

本科生毕业设计

|  |
| --- |
| 高效视频编码器在多核异构系统上的并行优化 |

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | 卓越1501 |
| 姓 名 | 董子恒 |
| 学 号 | U201514746 |
| 指导教师 | 郭红星 |

2019年06月10日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 年 月 日

**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障、使用学位论文的规定，同意学校保留并向有关学位论文管理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。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，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、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

摘 要

新一代视频编码标准HEVC于2013年正式发布。与前代标准H.264/AVC标准相比，HEVC标准在编码压缩率上取得了大幅提升，但同时也增加了约70%的编码时间。这一耗时的增加主要源自于HEVC标准中更加复杂的图像划分方式与计算过程。正是由于其低下的运行速度，HEVC目前仍难以在市场中被大规模使用。以HEVC标准测试模型为参照，基于CPU-GPU的多核异构平台，利用CUDA编程平台对HEVC中的复杂运算进行了并行优化，充分利用了目前家用电脑中日渐常见的GPU计算资源。

对于HEVC中的两大主要耗时模块：帧内预测模块化和变换量化模块，分别选取其中核心功能函数进行并行算法的设计、实现与测试。在帧内预测模块中，对哈达玛失真函数实现并比较了2种优化方法，加速比达到919%；对预测值计算函数实现了并行优化，加速比达到170%。在变换量化模块中，对离散余弦变换函数设计并比较了2种优化方法，加速比达到811%；对离散正弦变换函数实现并行优化，加速比达到136%。最终试验结果表明，在模块和编码器层面，对帧内预测模块整体提高运行速度22%，对变换量化模块整体提高运行速度25.8%，对编码器整体整体提高运行速度19.7%。

**关键词**：HEVC；多核异构平台；帧内预测；并行优化

Abstract

The new generation video coding standard HEVC was officially released in 2013. Compared with the previous generation standard H.264/AVC, the HEVC standard has achieved a significant increase in encoding compression ratio, but it also increases the encoding time by about 70%. This time-consuming increase is mainly due to the more complicated image partitioning and calculation process in the HEVC standard. It is because of its low operating speed that HEVC is still difficult to use on a large scale in the market. Taking the HEVC standard test model as a reference, based on the CPU-GPU multi-core heterogeneous platform, the CUDA programming platform is used to optimize the complex operations in HEVC in parallel, making full use of the increasingly common GPU computing resources in home computers.

For the two main time-consuming modules in HEVC: intra prediction modularization and transform quantization module, the core function function is selected to design, implement and test the parallel algorithm. In the intra prediction module, two optimization methods are implemented and compared for the Hadamard distortion function, and the acceleration ratio is 919%; the parallel calculation is performed on the prediction value calculation function, and the acceleration ratio is 170%. In the transform quantization module, two optimization methods are designed and compared for the discrete cosine transform function, and the acceleration ratio is 811%. The discrete sine transform function is optimized in parallel with an acceleration ratio of 136%. The final test results show that at the module and encoder level, the overall operation speed of the intra prediction module is increased by 22%, the overall operation speed of the transform quantization module is increased by 25.8%, and the overall operation speed of the encoder is increased by 19.7%.

**Keywords:** HEVC; multi-core heterogeneous platform; intra prediction; parallel optimization

目 录

[摘 要 I](#_Toc9822175)

[Abstract II](#_Toc9822176)

[1 绪 论 2](#_Toc9822177)

[1.1 课题背景 2](#_Toc9822178)

[1.2 国内外研究现状 3](#_Toc9822179)

[1.3 研究目的和主要内容 5](#_Toc9822180)

[1.4 论文结构 5](#_Toc9822181)

[2 关键技术 7](#_Toc9822182)

[2.1 HEVC关键技术 7](#_Toc9822183)

[2.2 本章小结 16](#_Toc9822184)

[3 并行优化算法设计与实现 17](#_Toc9822185)

[3.1 HM模块分析 17](#_Toc9822186)

[3.2 帧内预测模块优化 22](#_Toc9822187)

[3.3 变换量化模块优化 29](#_Toc9822188)

[3.4 本章小结 35](#_Toc9822189)

[4 实验结果分析 36](#_Toc9822190)

[4.1 哈达玛失真函数并行优化实验结果与分析 36](#_Toc9822191)

[4.2 预测值函数并行优化实验结果与分析 37](#_Toc9822192)

[4.3 帧内预测模块整体并行优化实验结果与分析 38](#_Toc9822193)

[4.4 快速DST函数并行优化实验结果与分析 39](#_Toc9822194)

[4.5 蝶形DCT变换函数并行优化实验结果与分析 40](#_Toc9822195)

[4.6 HM整体并行优化实验结果与分析 42](#_Toc9822196)

[4.7 本章小结 43](#_Toc9822197)

[5 总结与展望 44](#_Toc9822198)

[致 谢 46](#_Toc9822199)

[参考文献 47](#_Toc9822200)

# 绪 论

本章我们首先介绍了当前视频编码标准的发展趋势，针对新一代视频编码标准HEVC简要介绍了其优势与不足，然后介绍了图像处理单元GPU技术的发展现状，并概述了利用GPU对HEVC标准编码器进行加速的相关研究工作，并对本文的主要内容和工作意义进行了说明。

## 课题背景

近年来，随着视频技术以及Internet技术的迅猛发展，视频应用已经从传统的电视技术转向了数字视频广播、便携摄像以及移动无线视频等领域，更加深刻地改变和影响着日常生活。据统计，视频流数据再2015年占到了整个互联网流量的90%左右，互联网几乎已经成为视频网了[1]。

另一方面，虽然网络网络带宽和传输能力增加迅速，但仍远不能满足海量视频数据的传输和存储要求。以常见的1280×720视频序列为例，如果三个色度分量都用8bit表示，则一帧图像的原始数据量约为22Mbits。假如传输帧率为25fps，视频时长为一个小时。则整个视频的大小将达到约2Tbits，这是当前网络带宽和存储设备都难以承受的。因此如何高效地压缩视频，一直以来都是视频处理相关技术研究的核心课题。从20实际80年代开始，国际电信联盟电信标准化组织（ITU-T）和国际标准化组织（ISO）/国际电工委员会（IEC）就一直致力于视频编码技术的标准化工作[2]。由视频编码专家组（VCEG）和动态图像专家组（MPEG）先后推出了的H.26x系列和MPEG系列等视频编码标准。其中，在2003年发布的H.264/AVC标准由于具有良好的视频压缩率、网络适配性和防出错能力，因此被广泛应用于视频存储、流媒体等领域，成为目前市场占用率最高的视频编码标准之一[3]。

