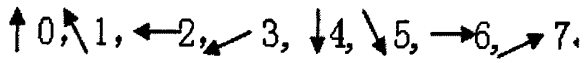


图 3-2 8 个方向模板对应的 8 个方向

3.2.2 快速帧内预测算法流程

(l) 首先，对4×4块用Kirsch算子求边缘向量的幅值:

```
K0=5*imgY_org[i+1][j-1]+5*imgY_org[i+1][j]+5*imgY_org[i+1][j+1]-
3*imgY_org[i][j-1]-3*imgY_org[i][j+1]-3*imgY_org[i-1][j-1]-3*imgY_
org[i-1][j]-3*imgY_org[i-1][j+1];
K1=5*imgY_org[i+1][j]+5*imgY_org[i+1][j+1]-3*imgY_org[i-1][j-1]+5
*imgY_org[i-1][j]-3*imgY_org[i+1][j-1]-3*imgY_org[i-1][j+1]-3*imgY
_org[i][j-1]-3*imgY_org[i][j+1];
K2=5*imgY_org[i+1][j+1]-3*imgY_org[i][j-1]+5*imgY_org[i][j+1]-3*i
mgY_org[i+1][j-1]+-3*imgY_org[i+1][j]-3*imgY_org[i-1][j-1]-3*imgY_
org[i-1][j]+5*imgY_org[i-1][j+1];
K3=5*imgY_org[i][j+1]-3*imgY_org[i+1][j-1]-3*imgY_org[i+1][j]-3*i
mgY_org[i+1][j+1]-3*imgY_org[i][j-1]-3*imgY_org[i-1][j-1]+5*imgY_
org[i-1][j]+5*imgY_org[i-1][j+1];
K4=5*imgY_org[i-1][j-1]+5*imgY_org[i-1][j]+5*imgY_org[i-1][j+1]-3*
imgY_org[i+1][j-1]-3*imgY_org[i+1][j]-3*imgY_org[i+1][j+1]
-3*imgY_org[i][j-1]-3*imgY_org[i][j+1];
```

K5=5*imgY_org[i-1][j-1]+5*imgY_org[i-1][j]-3*imgY_org[i+1][j-1]-3*imgY_org[i+1][j]-3*imgY_org[i+1][j+1]+5*imgY_org[i][j-1]-3*imgY_org[i][j+1]-3*imgY_org[i-1][j+1];

K6=5*imgY_org[i+1][j-1]-3*imgY_org[i+1][j]-3*imgY_org[i+1][j+1]+5*imgY_org[i][j-1]-3*imgY_org[i][j+1]+5*imgY_org[i-1][j-1]-3*imgY_org[i-1][j]-3*imgY_org[i-1][j+1];

K7=5*imgY_org[i+1][j-1]+5*imgY_org[i+1][j]-3*imgY_org[i+1][j+1]+5*imgY_org[i][j-1]-3*imgY_org[i][j+1]-3*imgY_org[i-1][j-1]-3*imgY_org[i-1][j]-3*imgY_org[i-1][j+1];

其中imgY_org[i][j]为像素点的灰度值。

(2) Kirsch 算子的梯度幅度值由如下公式计算：

img->Edge_Map_Amp[i][j]=max(|M0|,|M1|,|M2|,|M3|,|M4|,|M5|,|M6|,|M7|) (3-1)

(1) 最大幅值对应的模式，于此相邻的两个方向模式和DC模式作为候选模式。例如：img->Edge_Map_Amp[i][j]=K4，则K4对应的图3-1中的垂直方向，img->Edge_Map_Mode[BLK4x4][i][j] = 0;

（4）重复步骤1～3，预测完16个的4×4亮度子块。

16×16亮度块和8×8色度块预测算法流程：

（1）按照4×4亮度块的步骤1～3，求出每个像素的幅值和边缘方向。

（2）最大幅值对应的模式和DC模式作为候选模式。

如何给二维，三维数组分配显存空间，以及如何在设备端遍历二维，三维数组的元素，是该并行算法的重点和难点。内存和显存的分配机制和访问机制都不一样，对于二维和三维数组，我们使用cudaMallocPitch()和 cudaMalloc3D()分配线性存储空间，以保证分配满足对齐要求。

以下部分代码说明了本并行算法如何分配 2D 数组，以及如何遍

历数组元素。调用 cudaMallocPitch()时，该函数会返回经过填充，对齐后的 pitch 值。

```
//定义 GPU 二维数组
 int * gpu_Amp;
 byte * imgYorg;
 size_t pitch_amp;   //GPU数组的pitch
 size_t pitch_img;
 //分配显存空间
CUDA_SAFE_CALL( cudaMallocPitch((void**) &gpu_Amp,
&pitch_amp, sizeof(int) * BmpW, BmpH));
CUDA_SAFE_CALL( cudaMallocPitch((void**) &imgYorg, &pitch,
sizeof(int) * BmpW, BmpH));
//将二维数组从主机端拷贝到设备端
CUDA_SAFE_CALL(cudaMemcpy2D( gpu_Amp, pitch_amp,
img->Edge_Map_Amp, sizeof(int) * BmpW, sizeof(int) * BmpW,
BmpH, cudaMemcpyHostToDevice));
 CUDA_SAFE_CALL(cudaMemcpy2D( imgYorg, pitch, imgY_org,
sizeof(float) * BmpW, sizeof(float) * BmpW, BmpH,
cudaMemcpyHostToDevice));
//设备端遍历二维数组元素
__global__ void callone(cudaPitchedPtr gpu_Mode,cudaExtent
extent_Mode,int * gpu_Amp,int pitch_amp,byte *imgYorg,int
pitch,int height,int width,int uv)
{
    int i=blockIdx.x*blockDim.x+threadIdx.x+1;
    int j=blockIdx.y*blockDim.y+threadIdx.y+1;
```

```
    byte * row=(byte*)((char*)imgYorg+i*pitch);
    byte * row1=(byte*)((char*)imgYorg+(i+1)*pitch);
    byte * row2=(byte*)((char*)imgYorg+(i-1)*pitch);
    if(i < height-1 && j < width-1)
      {
     K0=5*row1[j-1]+5*row1[j]+5*row1[j+1]-3*row[j-1]-3*row[
j+1]-3*row2[j-1]-3*row2[j]-3*row2[j+1];
         //代码省略
       }
}
```