然而，随着目前高清、超高清视频（4K，8K）的逐渐普及，以往以H.264为代表的视频编码标准收到了越来越大的挑战，主要表现为视频编码效率的提高跟不上视频数据量的增加。为了进一步应对这一趋势，2010年，VCEG与MPEG合作组件了视频编码联合小组（JCT-VC），致力于研发新一代“高效视频编码”（High Efficiency Video Coding，HEVC），并在2013年1约正式发布了该标准。与前一代主流编码标准H.264/AVC相比，在保证相同图像质量下，HEVC可以减少约50%的码率[4]。但是，HEVC为了获得优异的压缩而造成的超高的计算复杂度对其推广产生了不小的影响。2018年，HEVC的市场占有率从3%提高为9%，仍远低于H.264的81%[5]。因此，如何提高编码效率成为HEVC相关研究的热门方向。

近年来，随着GPU（Graphics Processing Unit）技术的高速发展，GPU已经不仅仅被用于图像渲染，而被更广泛地用于多种不同地计算任务，包括机器学习、计算机仿真和通信技术等等[6]。GPU在并行计算方面相比于CPU具有明显优势，这主要源于GPU与CPU在处理任务方式上地区别。GPU专为复杂的数学和几何运算而设计，其运算核心由大量较小而高效的核心组成，而CPU一般由一个或几个专门顺序执行的串行处理而优化的核心组成。2006年，NVDIA公司推出了针对其旗下GPU产品的编程平台CUDA（Computer Unified Device Architecture，统一计算框架）[7]。随着这一编程环境的出现和发展，GPU的通用计算编程实用性大大提高，复杂度不断降低，使得GPU真正成为了一种实用的通用并行计算资源。这也使得利用CUDA平台来对HEVC中的相关并行计算进行优化加速成为可能。

## 国内外研究现状

目前关于HEVC优化的方式主要有两大方向，分别是针对帧内预测过程和帧间预测过程。

对于帧间预测的大部分工作都是加速运动估计。在参考文献[8]中，使用ME的加速运动来改变编码树单元（CTU）之间的依赖性并开发出一种搜索模式，该搜索模式比完全搜索需要更少的数据;搜索结果的质量在CPU中得到改进。参考文献[9]提出了一种完整的HEVC解码解决方案用于异构CPU + GPU系统，其中熵解码器在CPU上执行，而剩余的内核在GPU上执行。 参考文献[10]在编码器上提出了两种新的方法，用于通过将运动估计过程分成多个步骤并使用前面步骤的结果来放宽数据依赖性，而不是在同一步骤中相邻块的结果。这样可以在GPU上充分利用许多处理器内核，同时保持压缩效率。最后，文献[11]描述了基于图形处理单元（GPU）上的关键点检测的快速帧内编码单元（CU）大小决策框架。在此框架中，首先将原始帧发送到GPU，然后使用多个线程进行关键点检测，这样即使在实时系统中也能够避免引入额外的计算复杂性。这些先前的工作和我们提出的解决方案之间的主要差异在于我们使用我们自己的帧内预测算法MDV-SW;它从一开始就被设计为易于并行化，并且已经证明可靠地将计算减少了30%，同时对比特率的影响最小。

在帧内预测方向，同样有许多减少处理时间的方法。这些方法可以分为三个方面。第一种方法采用快速块大小决策，这减少了预测块大小的复杂性。为了解决最新HEVC标准中帧内预测的高计算需求，文献[12]提出了一种快速的硬件友好的帧内块大小选择算法，这种算法具有简单的梯度计算和自下而上的结构。简单梯度计算和阈值T用于将最大CTU分成两种具有32×32和8×8大小的CU，这可以平均节省大量编码时间，并且BD速率增加很少。第二种是快速模式决策，可以降低帧内预测的计算复杂度，并在H.264中进行了大量的研究。文献[13]利用空间相关性来减少用于RDO过程的方向数量。发现相邻的块通常在自然图片中保持相似的纹理。因此，当前块的最佳帧内预测可以与其相邻块具有强相关性。基于此，当前块的最佳帧内方向的条件概率被用作其相邻块的最可能模式（MPM），这可以帮助减少方向的数量并提高准确度。此外，文献[14]在MPM的基础上通过基于组的算法进一步减少了方向的数量。模式以小于或等于2的差值合并到同一组中，并在每组和MPM中选择最小RMD-成本（粗略模式决策成本）模式内的最终候选模式。文献[15]提出了一种基于模式滤波方案的快速预处理阶段，以减少角度预测模式的最大数量。该算法将角度预测模式的最大数量从34减少到9，并实现50％的编码时间减少。第三个采用并行帧内预测。尽管上述两种方法都取得了良好的效果，但它们仅限于CPU上的串行计算过程。随着GPU的推广和发展以及并行机制的发展，并行帧内预测取得了良好的效果。 文献[16]提出了最有希望的并行化提议的一些有效实现，即条和波前并行处理（WPP），它们被广泛用于H.264中的并行帧内预测。提出了一种称为重叠波前（OWF）的新方法，该方法在WPP的基础上对垂直运动矢量的最大长度进行了限制。在文献[17]中，GPU执行由32个并行线程组成，这些线程称为warp并分组在多个线程块（ThB）中。分配具有八个warp的单个ThB来处理具有64×64个CTU的行，因此每个warp计算八个连续的像素行。通过设置扭曲定位（WP）以记录每个扭曲中最近完成的子块的水平位置，可以存储重建的相邻块以实现并行编码。文献[18]对HEVC帧内编码的细粒度并行性进行了全面分析。但他们只是利用了CTU级并行化，而这种并行化并不是细粒度并行性。通常，上述并行方法是粗粒度的，仅仅很有限地利用了GPU。需要细粒度的方法以获得更多潜力。文献[19]在GPU上实现了像素级细粒度并行性，用于H.264的帧内预测。它是通过将预测公式转换为统一形式，引入统一预测器阵列和建立系数查找表来实现的。该工作对H.264的帧内预测产生了很好的效果。