以下部分代码说明了本并行算法如何分配3D数组，以及如何遍历数组元素。与cudaMallocPitch()类似，cudaMalloc3D()将返回一个存储了每个维度填充后的偏移量大小的cudaExtent型结构体，并需要在_global_函数中使用它对数组中的元素进行寻址。

```
   //定义GPU三维数组
   cudaPitchedPtr gpu_Mode;
   cudaPitchedPtr gpu_C_Amp;
   cudaPitchedPtr gpu_C_Mode;
     //分配显存空间
   cudaExtent extent_Mode=make_cudaExtent(2,BmpW,BmpH);
   cudaExtent
extent_C_Amp=make_cudaExtent(2,BmpW_C,BmpH_C);
   cudaExtent
extent_C_Mode=make_cudaExtent(2,BmpW_C,BmpH_C);
   cudaMalloc3D(&gpu_Mode,extent_Mode);
   cudaMalloc3D(&gpu_C_Amp,extent_C_Amp);
```

```
    cudaMalloc3D(&gpu_C_Mode, extent_C_Mode);
    cudaMemset3D(gpu_Mode, 0, extent_Mode);
    cudaMemset3D(gpu_C_Amp, 0, extent_C_Amp);
    cudaMemset3D(gpu_C_Mode, 0, extent_C_Mode);
//将三维数组 gpu_Mode 从主机端拷贝到设备端,另外两个三维数组
执行类似
     cudaMemcpy3DParms ModeToDevice = {0};
     ModeToDevice.dstPtr = gpu_Mode;
     ModeToDevice.srcPtr=
make_cudaPitchedPtr((void*)img->Edge_Map_Mode,  sizeof(int),
2, BmpW);
     ModeToDevice.extent = extent_Mode;
     ModeToDevice.kind   = cudaMemcpyHostToDevice;
   status = cudaMemcpy3D(&ModeToDevice);
   //设备端遍历三维数组元素
__global__ void callone(cudaPitchedPtr gpu_Mode,cudaExtent
extent_Mode,int * gpu_Amp,int pitch_amp,byte *imgYorg,int
pitch,int height,int width,int uv)
{
     char*  modePtr= (char*)gpu_Mode.ptr;
     size_t pitch_mode=gpu_Mode.pitch;
     size_t slicePitch=pitch_mode*extent_Mode.height;
     char* slice=modePtr+uv*slicePitch;
     int* mode_row=(int*)(slice+i*pitch_mode);
     if (maxK==K0)
     {
```

```
        mode_row[j]=8; // Mode 0: Vertical
        //代码省略
        }
}
```

首先定义两个 Kernel 函数，callone 函数主要是在求 Intra4×4 和 Intra16×16 亮度块预测模式时，对子块中的每个像素求卷积，然后得到每个像素的方向；calltwo 函数主要是在求 8×8 色度块预测模式时，对子块中的每个像素求卷积，然后得到每个像素的方向。假如帧大小为 BmpH×BmpW，这两个核函数类似，这里只介绍 callone 函数算法的并行过程。算法流程如下：

(1) 首先做好初始化的工作，定义 CPU 和 GPU 数组相关变量，并分配显存空间。

(2) 将一帧图像的信息从 CPU 拷贝到设备端中。以上过程前面已经详细介绍了。

(3) 分配线程块（block）数和线程(thread)数，安排好 block 数和 thread 数后，将载入当前帧的数据载入到每个线程中。然后执行内核函数，采用 Kirsch 算子求每个像素的方向幅值和边缘方向。

```
//调用 kernel 函数
dim3 dimBlock4(16,16);
dim3 dimGrid4((BmpH+dimBlock4.x-1)/dimBlock4.x,(BmpW+
dimBlock4.y-1)/dimBlock4.y);
callone<<<dimGrid4,dimBlock4,0>>>(gpu_Mode,gpu_Amp,pitch_a
mp,imgYorg,pitch,BmpH,BmpW,BLK4x4,int uv);
```

(4) 将核函数计算得到的结果存入全局存储器，用于输出。

(5) 将结果从显存拷贝到主机，释放空间。

3.2.3 实验及结果分析

3.2.3.1 实验开发环境

（1） CUDA环境：CUDA toolkit3.2和CUDA SDK 3.2 for Windows XP；

（2） CPU：Pentium Dual-Core 2.93GHz；内存：4GB；显卡：NVIDIA GeForce GTX 280；系统：Windows XP, Visual studio 2005.

（3） 不同分辨率的标准测试系列：news分辨率176×144；foreman分辨率 352×288；mobile分辨率 720×576；riverbed分辨率1920×1080；

（4） JM6.1参数设置为：熵编码采用CAVLC；全I帧序列；测试序列数为50帧；运动估计搜索半径16×16。

3.2.3.2 性能比较

为了方便测试，将本文算法与 Pan 算法的运算时间提取出来进行对比，如表 3-1 所示。本文还比较了两种算法的峰值信噪比 PSNR。如下表 3-2 所示：

表 3-1 算法的编码时间比较（/s）

测试序列	分辨率	Pan	New	加速比
news	176×144	0.358	0.141	2.5
foreman	352×288	1.576	0.391	4.0
mobile	720×576	6.559	0.886	6.4
riverbed	192×1080	33.672	3.118	10.8

表 3-2 算法的亮度分量的平均 PSNR 比较(dB)

测试序列	分辨率	Pan	New	$\Delta PSNR$
news	176×144	37.7	37.71	0.01
foreman	352×288	37.59	37.62	0.03
mobile	720×576	35.79	35.80	0.01
riverbed	1920×1080	38.38	38.40	0.02

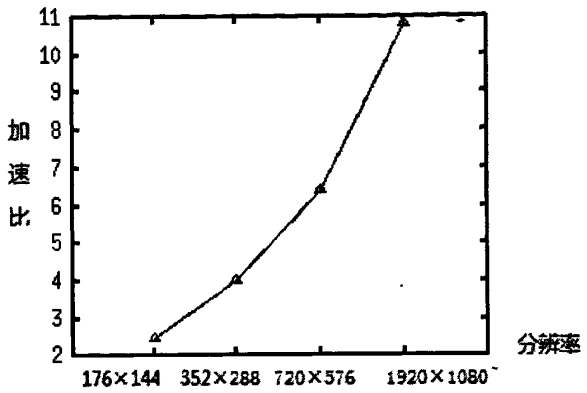


图3-3不同分辨率下加速比走势图

3.2.3.3 实验结果分析

表3-1对比了两种算法编码50帧的总耗时情况，以及改进算法相比Pan算法产生的加速比。实验数据表明，本论文提出的改进算法在保持编码质量的同时,总编码时间相对于Pan算法能达到10倍以上。

表3-2给出这3种算法的亮度分量的平均PSNR比较。其中的实验数据显示,本论文提出的快速算法与Pan算法和JM6.1相比,编码后的图像质量有小幅度提高。这说明本论文使用的Kirsch算子既能保持图像的细节又具有一定的抗噪能力。它比Pan算法的Sobel算子求得的边缘更精准。

图3-3给出的是不同分辨率下的加速比走势图，发现在CUDA环境下GPU上并行本论文改进的算法能提高运算效率；本文算法可获得10倍以上的加速比；而且根据走势图还得出一个结论:分辨率越大，GPU的并行处理能力越强，加速比越大。

3.3 本章总结

本章首先简单介绍了一下H.264帧内预测的算法流程，然后总结了国内外的改进算法现状；接下来着重介绍了几种经典的快速帧内预测算法的设计思路和算法流程，这些帧内算法都在一定程度上提高了编码效率；接下来本论文提出一种基于GPU的快速帧内预测算法，实验证明，把这种快速帧内算法移植到GPU上去运行，能在更大程度上减少运行时间，提高编码效率。

4 H.264运动估计算法的并行实现

将GPU应用到视频编解码中，和CPU并行工作。这一应用国内外已经有很多学者在研究。并且取得了很好的效果。我们知道，运动估计是帧间预测中最耗时的。各种快速运动估计算法都在不同程度上减少了编码复杂度，但如何将运动估计过程移植到GPU上去，是目前的研究热点。

4.1 经典的快速运动估计算法

目前，国内外已经提出了各种快速运动估计的算法。文献[47]提出了16×16大小全零块的检测，首先计算16×16宏块的SAD(绝对差值之和)，然后和预先设定好的一个阈值进行比较，如果判断出该宏块是全零块，那么大量4×4子块经过DCT（离散余弦变换）之后所有的系数都会被量化为零，这样DCT以及之后的Q（量化）、IQ（反量化）以及IDCT（反离散余弦变换）变换都能够省略掉[48,49]。而且这种全零块的数量多，可以有效的节省运算时间。文献[50]在菱形搜索的基础上，考虑到视频序列运动具有惯性的特点，即各图像块的最佳匹配块的梯度方向性很明显，基于此提出了一种基于运动方向的菱形-T形搜索算法。在进行匹配块搜索时，设置两种大小和方向不同的T形搜索模板，在已搜索最佳像素点的基础上确定预测像素点的位置。文献[51,52]经过多年对图像质量评价及人眼视觉系统的研究指出，人眼在观看图像时提取的是图像的结构信息，而且人眼能自适应实现这个目标，文献[53]在此思想的基础上提出基于结构相似的帧间预测改进算法。该算法引入一种新的图像质量评价方法SSIM（结构相似度），将其作为H.264帧间预测的失真测度，这种质量评价法比H.264标准中的MSE更加符合人眼主观感觉，而且计算也比较简单.

近些年随着GPU可编程性能的逐步提高，国内外视频研究者试图将运动估计算法移植到GPU上去并行计算。例如文献[54]提出了基于宏块级的并行运动估计算法。将16×16的宏块分成16个4×4块，然后计算出每个4×4块运动估计的代价并通过多次渲染将它们相加。接着文献[55]在此基础上，提出了基于帧级的并行运动估计算法。将整个帧放入GPU中进行运动估计。前几种算法都没实现对7种模式进行并行算法，而文献[56]则提出了H.264的7种模式全搜索并行方法。这些并行运动估计算法的提出，标志着GPU在视频编解码中应用越来越深入，具有重要的理论和实际意义。

4.2 CUDA 在视频编码中的应用

将各种快速运动估计算法在 GPU 上运行固然能提高编码效率，但是也可以从如何更好地使用 CUDA 技术的角度出发，提高并行效率。文献[57]着重研究了 CUDA 技术对并行运算的影响，而非研究快速运动估计算法对编码效率的影响。

4.2.1 线程块和线程的划分对运动估计算法的影响

该文献[57]首先分析了运动估计部分可以并行的两个部分：计算各种分割模式的绝对差值和（SAD）；找到最小的 SAD 值和其对应的运动向量。然后分析了在进行 SAD 并行计算时，那么如何分配 Grid 中的线程块（block）数和每个 block 中的线程（thread）数。文献提出了四种分法，并总结了各种分法对编码性能的影响。

（1） 一个线程计算两像素间的 SAD，线程块的维度必须和宏块大小相等。这种方法有两个限制条件：第一个问题是线程块的大小不能超过 22×22，因为硬件条件控制着最多只允许 512 个线程；其次必须另外增加一个核函数累加所有计算出来的独立 SAD 值；最大的一个问题是当我们计算一个宏块的 SAD 值时，每个宏块需要（2 *搜索范围）*（2 *搜索范围）个线程块，因此当搜索范围增大或者宏块数

较多时（图像尺寸变大），该考虑的问题就会变多了。

（2） 让每个线程计算一个宏块的 SAD，线程块的大小是（2 *搜索范围）*（2 *搜索范围）。这种方法的好处是一个线程块计算一个宏块的 SAD，这样分法不会存在以上问题，但是这样的话搜索范围就只有 11，这么小的搜索范围对一般目的的运动估计没有太大效果。

（3） 和之前方法不同的是一个线程块计算 1/4 搜索范围的 SAD 值，这样就需要为每个宏块分配 4 个线程块，这种方法最大的搜索范围是 22，相比上面的方法要好一些但还不够好。

（4） 结合 1，2，3 实验的结论提出一种混合的方法，即为了消除搜索范围大小对搜索结果的影响，动态地去分配线程块。这种方法既能实时的分配线程块数和线程数，又能安排一个比较理想的搜索范围。

4.2.2 不同的 CUDA 策略对编码性能的影响

文献主要将高效率的改进措施集中在如下几种 CUDA 策略上：

（1）全局存储器(global memory)带宽低，应尽量减少 global memory 的访问；

（2）尽可能的用共享存储器（shared memory）代替 global memory 的使用；

（3）消除 shared memory 中的 bank conflict，避免线程串行执行；

（4）线程块的尺寸为 16 的整数倍。

实验 I:通过 5 组对比实验，让他们同时执行运动估计中 SAD 的计算的代码，得出相应的实验结论。下面 5 组对比实验对应 5 种不同的 CUDA 策略；

（1）在 CPU 上运行标准的运动估计中 SAD 计算部分的 C 代码。

（2）用（3）提到的策略，即只使用 global memory 在 GPU 上运行标准的运动估计中 SAD 计算部分的 C 代码。

（3）在实验（2）的基础上，增加一个临时变量来存取产生的 SAD

值，减少数据在 global memory 上的频繁交互。

（4）在实验（3）的基础上，使用 shared memory 存取一个线程块产生的 SAD 值。

（5）在实验（4）的基础上，将以上 4 种优化策略都考虑在内进行 SAD 的计算。

该实验分别使用三种不同分辨率的图像，运行以上 5 组实验，如下表 4-1，4-2，4-3 所示；对比这三张表格，我们可以很明显的看出运行标准源码的运算速度是最慢的，当加入了临时变量之后运算时间才有了点变化。当使用 shared memory 之后，我们能看到显著的速度变化，这就表明了使用 shared memory 的重要性即使在很简单的没有经过优化的情况下。最后这个经过 4 种策略全优化后的实验(5)的加速效果是最明显的，当然图像分辨率越高，所做的运算工作也就越多，所花的时间也就越多。