## 研究目的和主要内容

本文基于HEVC测试模型（HEVC Test Model，HM）进行研究，主要针对其中的帧内预测模块进行并行优化。对于帧内编码模块中最为耗时的角度预测、率失真函数、变换函数分别提出了并行算法，在CUDA上加以实现，最后编译为静态链接库整合进HM整体代码中。并且针对不同的视频测试序列，对优化后的帧内编码模块进行了测试和分析。最终能较为显著地提高HEVC编码器的编码效率，并符合并行算法的理论设计预期。

## 论文结构

本文主要内容如下：

第一章首先介绍了视频编码技术的发展历程的目前面临的挑战，然后介绍了新一代高效视频编码标准HEVC的特点，以及其在广泛使用中面临的问题。之后对GPU硬件和编程平台CUDA的发展做了简单介绍，并指出了利用CUDA平台对HEVC进行并行优化的可行性。

第二章中，结合HEVC编码标准和相关理论，详细介绍HM的整体架构和设计细节，对其中核心的编码树结构、帧间预测、帧内预测模块进行分析。然后对CUDA编程平台的软硬件架构和使用方法做进一步说明。

第三章主要针对帧内编码部分进行研究，对角度预测、失真函数、变换函数的流程进行分析，并提出针对每一个算法相对应的并行优化方法。

第四章主要对优化后的HM代码在不同的视频序列下进行测试，并结合算法的理论并行度对实验结果进行分析。

第五章主要对全文工作进行总结，指出其中不足指出，对进一步可能的改进方向进行展望。

# 关键技术

本章主要对课题中涉及的HEVC主要技术进行介绍。由于主要主要针对帧内编码过程和变换计算进行优化，因此着重介绍HEVC的帧内预测相关技术以及变换技术。

## HEVC关键技术

### HEVC编码整体框架

HEVC视频编码层沿用以往视频编码标准的混合编码方式，其整体编码框架如图2‑1所示，其输入为原始视频序列，输出为符合HEVC标注你的比特流。简要的编码过程如下。

（1）将每一帧图像划分为大小不同的图像块单元，并将相应的块划分信息加入到码流中，传到解码器。

（2）对每个单元进行帧内或帧间预测，原始像素值和预测值相减形成该单元的残差；若为帧间预测，则进行运动估计和运动补偿，对需要用到的重建图像要提前进行去方块滤波和自适应采样值补偿（ASO）滤波。

（3）对每个单元的残差进行整数变换（近似离散余弦变换和正弦变换），对得到的变换系参数继续进行量化和扫描。

（4）对量化后的变化系数、预测信息、模式信息、运动信息和头信息等进行熵编码，形成压缩的视频码流（语法元素）输出。



图2‑1 HEVC整体编码流程

### HEVC编码结构

在HEVC中，一个很重要的革新之处就是其图像分块方式。HEVC针对预测和变换编码而对图像采取了一种基于四叉树的划分方式。其基本划分层次如图2‑2所示。



图2‑2 HEVC图像划分方式

视频序列由若干时间连续的图像组成，在对这些图像进行处理时，HEVC首先将其分为若干图像组（Group Of Picture，GOP），其大小可以通过配置文件进行设置。每一组GOP由多帧图像组成，每一帧图像即为HEVC中四叉树划分的基本单位，每一帧图像经过划分，形成覆盖全帧的多个同样尺寸的编码树块（Coding Tree Block，CTB）。CTB还可以进一步分割为更小的编码块（Coding Block，CB）。CB时HEVC中进行视频编码算法的基本单位，它还可以划分为预测块（Prediction Block，PB）和变换块（Transform Block，TB）。接下来依次介绍它们的划分方式。

1. 编码树块（CTB）和编码树单元（CTU）的划分

HEVC将一帧编码图像划分为同意大小、相邻但不重叠的2N×2N样点的编码快CTB，这一点类似H.264/AVC中的宏块（Macro Block，MB）。CTB的尺寸可以为16×16、32×32或64×64。同一位置的亮度CTB和2块色度CTB，以及相应的语法元素和所包含的CU形成一个CTU。

CTU可以按照四叉树结构分解为若干方型编码单元（CU）,统一层次的CU必须时统一尺寸的4个方块，最多可以有4层，即8×8、16×16、32×32或64×64。如果不分解，则一个CTU仅包含一个CU，此时亮度CTU的尺寸就是亮度CB的最大尺寸。每个CU包含一个亮度编码块（CB）以及两个色度编码块（CB）。同时还包含相应的语法元素，如预测模式（帧内、帧间）、PU划分、从属的PU和TU信息等。CU时决定进行帧内预测还是帧间预测的单元，因此整个CU只会有一种预测模式[20]。

CTB与CB之间的关系可以用一棵四叉树表示。CTB为树的根节点，往下还可以划分为更小的方块，由于分一层划分都是等分为四块，因此在树结构上形成四个分支，最后构成四叉树。以64×64尺寸的CTB为例，其划分方式如图2‑3所示，对应四叉树结构如图2‑4所示，扫描顺序按图2‑3中索引顺序。



图2‑3 HEVC编码树块结构



图2‑4 HEVC编码四叉树结构

1. 预测单元的划分

预测单元PU是HEVC进行预测运算的基本单元，只能定义在不能继续划分的最底层CU中。CU决定了其包含的所有PU的预测方式，包括帧内预测和帧间预测[21]。一个CU可以划分为一个或多个预测单元PU。CU到PU的划分仅允许一层。若CU的大小为2N×2N，当金星帧内预测，则可选的PU模式有2N×2N和N×N两种；如果进行帧间预测，则划分方式除了2N×2N、N×N、2N×N、N×2N以外，还有4种非对称划分方式[22]。除此之外，帧间预测的跳过模式中只允许2N×2N的划分。