表 4-1 325×288 分辨率下 5 组实验对比结果

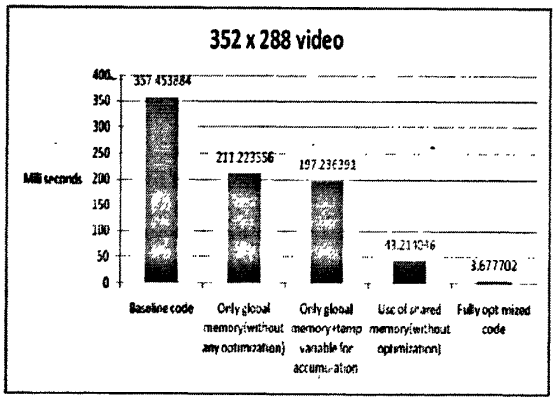


表 4-2 720×576 分辨率下 5 组实验对比结果

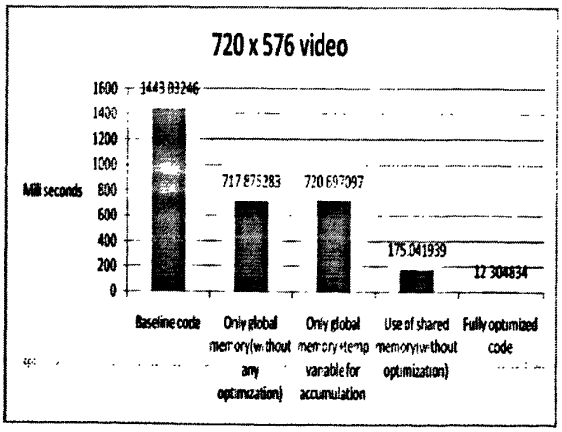
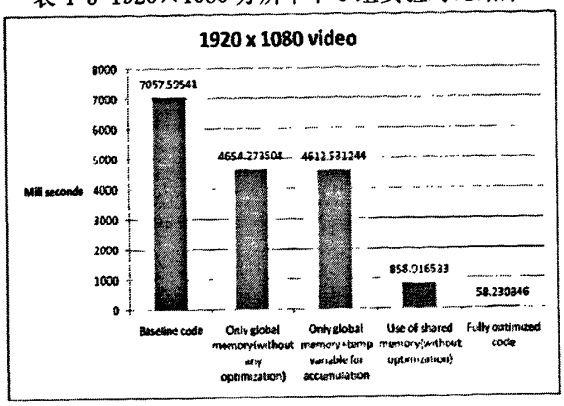


表 4-3 1920×1080 分辨率下 5 组实验对比结果



实验Ⅱ：通过 5 组对比实验，让他们分别在 800GT, 9800GTX, 285GTX 在三种不同处理能力的 GPU 上执行运动估计中 SAD 的计算的代码，而且将搜索范围设置在 5-46 之间，得出相应的实验结论。

（1）在 CPU 上运行标准的运动估计中 SAD 计算部分的 C 代码。

（2）用亚像素内插法求预测像素值，然后在 CPU 上计算相应的 SAD。

（3）用实验 I 中的(5)CUDA 策略，在 CPU 上运行标准的运动估计中 SAD 计算部分的 C 代码。

（4）在（3）的基础上，分别用整像素法和亚像素内插法求预测像素值，在 GPU 上计算相应的 SAD。

（5）用实验 I 中的（3）CUDA 策略，在 GPU 上运行标准的运动估计中 SAD 计算部分的 C 代码。

该实验同样使用三种不同分辨率的图像，运行以上 5 组实验，得出结论：随着搜索范围的增加，运行时间在 CPU 上呈指数增长，在 GPU 上呈半线性增长，这也就意味着搜索范围越大，加速比越大。搜索范围越大，CUDA 优化策略越好，加速比越大；另外，显卡性能越好，即处理器越多，加速比越大。

本论文根据该文献[55]经过实验得出的结论，将运动估计算法中 SAD 计算部分在 GPU 环境下并行实现。首先设置搜索半径为 16×16；采用的线程块的大小为 16 的整数倍；尽量定义浮点类型的变量，避免 shared memory 发生 bank conflict。因为每个 bank 的宽度固定

是32bit,如果线程访问的数据不是32bit，例如对double数组进行访问时就会发生2-way bank conflict,即线程串行执行;使用shared memory存取一个线程块产生的SAD值，而不是存储在global memory中，然后将所有产生的SAD值存入全局存储器中，这样就减少了数据在global memory上的频繁交互。具体的并行流程在后面章节中会详细介绍。

4.3 CPU+GPU 异构编码并行架构

传统的H.264编码器中存在着反馈环，而且必须等GPU在执行完ME后，将求得的最佳MV值交互给CPU，然后CPU才能进行整数变换(ICD)、量化(Q)、反量化(IQ)等操作，这时CPU必须等待GPU处理完每一个宏块才能进行后面的编码工作。如果对原先的编码器架构不加修改而直接移植的话，就会出现CPU和GPU相互等待的局面。为此，文献[58]借鉴文献[59]的思路，提出了一种基于帧级的编码器并行架构。其基本思想是：将重建滤波后的图像用原始图像代替作为参考帧，切断反馈环。这样CPU和GPU就同时处于编码器的不同阶段，如GPU在执行ME运动估计时，CPU则可以执行DCT。它们的各自工作对象都是整帧图像。只有整帧图像编码结束后， CPU和GPU之间才发生交互。使得CPU和GPU可以并行工作而无需相互等待。这种帧级的并行框架可以使GPU的工作更加有效。

文献[58]提出的基于GPU的编码器并行框架如图4-1所示。在该并行框架中，采用了两条并行流水线，CPU和 GPU。GPU上的流水线只负责运动估计(ME)，计算速度只与I／O操作有关，而不受CPU流水线的影响。CPU上的流水线的计算速度不仅与I／O操作有关，还需要等待GPU产生7种可变尺寸块的最佳MV值。为了减少CPU的等待时间，并充分发挥GPU的并行计算能力，文献设置的共享缓冲区为5-8帧MV大小，用来存放GPU运算得到的7种模式的最佳MV值。

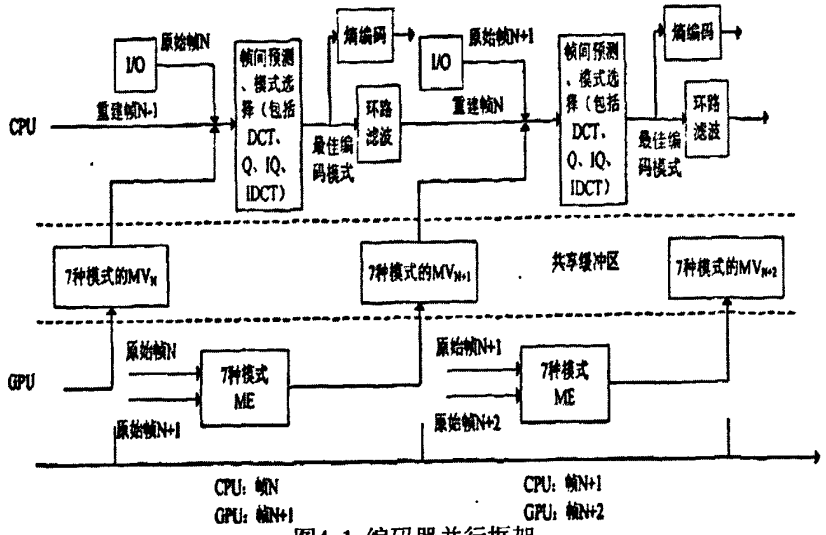


图4-1 编码器并行框架

该文献提出的并行框架虽然采用了原始帧代替重建帧进行运动估计，但不会影响编码器的性能。这是因为在CPU上进行的运动补偿（ME）过程仍然使用的是重建帧作为参考帧。而GPU产生的MV只是反映了该块的运动大小和方向，即CPU流水线仍然保留了从重建帧到运动补偿的这条回路，从而确保了不会产生误差累积。该文献提出的并行框架，减少了CPU和GPU之间交互的次数，能使GPU工作效率更高。

4.4 基于 CUDA 的运动估计算法并行实现

运动估计是视频编码中最耗时也是复杂度最高的部分，H.264运动估计过程采用的是全搜索（FS）算法，我们知道帧间预测有7种模式划分，因为基本处理宏块单元是16×16，它又可以划分成16×8，8×16和8×8；8×8又可以分成4×8，8×4和4×4三种子宏块分割方式。模式选择部分里面包含太多判断条件，不利于并行实现，本文在文献[60,61]并行思想上将SAD过程移植到GPU上去并行实现。由上我们可以看出，首先需要求解计算搜索范围内所有7种分割模块对应的SAD值，分别记为$SAD_{16\times16}$，$SAD_{16\times8}$，$SAD_{8\times16}$，$SAD_{8\times8}$，$SAD_{8\times4}$，$SAD_{4\times8}$，$SAD_{4\times4}$。然后找出其中最小的SAD，$SAD_{min}=min\{SAD_{16\times16}, SAD_{16\times8}, SAD_{8\times16}, SAD_{8\times8}, SAD_{8\times4}, SAD_{4\times8}, SAD_{4\times4}\}$。最小的SAD对应的分割方式即最佳帧间预测模式。其对应偏移量即为所求的运动矢

量。

4.4.1 运动估计并行算法

本论文设置的运动向量的最大偏移量为16，因此对应每一个当前图像块，全搜索算法需要计算$(2\times16+1)^2$次评价函数，即当前块就会产生最大搜索位置max_pos为 $(2\times16+1)^2$个候选块。首先定义每个block的维度为dim3 dimBlock(256,1)，则需要的线程块数为dimGrid =(max_pos+ dimBlock.x-1)/dimBlock.x。安排好block数和thread数后，载入当前帧和参考帧的数据到每个线程中。 将每个线程块计算得到的SAD值保存在shared memory（共享存储器）中。当所有线程块中的线程执行完后，所有子块的SAD值也都计算完毕，并将结果用一个全局数组保存起来存入全局存储器中。然后将该数组从设备端拷贝到主机端，比较得出最小的SAD值与其对应的MV（运动矢量）。

4.4.2 运动估计并行算法流程

运行内核函数mvsad_kernel之前，先要在CPU上进行预处理，以及参数的传递，CPU和GPU的数据交互。部分代码如下：

```
//定义CPU数组，并为之分配内存
int * cpu_mcost=(int*)malloc(sizeof(int)*max_pos);
int * gpu_mcost;
 //定义GPU数组, 并为之分配显存空间
 CUDA_SAFE_CALL(cudaMalloc((void**)&gpu _mcost,sizeof(int)*
max_pos));
//定义block数和thread数
dim3 dimBlock(256,1);
dimGrid =(max_pos+dimBlock.x-1)/dimBlock.x;
//执行内核函数mvsad_kernel
__global__ void mvsad_kernel(unsigned char ** orig_pic,int *
```

```
spiral_search_x,int * spiral_search_y,double lambda,int* mvbits,int
check_for_00, int pic_pix_x,int pic_pix_y,int * min_mcost,int
blocksize_x,int blocksize_y,int blocksize_x4,int flag,int width,int
height,int search_range,int pred_x,int pred_y,int center_x,int
center_y,unsigned char * ref_pic,int* byte_abs)
{
    int pos=blockIdx.x*blockDim.x+threadIdx.x;
    int max_pos= (2*search_range+1)*(2*search_range+1);
mcost=(((lambda_factor)*(mvbits[((cand_x)<<(2))-pred_x]+mvbits[((can
d_y)<<(2))-pred_y]))>>16);    //初始化运动代价
//将计算得到的当前块的残差SAD值加到最小代价mcost中去
for (y=0; y<blocksize_y; y++)
//预判断，得到原始数据orig_line和参考数据ref_line，代码省略
for (x4=0; x4<blocksize_x4; x4++)
{
        mcost += byte_abs[ *orig_line++ - *ref_line++ ];
        mcost += byte_abs[ *orig_line++ - *ref_line++ ];
        mcost += byte_abs[ *orig_line++ - *ref_line++ ];
        mcost += byte_abs[ *orig_line++ - *ref_line++ ];
  }
//将该搜索点位置得到的代价存入全局数组
min_mcost[pos] = mcost;
//将该数组从设备端拷贝到主机端
CUDA_SAFE_CALL(cudaMemcpy(cpu_mcost,gpu_mcost,sizeof(int)*m
ax_pos,cudaMemcpyDeviceToHost));
//求该数组的最小值即最小代价，best_pos对应的MV即最佳运动矢量
```

}

4.4.3 实验及结果分析

4.4.3.1 实验开发环境

（5）CUDA环境：CUDA toolkit 3.2和CUDA SDK 3.2 for Windows XP；

（6）CPU：Pentium Dual-Core 2.93GHz；内存：4GB；显卡：NVIDIA GeForce GTX 280；系统：Windows XP sp3,Visual studio 2005.