PU具体的划分方式如图2‑5所示。



图2‑5 预测单元PU划分方式

1. 变换单元的划分

变换单元是进行变换和量化操作的基本单元，和PU划分类似，他也是在CU的基础上进行划分的，但它不受所在CU的预测单元（PU）划分的限制。TU的划分方式也是一种四叉树划分。HEVC中TU的尺寸可以为4×4、8×8或32×32，这是因为最大DCT变换运算的尺寸为32×32。变换单元的最大尺寸以及四叉树的层级可以根据不同的应用进行相应的配置[23]，对实时性或复杂度较低的应用可以通过增加四叉树的深度来提高编码效率[24]。如图2‑6所示是一个深度为3的TU四叉树划分。



图2‑6 编码单元结构

### HEVC帧内预测编码

帧内预测编码利用了视频序列每一帧内的空间相关性，利用当前图像中已经编码的部分对当前正在编码的块进行像素值预测。这一思路基于同一幅图像中存在的平坦区域和纹理区域，利用不同位置上像素块的相关性，从而达到去除空间域的冗余，达到压缩编码的目的。

HEVC的帧内预测模式相比于H.264/AVC，很重要的一个变化就是添加了更加细分的预测角度，大大增加了预测方向。如图2‑7所示，HEVC中共有33种角度预测模式，联通直流和平面模式，共35种预测模式。



图2‑7 35种帧内预测模式

对于每种预测模式，帧间预测主要分为三步：参考像素的准备，参考像素的平滑滤波，以及根据参考像素计算出预测像素的预测值。下面分别对帧内预测的每个过程进行简要阐述。

1. 参考像素的准备

与H.264/AVC不同，HEVC规定左下方、右上方的像素也可以作为当前预测块地的参考像素块[25]，这是因为HEVC需要支持不同大小的编码快，所以常出现左下方和右上方像素块已经完成编码的情况。如图2‑8所示，设参考像素坐标为p[x][y]，其中x为横坐标，y为纵坐标，则左边参考像素的坐标为x=-1，y=-1，-2,…,-nTbS×2+1，上方的参考像素坐标为x=0，1，…,nTbS×2-1，y=-1。确定像素坐标后，HEVC根据坐标的可用性采用三种不同的填充方式。如果所有参考像素p[x][y]都可用，则p[x][y]直接取重建图像中对应像素的值；如果所有p[x][y]都被标记为不可用，则所有p[x][y]都填充为，其中bitDepth为比特深度，比如在常见的比特深度为8时，p[x][y]填充为128；如果至少一个p[x][y]被标记为不可用，HEVC根据相应规则将其临近可用的像素值填充进p[x][y]。



图2‑8 帧内预测参考像素

1. 参考像素的平滑滤波

为了提高帧内预测的效率，HEVC对8×8或更大的帧内PU的部分参考像素在预测前进行平滑滤波处理，从而减少噪声对预测的影响。对于不同的模式，选取需要进行平滑滤波的块也有所不同。对于4×4预测块的参考像素集，不需要进行平滑滤波。对于其他尺寸，如果是直流预测模式（DC），也不需要平滑滤波，如果是平面预测模式（Planner），则需要平滑滤波。对于角度模式，当参考像素及尺寸为8×8，只需要对3个对角方向的模式，即2、18、34模式进行平滑滤波。对16×16预测块的参考像素，出9、10、11和25、26、27外，其余模式都需要平滑滤波。对32×32预测块，有常规平滑滤波和强滤波两种处理方式。其中常规平滑滤波可通过3抽头滤波器来实现，强平滑滤波则用一个双线性插值滤波器实现[26]。

1. 根据参考像素计算预测块

完成像素值填充和平滑滤波之后，HEVC针对不同的预测模式，根据参考像素值计算出预测块的像素值。其种主要分为角度类预测和非角度类预测算法，非角度类预测又包含直流和平面预测两种。当预测模式为平面模式时，预测块p由像素由a、b、c、d个参考像素来决定，具体如图2‑9所示。



图2‑9 预测块计算方式

记，则预测值的计算公式为:

(2‑1)

当预测模式为直流模式时，首先对所有像素块计算怕平均值dcVal作为基本直流量。计算公式为：

(2‑2)

当使用角度模式时，左侧与上方的模式预测值计算方法相似[27]。从（预测模式示意图）种可以看出，对于编号为18～34的模式 ，各个模式的角度分布是不均匀的，越是靠近垂直和水平两个方向，分布越密集。这以设计是因为自然界中大量的景物是垂直或水平分布的。在角度预测模式种，编号2～18称为水平方向类预测模式，18～34称为垂直方向类预测模式。一水平方向类的上方角度预测模式为例，介绍帧内预测值的计算方式。对角度模式modelIntra，尺寸为nTbS的块中某一位置的像素计算可分为三步。

首先找出该预测快的参考像素集，的值根据预测模式角度参数intraPredAngle和相应的反角度参数invAngle确定。具体如下：

(2‑3)

然后设置不同模式的索引Idx和权重Fact，具体计算方法如下：

(2‑4)

(2‑5)

最后计算预测值predSample,。具体计算方法如下，其中预测值predSample记作pS：

(2‑6)

### HEVC变换技术

HEVC采用混合编码框架，首先对视频数据进行预测，然后对预测结果的残差进行整数变换，再对变换后的系数进行量化，从而提取低频信息，达到提高编码压缩效率的目的[28]。下面依次介绍基本的量化和变换算法。

1. 二维离散余弦（DCT）矩阵与离散变换

离散余弦变换（Discrete Cosine Transform，DCT）的物理意义与傅里叶变换基本一致。由于DCT的正变换与逆变换核都是相同的，并且相比于DFT而言没有复杂的复数运算，因此在图像领域使用广泛[29]。在二维DCT变换中，N×N的信号矩阵f，经过变换后形成矩阵F，其正变换和逆变换公式如下：

(2‑7)

(2‑8)

其中C为二维DCT变换矩阵，对于C的第u行第x个元素记为，则该元素可表示为：

(2‑9)

其中x,u=0,1,,N-1。

1. 二维离散正弦(DST)矩阵与离散正弦变换

离散正弦变换（Discrete Sine Transform，DST）是一种与傅里叶变换相关的变换，本质是将一个一维实数信号进行奇对称延拓，长度约为原来的两倍，对这个延长后的信号进行离散傅里叶变换。与DCT变换类似，二维DST变换也可以写为矩阵乘法形式，对于（N-1）×（N-1）大小的信号矩阵f，其DST变换的正逆过程分别为：