（7）不同分辨率的标准测试系列：news分辨率176×144；foreman分辨率352×288；mobile分辨率720×576；riverbed分辨率1920×1080 .

4.4.3.2 性能比较

下面是本文采用GPU加速的算法与CPU环境下运行的全搜索算法的性能比较。为了方便测试，将运动估计中的全搜索算法提取出来测试50帧序列，对不同分辨率大小的测试序列，运行搜索算法的时间不同。如下表4-9所示：

表4-4 运动估计算法在CPU和GPU上的性能比较

测试序列	分辨率	CPU/(s)	GPU/(s)	时间变化率
news	176×144	8.594	6.000	30.2%
foreman	352×288	49.617	34.609	30.3%
mobile	720×576	231.796	100.032	56.8%
riverbed	1920×1080	1975.290	511.680	74.1%

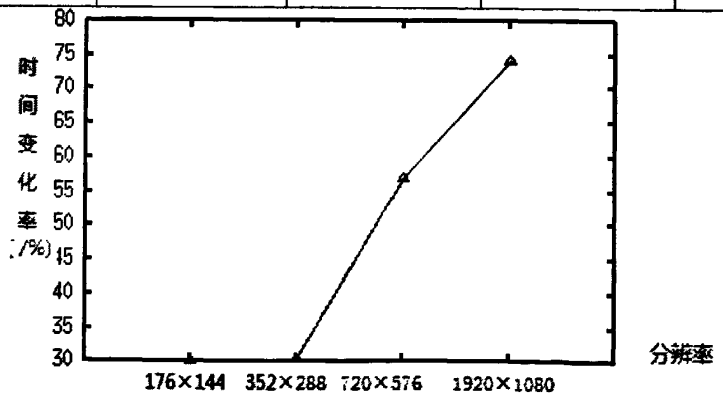


图4-2不同分辨率下时间变化率走势图

4.4.3.3 实验结果分析

由表4-4可以看出：在GPU环境下分别运行分辨率为176×144大小和352×288大小的序列运行时间差别不大，和CPU环境下运行的时间差不多，优势不太明显；但是当在GPU环境下运行分辨率为704×576和1920×1080的序列时，相比在CPU环境下的运行时间少了很多。可见，随着图像分辨率变大，GPU并行计算体现出来的优势逐渐明显。

通过图4-2走势图，发现在CUDA环境下GPU上运行全搜索算法能提高搜索效率，而本文并行算法随着分辨率的增大，编码时间能提高至74%以上；而且本实验还能得出一个结论:分辨率越大，数据量越大，越利于CUDA加速，GPU的并行处理能力越强，加速比越大，这也验证了文献[55]得出的实验结论。

4.5 本章总结

本章首先简单总结了国内外提出的快速运动估计算法的优缺点，然后着重分析了文献中提出的CUDA技术对并行运算的影响，通过几组对比实验得出使用shared memory的好处，搜索范围越大，CUDA优化策略越好，加速比越大，另外，显卡性能越好，即处理器越多，加速比越大。接下来在文献[45]的思想上提出基于帧级的视频编码的并行框架，该文献切断了原编码框架中反馈环，用原始帧代替重建帧，这样就产生了CPU和GPU两条流水线。最后本论文实现了运动估计中SAD的并行计算，分别对几组不同分辨率大小的视频序列进行运动搜索，得到最小SAD，让他们分别在CPU和GPU环境下运行，实验证明CUDA环境下并行运动估计算法能大大提高搜索效率；分辨率越大，CUDA计算能力越强，加速比越大。

5 总结和展望

H.264 标准作为面向未来 IP 和无线环境下的新一代视频编码标准，同 H.263 等标准的效率相比，能够平均节省大于 50%的码率。而且在较低带宽上提供高质量的图像传输是 H.264 的应用亮点。不仅如此，H.264 既可以在实时通信应用（如视频会议）低延时模式下工作，也可以在没有延时的视频存储或视频流服务器中应用。另外，H.264 相比以前的标准增加了多模式运动估计、帧内预测、多参考帧预测、基于内容的变长编码、4×4 二维整数变换等新技术。但这些新技术的应用都是建立在高编码复杂度的前提下的，因此提高编码效率成了当务之急。目前，国内外已经提出来大量改进算法，而且随着 GPU 的不断发展，它的并行处理能力越来越强，价格也越来越低廉，应用也越来越广泛。本文把如何利用 CUDA 技术提高 H.264 视频编码效率作为本文的研究目标。在本学位论文中，主要有下面几个创新点：

1. 本文首先研究和总结了大量国内外提出的快速帧内预测算法，提出一种基于纹理的快速帧内预测算法，由于边缘方向能很好的反映图像的纹理方向，因此本论文根据求出的边缘方向判断出宏块的候选预测模式，而不是全搜索的方法去进行帧内预测，这样就减少了帧内预测的复杂度，提高了编码效率。基于以上理论本文首先利用Kirsch算子（8个方向模板）对整幅图像做卷积，求出每个像素的梯度幅度值和梯度的方向。在进行帧内4×4块预测时，先计算8个方向梯度幅度值，其中最大值作为该像素的边缘强度。最大梯度幅度值的序号对应的便是该像素点的边缘方向。最大幅值对应的预测方向模式，与此相邻的两个方向模式和DC模式作为候选模式。在进行16×16亮度块和8×8色度块预测时，按照类似的方法求出最大幅值对应的方向模式，

该方向模式与DC模式作为候选模式进行帧内预测。由于在对整幅图像求卷积需要大量的计算量，因此本论文将该算法移植到GPU上去并行实现。通过实验比较Pan算法和改进的基于GPU的帧内预测算法的编码效率。实验证明，在图像质量稍有提高的情况下，改进的基于GPU的帧内预测算法的加速比高达10倍以上。