(2‑10)

(2‑11)

其中S为二维DST变换矩阵，对于S中的第u行第x个元素，该元素可表示为：

(2‑12)

其中x,u=0,1,,N-1。

## 本章小结

本章首先分析了HEVC编码器的整体流程，然后对帧内预测和变换量化的相关简述进行了介绍。着重介绍了HEVC中的四叉树编码结构、角度预测模式等HEVC所特有的技术和原理。

# 并行优化算法设计与实现

基于上一章中对于HEVC关键技术和CUDA编程相关技术的理解，本章首先简要介绍官方HEVC测试模型（HEVC Test Model，HM）运行流程，分析其中主要的耗时模块和函数。接下来针对其中的帧内预测模块和变换量化两大模块进行进一步研究。针对帧内预测模块，首先分析其整体逻辑，设计合适的并行算法，然后分别在对其中的率失真函数与角度预测函数进行并行优化算法的设计和实现，然后针对整个模块进行并行优化并研究这一算法下的理论并行度。针对量化变换模块，同样首先分析其整体结构，然后针对其中的DST变换函数、蝶形快速变换函数分别设计和实现并行优化算法，并计算和对比了各个算法的理论并行度。

## HM模块分析

### HM运行流程

HEVC测试模型（HEVC Test Model，HM）是由德国通讯技术研究所（HHI）发布的官方HEVC标准编解码器实现。HM使用C++语言开发，包含了多个工程，通过跟踪和调试HM代码，可以发现HM各个工程和模块之间的关联关系。其中涉及到视频编码的主要为：包含编码器和编码器公共应用函数的TAppCommon，包含了编码器应用函数的TAppEncoder，包含编码器和解码器公用库函数的TLibCommon，包含编码器库函数的TLibEncoder。下面结合HEVC标准和HM的代码实现，以全帧内预测模式为例，简述其运行流程。

（1）HM中对整个编码过程封装了cTAppEncoder类，整个编码流程在其成员函数encode中实现。在encode编码函数中，处理的编码对象为视频序列划分之后的图像组GOP，在HM中通过图像组末尾iPOCLast以及图像组宽度iNumPicRcvd来标识。GOP依次经copressGOP划分后形成slice，经compressSlice划分后形成CTU、经compressCTU划分后形成CU，最后对CU再进行实际的编码操作。

（2）由于HM配置为全帧内模式，因此CU的编码中仅需要进行帧内预测即可，该过程由xCheckRDCostIntra函数完成，HM接下来分别对CU进行亮度预测和色度预测，其中主要为亮度预测，函数estIntraPredLumaQT主要完成亮度预测。按照HEVC标准，帧内预测的流程需要在平面模式、直流模式以及33种角度预测模式种选择“最佳”模式来为CU进一步进行编码[30]。这一过程由xRecurIntraCodingLumaQT实现。对应HEVC标准，该函数中遍历了35种预测模式，对每种模式分别调用角度预测函数predIntraAng和率失真函数xGetHADs，形成候选模式集合，然后再从中选出最佳的编码模式。

（3）完成帧内预测之后，HM根据HEVC标准，将CU划分为TU，进行变化和量化操作。这一操作的入口函数为xIntraCodingTUBlock，其中量化操作的实现为xQuant，变换操作的实现为xT。

（4）最后进行熵编码，并输出为编码码流。

### HM模块运行时间分析

为了明确优化方向和优化重点，需要首先了解和分析HM代码中各个模块的耗时情况，从而找出其中适合优化且有优化价值的部分。此处，对于HM的模块性能分析主要分为两部分，第一部分为，对于不同视频测试序列，测试和分析HM代码中不同模块的耗时情况。第二部分为，针对其中每个模块，分析其中主要功能函数的耗时情况。测试环境如表3‑1所示：

表3‑1 测试条件表

|  |  |
| --- | --- |
| 测试环境项 | 测试环境值 |
| CPU | Intel i5-4210H 2.9GHz |
| GPU | NVDIA GTX860M |
| 操作系统 | Window8.1 64位专业版 |
| 编码总帧数 | 60 |
| 编译环境 | Visual Studio 2017 |

对于不同分辨率和帧率的测试序列，在全帧内模式下，HM代码中各个模块的运行时间如图3‑1所示，其运行时间在总时间中所占百分比如图3‑2所示，需要指出，图中时间单位为100秒，下文图表中单位与此处一样，不再说明。

图3‑1 HM各模块运行时间

图3‑2 HM各模块运行时间占比

可以看出，在不同的测试序列下，各个模块的运行时间占比基本上一致，在全帧内编码模式下，帧内预测模块的运行时间约占总体时间的81%，变换量化模块运行时间约占总体时间的10%。因此在下面的各模块分别分析中，不再需要针对不同的测试序列一一测试，而只需要使用同一个视频测试序列即可。需要指出的是，由于在HM中，整个编码流程被封装在一个xCheckRDCostIntra函数中，而帧内预测模块和变换量化模块并不是严格上分离的，因此测试中统计的所谓“帧内预测模块”和“变换量化模块”，事实上是指，根据HEVC编码标准，分别完成这两部分功能的函数集合。其中帧内预测模块具体包含的函数以及各自的运行时间如图3‑3所示，其占整个模块运行时间的百分比如图3‑4所示：

图3‑3 帧内预测模块函数运行时间

图3‑4 帧内预测模块函数运行时间占比

从图中可知，对于不同测试序列，帧内预测模块中的哈达玛失真函数和预测值计算函数运行时间虽然有很大变化，但是所占的比重基本不变，分别为13%左右和12%左右。

对于变换量化模块，由于HM中实现了不同尺寸的变换函数，因此对每种尺寸的变换分别进行统计，便于之后的优化结果分析。该模块中各个函数的运行时间如图3‑5所示，所占百分比如图3‑6所示：

图3‑5 变换量化模块函数运行时间

图3‑6 变换量化模块函数运行时间占比

基于以上实验，可以发现对于不同测试序列，HM的变换量化模块中各个函数的运行时间虽然有很大变化，但占比基本保持不变，DST变换占约18%，4×4DCT变换约占2%，8×8DCT约占8%，16×16DCT约占10%，32×32DCT约占10%。

基于以上对模块耗时情况以及各模块内部功能函数的分析，便可以进一步确定具体优化内容。在全帧内编码模式下，HM中帧内预测模块和变换量化模块的运行时间共同占到了总运行时间的90%左右，因而都这两个模块进行并行优化对于提高HM整体速度是最有意义的。

而对于帧内预测模块，哈达玛失真函数和预测值计算函数总计占到了运行时间的25%左右，对于变换量化模块，其中5个变换函数分别占到了模块运行时间的50%左右。因此，对于帧内预测模块优化最大加速比为1/（1-25%）≈1.333，对于变换量化模块的优化最大加速比为1/（1-50%）=2。因此，本文选取的优化部分对于整个HM的优化最大加速比为1/（0.8/1.33+0.1/2+0.1）≈1.3307。

## 帧内预测模块优化

### 帧内预测模块分析

与HEVC标准一致，HM代码中帧内预测也分为亮度预测与色度预测，其流程大致相同，此处以亮度预测函数estIntraPredLumaQT为例，其主要流程如下：

（1）对模块中参数进行初始化。

（2）为了计算次数，HEVC中默认使用了帧内快速搜索算法，模块整体分粗选和细选两大阶段。之所以使用快速搜索，是为了避免对所有帧内预测模式都进行率失真优化。在粗选阶段，遍历35种帧内预测模式，调用预测值计算函数predIntraAng，再调用哈达玛失真函数xGetHADs，然后更新候选列表，构建率失真优化的候选列表。全率失真优化候选列表的长度由块宽度决定。在细选阶段中，首先构建MPM列表，加入全率失真优化候选列表中。遍历全率失真优化候选列表，然后调用变换量化重构函数，此时函数中会按照四叉树向下划分，计算率失真代价，直到找到最佳的模式。

（3）对获得的最优模式再次调用变换量化重构函数，此时在分别划分四叉树和不划分的条件下进行计算，最终选出最佳模式为最终的亮度帧内预测模式。

（4）收尾工作，记录信息。

### 哈达玛失真函数并行优化设计与实现

HM中在帧内预测粗选阶段，通过计算哈达玛失真（HAD）来进行初步筛选，这种失真的计算复杂度比率失真要小，因此能够提高初选的速度。其内部分别实现了对4×4矩阵和8×8矩阵的哈达玛（Hadamard）变换以及HAD值的计算。此处以4×4矩阵为例，说明并行优化算法的设计与实现。

算法的输入参数为一维原始图像数组piOrg与一维预测图像数组piCur。这两个图像数组的大小均为16，即包含16个像素值。其中，为了增加变量的复用性，减少运算次数，在哈达玛变换中使用了残差矩阵diff和中间变量矩阵m。首先进行循环，计算出每个像素的残差，即diff[k] = piOrg[k]-piCur[k]。接下来反复对diff和m矩阵进行一系列计算，具体过程如(3-1)、(3-2)所示，其中为方便起见，diff矩阵记为d：

(3‑1)

(3‑2)

最后累加diff矩阵中元素的绝对值abs(diff[i])，得到失真值。

从这一过程中可以发现，diff矩阵与m矩阵的计算的交替进行的，每一次矩阵计算需要依赖于前一次的计算结果，因此不能为每一个thread分配形式相同的计算任务，不便于直接转化为CUDA并行函数。为了能适合CUDA并行程序，需要解除中间矩阵diff与m的数据相关性。此处有两种可行方式，第一种方法（下文称为“代入展开法”）为：直接将之前的中间计算过程代入到最后依次diff矩阵，将其展开，这种方法可以将原本的4次矩阵运算变为一次，且可以省去中间变量m，从而消除数据依赖。经过这种展开之后diff矩阵。具体过程如图3‑7所示：



图3‑7 矩阵元素计算过程展开

经过展开，d[0]=d[0]+d[1]+…+d[15]，d[1]、d[2]依次同理可得，因而对CUDA核函数，只需在一个block中开辟4×4个thread，分别计算一个d[i]的值即可，如图3‑8所示：



图3‑8 代入展开法线程划分

第二种方法（下文称为“依赖整合法”）为：观察到每一步变换中，矩阵中的任意一项元素都依赖于前一矩阵的两项元素。因此可以分析具体的数据相关性，然后将具有依赖关系的元素放在同一CUDA线程中计算。数据依赖关系以及对应的线程规划如图3‑9所示：



图3‑9 依赖整合法线程划分

这两种方法的理论并行度、计算复杂度和实际性能有所不同，具体的测试和分析将在第四章详细介绍。

### 模式预测值计算函数并行优化设计与实现

如前文所述，HM中帧内预测值计算函数主要为：首先判断预测模式为平面、直流还是角度预测模式，其中平面模式和直流模式各如前文所述进行处理，不再赘述。对于角度模式，以预测模式为21，预测块尺寸为4×4为例分析具体过程。首先生成预测参考像组集，上方参考像素为p[x][-1],x=-1,…,8，左侧参考像素为p[-1][y],y=1,…,8，记一维参考像素集为ref，其中ref[x]中x=-4,…,8。对于x大于等于0，ref取p[-1+x][-1],x=0,…,8；对于x小于0，ref取p[-1][-1+Round(32×(-1)/(-17))]=p[7][-1]。因此整个参考像素集的产生即如图3‑10所示：



图3‑10 参考像素集选择

接下来继续得到模式索引和权重参数并计算预测值，具体算法前文已说明。将预测值计算函数改写为CUDA核函数的主要目的是为了便于帧内预测模块的整体优化，其内部的优化并没有统一方法，主要为对规模较大的for循环进行改写，此处给出一处为例。串行代码为：

*……*

*for (Int y=0, deltaPos=intraPredAngle; y<height; y++, deltaPos+=intraPredAngle, pDsty+=dstStride)*