2.接下来本论文简单介绍了 CUDA 的硬件映射以及编程基础，并详细分析了 CUDA 优化策略对并行效率的影响；运动估计在整个视频编码运算中占最大比例，如何减少运动估计所耗费的时间是本文研究的另一个重要。通过查阅大量国内外文献和论文，本论文总结了前人提出的各种快速运动估计算法的优缺点。在前人提出的思想上将运动估计算法中的 SAD 计算部分并行实现。通过实验并与 H.264 标准源码中的全搜索算法所需的时间进行比较。实验证明，在 GPU 下运行全搜索算法的时间能提高 74%左右。

当然，本文还有一些不足和需要改进的地方。例如还可以把编码的其他流程如DCT,解码的IDCT，运动补偿等也交给GPU去做，做成一个完整的并行编解码器。这样不仅可以进一步分担CPU的工作，也减少了显卡和内存之间的数据交换.而且具有理论意义和实际意义。目前利用GPU做通用计算的研究还处于初期阶段，还只有少量的相关的成果供参考和运用。而且使用GPU做并行计算也存在一些限制，比如循环和分支的限制。相信随着GPU的发展，能把CPU从编解码的负担中解放出来，促进视频处理技术的发展，同时随着GPU的并行处理能力的不断提高，它的应用范围也越来越广，必能促进其他行业的发展，服务社会，给人们的生活带来各种意想不到的体验。

参考文献

[1]毕厚杰.新一代视频压缩编码标准-H.264 / AVC[M].北京：人民邮电出版社,2005:12-13.

[2]Pei Shi-bao,Li houqiang,YU Neng-hai.A fast intra-frame prediction mode selection algorithm of H.264/AVC[J].Computer Engineering and Application,2005(8):1-2.

[3]杨松,任永峰.H.264 中帧内预测模式选择算法的研究[J].淮阴工学院学报,2010.

[4]倪红霞,范志浩.基于邻块预测的 H.264 快速帧内预测模式选择算法[J].江南大学学报，2010,08.

[5]孟庆磊，姚春莲等.一种面向 H.264/AVC 的快速帧内预测选择算法[J].北京航空航天大学学报，2007，33（2）：219-223.

[6]朱金秀，李国选. 基于纹理复杂度的快速帧内预测算法[M].计算机工程与设计,2010,31(7)

[7] 肖广,滕国伟等. 一种基于H. 264 /AVC的帧内预测模式快速选择算法[J].中国图象图形学报.2005,11.

[8]TSAI A，PAUL A，WANG J，et al.Intensity gradient technique for efficient intra-prediction in H.264/AVC[J].IEEE Trans.Circuits and Systems for Video Technology，2008，18（5）：694-698.

[9]Liu Dong，SUN Xiaoyan，WU Feng，etal.Edge-oriented uniform intra prediction[J].IEEE Trans.Image Processing，2008，17（10）：

[10]KOGA T,IINUMA K，HIRANO A，etal. Motion compensated inter frame coding for video conferencing[C].In Proc Nat Tele Conf New Orleans，LA，1981，G.5.3.1-G.5.3.5.

[11]LI R，ZENG B，LIOU M L. A new three step search algorithm for block motion estimation[J].IEEE Transactions on Circuits and Systems for Video Technology，

1994，4(4):438—442.

[12]ZHU S，MA K K. A new diamond search algorithm for fast block matching motion estimation[C]. International Conference on Information，Communications and Signal Processing，1997，1：292—296.

[13]PO L M，Ma W C. A novel four step search algorithm for fast block motion estimation[J]. IEEE Transactions on Circuits and systems for video technology，1996，6(3)：313—317.

[14]ZHU C，LIN X，CHAN L P，et al. A novel hexagon—based search algorithm for fast block motion estimation[C]. IEEE International Conference on Acoustics，Speech，and Signal Processing，2001，3：1593—1596.

[15]ZHU C，LIN X，CHAU L P. Hexagon—based search pattern for fast block motion estimation[J].IEEE Transactions on Circuits and Systems for Video Technology，2002，12(5)：349—355.

[16]ZHOU G F，LIU G Z，SU R. A Modified Hexagon—Based Search algorithm for block motion estimation[C]. Proceedings of the 2003 International Conference on Neural Networks and Signal Processing，2003，2：1205—1208.

[17]CHEN ZH B，ZHOU P，HE Y. Fast Integer Pel and Fractional Pel Motion Estimation for JVT.JVT-F017. doe[R]，6th Meeting：Awaji，Island，JP，5—13,2002,12.

[18]吴晓军,白世军,卢文涛.基于 H.264 视频编码的运动估计算法优化[J].电子学报,2009,11.

[19]甘志鹏.一种适用于 H.264 运动估计及块模式选择算法研究[D].武汉：武汉大学，2006,03.

[20]刘丹.基于 GPU 的分子动力学模拟方法研究[D].武汉：武汉理工大学，2010,08.

[21]徐显.基于 GPU 硬件加速的医学图像融合研究[D].上海：上海交通大学，2010,07.

[22]周海兵.基于 GPU 加速的 Otsu 图像阈值分割算法实现[D].大连：大连理工大学，2010,01.

[23]付娟.信息安全算法的 GPU 高速实现[D].南昌：南昌大学,2010,05.

[24]SN G B，GAO G P，LI S P，et al. Accelerate Video Decoding With Generic GPU[J]. IEEE Transactions on Circuits and Systems for Video Technology，2005，15(5): 685—693.

[25]FANG B，SHEN G B，LI S P，et al. Techniques for Efficient DCT / IDCT Implementation on Generic GPU[C].IEEE International Symposium on Circuits and Systems，2005，2: 1126-1129.

[26]PIETERS B，VAN R D，DE N W，et al. Performance Evaluation of H.264 / AVC Decoding and Visualization using the GPU[C]. Applications of Digital Image Processing XXX，2007，6696(6): 13.

[27]PIETERS B，VAN R D，DE N W，et al. Motion Compensation and Reconstruction of H. 264 / AVC Video Bit streams using the GPU[C]. Eighth International Workshop on Image Analysis for Multimedia Interactive Services，2007: 69—69.

[28]贺克军.H.264 运动估计算法的研究[J].广西工学院学报.2007,08.

[29]李道盛.基于 H.264 的快速运动估计算法的研究[D].上海：上海交通大学,2010,07.

[30]LIN H Y，CHAO Y C，CHEN C H，et al. Combined 2 D transform and quantization

architecture for H. 264 Video Coders[C]. 1EEE International Symposium on Circuits and Systems，2005，2: 1802—1805.

[31]WANG T C，HUANG Y W，FANG H C，etal. Parallel 4x4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264[C]. Proceedings of the 2003 International Symposium on Circuits and Systems,2003,2: 11800-11803.

[32]LOU J,JAGMOHAN A, HE D, etal. Statistical analysis based H. 264 high profile deblocking speedup[C]. 1EEE International Symposium on Circuits and Systems, 2007: 3134-3146.