*{*

*const Int deltaInt = deltaPos >> 5;*

*const Int deltaFract = deltaPos & (32 - 1);*

*// Do linear filtering*

*const Pel \*pRM=refMain+deltaInt+1;*

*Int lastRefMainPel=\*pRM++;*

*for (Int x=0;x<width;pRM++,x++)*

*{*

*Int thisRefMainPel=\*pRM;*

*pDsty[x+0] = (Pel) ( ((32-deltaFract)\*lastRefMainPel*

*+ deltaFract\*thisRefMainPel +16) >> 5 );*

*lastRefMainPel=thisRefMainPel;*

*}*

*}*

*……*

其中y的循环控制变量height和x的循环控制变量width最大都为64，因此这是一个规模较大的函数，这一过程也可以较容易地改写为并行函数。即配置64个线程块，其中每个线程块包含64个线程，每一个线程块对应1次外层循环，其中地每一个线程对应1次内层循环的计算。代码实现为：

*int bid = blockIdx.x;*

*int tid = threadIdx.x;*

*for (Int y=0, deltaPos=intraPredAngle; y<height; y++, deltaPos+=intraPredAngle, pDsty+=dstStride)*

*{*

*if (y == bid)*

*{*

*……*

*for (Int x=0;x<width;pRM++,x++)*

*{*

*if (x == tid)*

*……*

*}*

*}*

*}*

*…..*

其余部分类似的循环也可以按照相似方法进行改写。

### 帧内预测模块整体并行优化设计与实现

除了对具体功能函数，即哈达玛失真函数和预测值计算函数进行优化之外，整个帧内预测的入口函数也有可以并行优化的空间。根据HM代码，其中候选集粗选的过程，是对于35中预测模式进行遍历进行的，对其中每一种模式都进行“预测值计算-哈达玛失真计算”的操作。根据前文工作，这两个主要的功能函数已经被优化为了CUDA核函数，因此可以考虑对全部35种预测模式进行并行优化。具体做法为：64×64的预测图像最大被划分为64个8×8的预测块，因此在CUDA中至多开辟64×35个block，其中每个block分别并行处理一个PU的一种预测模式，每个block内部再开辟最多8×8个thread，每个thread计算变换矩阵一个元素值，最后再串行累加计算出该模式的失真值。优化后整个“预测值计算-哈达玛失真计算”过程如图3‑11所示：



图3‑11 帧内预测整体优化线程划分

### 帧内预测优化算法理论并行度分析

在实际的CUDA并行优化中，由于涉及到GPU初始化、内存的申请与数据拷贝、CPU与GPU数据通信等等问题，因此实际的优化效果与预期存在有较大差距。为了对优化效果有更好地评估和分析，通过计算并行算法的最大理论并行度来作为实际优化结果的参考值。

对于哈达玛失真函数，为前文提出的两种方法分别分析并行度。当矩阵大小为4×4，“代入展开方法”开辟16个线程，理论并行度为16， “依赖整合方法”开辟8个线程，理论并行度为8；当矩阵大小为8×8，同理，两种方法理论并行度分别为64和32。

对于预测值计算函数，取代的最大循环为64×64，因此理论最大并行度为4096。

对于模块整体优化，串行算法在粗选阶段遍历35种预测模式，再对最大为64×64大小的块分割处理，计算每个8×8小块的预测值和哈达玛失真，而在并行优化后，至多同时生成35×64个线程块（blocks），且其中每个线程块又分别产生最大8×8个线程。因此整体的最大理论并行度为143360。可见整体的理论优化效果是较好的。本模块所有优化后的理论并行度汇总如表3‑2所示，其中PIA为帧内角度预测predIntraAng的缩写，此外，虽然前文对哈达玛失真计算给出了两种并行思路，但为简洁起见，表中仅列出了“带入展开法”的理论并行度，“依赖整合法”的并行度为其1/2：

表3‑2 帧内预测模块优化理论并行度表

| 函数名称 | 功能描述 | 线程块 | 线程数 | 理论并行度 |
| --- | --- | --- | --- | --- |
| CUDA\_HAD4 | CUDA 4×4哈达玛失真 | 1 | 16 | 16 |
| CUDA\_HAD8 | CUDA 8×8哈达玛失真 | 1 | 64 | 64 |
| CUDA\_PIA | CUDA 预测值计算 | 1 | （64，64） | 4096 |
| CUDA\_Intra | CUDA 模块整体优化 | 35 | 1 | 35 |
| CUDA\_Intra  +CUDA\_HAD4 | CUDA 模块整体优化  +4×4哈达玛失真 | （35，64） | 16 | 35840 |
| CUDA\_Intra  +CUDA\_HAD8 | CUDA 模块整体优化  +8×8哈达玛失真 | （35，64） | 64 | 143360 |
| CUDA\_Intra  +CUDA\_PIA | CUDA 模块整体优化  +预测值计算 | （35） | （64，64） | 143360 |

## 变换量化模块优化

### 变换量化模块分析

由于HM中的量化函数涉及较多的条件判断，计算复杂度较高，因此在实际优化中仅针对一系列变换函数进行优化，但是在HM中变换与量化的过程在很多入口函数都较为模糊，因此仍然对整个变换量化模块进行分析。入口函数transformNxN的逻辑较为简洁，流程如下：

1. 对残差值进行处理PCM处理，利用周围临近参茶之预测当前残差。
2. 检验是否为trans-quant-bypass模式，若为该模式，则直接使用前一步预测出的残差值。
3. 检验是否为transform-skip模式，若为该模式，则调用跳过模式下的变换函数xTransformSkip。若为否则进行下一步。
4. 调用变换计算的入口函数xT，进行普通变换。
5. 调用量化计算的入口函数xQuant，进行量化。

### 快速DST变换函数并行优化设计与实现

根据前文简述的二维离散正弦变换矩阵，其元素值为正弦函数值，存在无理数，因此在HM中，为了便于存储和计算，使用存放整数矩阵和系数的方式来表示变换矩阵。需要指出的是，为了性能提升，在HM中并没有实现完整的DST变换，而是使用了一种近似的“快速DST变换算法”。在HM中，其实现方法如下：