[33]LOU J, JAGMOHAN A, HE D, etal. High speed H.264 high profile deblocking using statistical analysis and logic optimization[C]. IEEE International Conference on Multimedia and Expo, 2007: 1918—1921.

[34]魏磊.AVS 视频编码器中熵编码及运动估计的研究与优化[D].山东:山东大学,2010,07.

[35]江静.AVS 熵编码技术研究及其在 DSP 上的实现[D].武汉:华中科技大学,2009,09.

[36] Kim H, Altunbasak Y. Low-complexity macro block mode selection for H.264/AVC encoders[C] //Proceedings of the 2004 International Conference on Image Processing. Singapore: ICIP, 2004: 765-768.

[37]PAN F, LIN X, Rahardja S, eta.l Fast mode decision for intra prediction[C] // JVT 7thMeeting. Pattaya: IEEE. 2003.

[38] TsaiA C, PaulA, WANG J C, etal efficient intra prediction in H. 264 based on intensity gradient approach[C] //Circuits

[39]Bojun M,Oscar C,et al.Efficient Intra Prediction Mode Selection for 4×4 Blocks in H.264 [A.In ICME'03 Proceedings of the 2003 International Conference on Multimedia and Expo [C.Baltimore,Maryland:IEEE,2003.521-524.

[40]Jeon B,Lee J.Fast mode decision for H.264.[C].JVT 10th Meeting.Waikoloa,Hawaii,2003-12.

[41]管梅.H.264/AVC 中 4×4 块的快速帧内预测算法的研究[J].科技信息报,2010,04.

[42]杨军,自适应阈值的宏块MAD快速帧内算法[J].计算机工程与应用，2009

[43]王维哲,周兵.H.264编码中的帧内预测模式选择算法[J].计算机工程，2008.

[44] Jianjun Li and Esam Abdel-Raheem, "Fast implementation of H.264 4x4 intra prediction", *IEICE Electron. Express*, Vol. 7, No. 5, pp.332-338, (2010).

[45] gai-Man Cheung,Au, Parallel rate-distortion optimized intra mode decision on multi-core graphics processors using greedy-based encoding orders[C].// Proceedings of the IEEE International Conference on Image Processing,2009,2309-2312.

[46]Jamil-Ur-Rehman,Ye Zhang. Fast intra prediction mode decision using parallel processing[C],// Proceedings of the IEEE International Conference on Machine Learning and Cybernetics,2005,5094-5098.

[47]裴朝科,杨树元.一种提前进行模式选择的H.264编码优化算法[J].微计算机应用, 2009,12

[48]G1Y1 Kim, Y1H1 Moon, J1 H1 Kim1An Early Detection of All Zero DCT Blocks in H.264 Proc1 IEEE Int1 Conf1 Image Processing,2004,(10):453-456

[49]Hanli1 W ang1Hybrid Model to Detect Zero Quantized DCT Coefficients in H.264 IEEE Transactions on Multimedia,2007,9(6):

[50]杨云飞,韩彦芳,张定会. H.264整像素快速运动估计算法研究.[J]计算机工程与应用,2009

[51]Wang Z,Bovik AC,Sheikh HR,Simoncelli EP.Image quality assessment:From error visibility to structural similarity.IEEE Transactions on Image Processing,2004,13(4):600-612

[52]Wang Z,L u L,Bovik A C.Video quality assessment using structural distortion measurement//Proceedings of the IEEE International Conference on Image Processing.Rochester,2002:65-68

[53]杨春玲,王华兴. 基于结构相似的H.264帧间预测改进算法[J].计算机学报, 2009,08

[54]HO C W，AU C，. CHAN G，eta1. Motion Estimation for H.264 / AVC using Programmable Graphics Hardware[C]. IEEE International Conference on Multimedia

and Expo，2006：2049-2052.

[55]LIN Y C, LI P L, CHANG C H, etal. Multi—pass algorithm of motion estimation in video encoding for generic GPU[C]. IEEE International Symposium on Circuits and Systems，2006：4451—4454.

[56]CHEN W N，HANG H M. H.264 / AVC motion estimation implementation on Compute Unified Device Architecture(CUDA)[C]. IEEE International Conference on Multimedia and Expo，2008：697—700.

[57] Aleksandar Colic, Hari Kalva, Borko Furht, Exploring NVIDIA-CUDA for Video Coding[J]1827-1836.

[58]江辉,基于GPU的H. 264视频并行编解码器[D].大连：大连理工大学,2009.

[59]房波.基于通用可编程GPU的视频编解码器架构算法与实现[D].浙江：浙江大学,2005.

[60]陈佐,陈汉.运动估计搜索算法的 CUDA 优化与实现[J].计算机工程与应用,2010.

[61]甘新标,沈立.基于CUDA的并行全搜索运动估计算法[J].计算机辅助设计与图形学学报,2010.

致 谢

三年研究生生活即将接近尾声，回首三年的欢乐时光，真的有太多的人在学习、生活上给与了我无私的关心和帮助，在这里我要诚挚的感谢你们。

首先要感谢我的导师满家巨教授。本文是在满老师的精心指导下完成的，从论文的选题、设计方案直至完成论文的整个过程中，都得到了满老师的耐心指导。满老师不仅教给了我知识，还教会了我很多做人的道理。他平易近人，待人热情，治学严谨，他的很多品质都值得钦佩。

另外，我还要向曾教过我知识和给予过我帮助的数计院老师们，我还要感谢院领导在我的学习和生活组织中给予的指导和帮助。

其次，还感谢我的同门邹有，赵海媛，朱伟以及计算机方向的同学们。我总是在学习和生活上麻烦你们。当我遇到难题时，你们总能耐心的给我分析，并给我提出了许多宝贵的意见和建议。在这里我还要特别感谢师兄兰敏，师弟伍毅子和丁玉高，还有 H.264 乐园视频论坛版主 firstime，他们在我论文课题研究的过程中给予了我很多无私的帮助。

最后，深深感谢我的父母，是你们给我创造了优越的条件，使我在学习的道路上乐观向上、勇往直前。

再一次向关心、支持和帮助我的所有人表示感谢！

湖南师范大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：胡佳　　2011 年 6 月 21 日

湖南师范大学学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，研究生在校攻读学位期间论文工作的知识产权单位属湖南师范大学。同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南师范大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密☑。

（请在以上相应方框内打“✓”）

作者签名：胡佳　　日期：2011 年 6 月 21 日

导师签名：　　日期：　　年　　月　　日

H.264视频编码的并行实现

作者：胡佳
学位授予单位：湖南师范大学