*// input block, output coeff*

*Void fastForwardDst(TCoeff \*block, TCoeff \*coeff, Int shift)*

*{*

*Int i;*

*TCoeff c[4];*

*TCoeff rnd\_factor = (shift > 0) ? (1<<(shift-1)) : 0;*

*for (i=0; i<4; i++)*

*{*

*// Intermediate Variables*

*c[0] = block[4\*i+0];*

*c[1] = block[4\*i+1];*

*c[2] = block[4\*i+2];*

*c[3] = block[4\*i+3];*

*for (Int row = 0; row < 4; row++)*

*{*

*TCoeff result = 0;*

*for (Int column = 0; column < 4; column++)*

*{*

*// use the defined matrix, rather than hard-wired numbers*

*result += c[column] \* g\_as\_DST\_MAT\_4[TRANSFORM\_FORWARD][row][column];*

*}*

*coeff[(row \* 4) + i] = rightShift((result + rnd\_factor), shift);*

*}*

*}*

*}*

经过分析，不难发现其中循环部分本质是对每一个矩阵元素进行的变换计算，因此并行算法设计中，中只需要将每一个元素的计算过程从循环展开为一个表达式，然后对每一个元素分别开辟一个线程进行计算即可。如线程0计算coeff[0]：元素，线程1计算coeff[1]，以此类推。其中以coeff[0]为例，将其循环计算过程展开之后，coeff[0]=29×block[0]+55×block[4]+74×block[8]+84×[12]，其中block数组元素前的系数为DST变换矩阵对应的变换系数，其余的元素值也可以类似推导得到。

### 蝶形DCT变换函数并行优化设计与实现

在HM中，为了简化矩阵乘法运算，实现了一种蝶形DCT变换partialButterfly，充分利用了矩阵运算过程中间变量的重复性，大大降低了矩阵乘法计算的复杂度。针对不同大小的矩阵，HM中分别实现了4×4、8×8、16×16和32×32矩阵的。为了便于说明蝶形算法实现方法，此处以8×8矩阵的蝶形DCT变换为例，使用常规的矩阵乘法运算需要执行512次乘法运算和448次加法运算，而蝶形变换仅需要64次乘法运算和160次加法运算[31]。HM中该算法实现思路如图3‑12所示：



图3‑12 8×8蝶形DCT变换

针对这一算法，从图中可以看出，上下两半部分存在不同的分支情况，因而也就存在不同的计算复杂度的分支。因此在并行优化时需要考虑的主要问题为线程的设置和计算任务的划分。考虑运算过程的最后一步的蝶形加运算，下半部分的每个结果值为4个中间变量的求和。因此在数据依赖关系上，最后的每一个运算结果依赖于上一步的4个中间结果，而进一步向上分析，又依赖于2个原始矩阵中的元素值。为了解决数据相关性问题，应将公共数据部分写入通过\_\_shared\_\_关键字申请的CUDA共享内存，然后为每一线程分配计算量相当的计算任务，当需要使用公用变量时，可以直接从线程块共享的共享内存中读取，从而同时兼顾提高并行度和降低线程闲置与等待时间。

通过分析，上半部分（编号为0、1、2、3的原始块）和下半部分（编号为4、5、6、7的原始块）的分支情况，发现计算过程中上半部分的每个块进行3次加法和1次乘法计算，下半部分中每个块进行4次加法和1次乘法运算，上下半区的计算量存在不同，如果统一分配线程可能导致最后同步线程时发生等待和闲置。为避免线程闲置等待，可以将将整个运算过程分拆为3个CUDA核函数执行，保证每一线程计算量相等。因而此处也有两种并行方法。第一种方法（下文称为“单一核函数法”）计算过程的线程分配如图3‑13所示：



图3‑13 单一核函数法线程划分

根据上图，编号为0~3的线程计算中共执行3次加法运算和1次乘法运算，编号4~7的线程计算中共执行4次加法运算和1次乘法运算。在第一次出现数据相关的整体蝶形加减法计算，每个线程执行的计算过程一致，都是1次加法，因而基本不存在线程等待。在第二次出现数据相关的上半部分的蝶形加减法计算，上半部分的每一线程都执行了2次加法，因而运行时间也基本一致。在第三次出现数据相关的下半部分的蝶形加法计算，下半部分每一个线程执行了1次乘法1次加法，因此线程间基本不存在等待。同时上半部分线程执行了2次加法一次乘法，而最后一次蝶形加法中，上半部分每个线程执行1次加法，下半部分执行3次加法，各线程计算任务基本一致。

第二种方法，也即是将整个变换过程拆分为多个CUDA核函数依次运行（下文称为“多阶段核函数法”）。具体为：核函数1开辟8个线程，分别完成第一步的蝶形变换加减法，核函数2开辟4个线程，每个线程分别完成对上半部分中间变量的计算，核函数3开辟8线程，完成对每个元素值和对应变换系数的乘法计算，核函数4完成对8个结果的计算。具体流程和线程配置情况如图3‑14所示：



图3‑14 多阶段核函数法线程划分

### 变换量化优化算法理论并行度分析

对于量化变换模块，主要针对快速离散正弦函数fastForwardDST和4种不同尺寸的蝶形离散余弦变换函数partialButterfly进行了并行优化。

对于离散正弦变换函数，将其中每一个矩阵元素值的计算过程直接展开，矩阵中共有16个元素，理论并行度为16；对于4×4大小的蝶形离散余弦变换函数partialButterfly4（一下简写为PBF），由于蝶形计算只进行一次，所以不需分两种情况讨论，直接展开后理论并行度为16，此处为例函数命名的一致性，仍将其命名为CUDA\_PBF4\_SK。针对8×8大小的蝶形变换函数PBF8，单一核函数法的理论并行度为64，多阶段核函数的理论并行度在各个阶段不同，最高值也为64。对于16×16的蝶形变换函数PBF16，为了简化核函数内部逻辑，仅对蝶形变换的第一步、乘法计算以及之后的部分蝶形计算过程进行了并行，单一核函数法配置为8线程块，每一线程块16线程，理论并行度为128，多阶段核函数法至多配置16个线程块，每一线程块包含16个线程，理论并行度为256。对于32×32的蝶形变换函数PBF32，与PBF16情况类似，单一和函数法理论并行度为512，多阶段核函数理论并行度最大为1024。汇总如表3‑3所示，表中\_SK为单核函数single-kernal缩写，MS为多阶段multi-stages缩写：

表3‑3 变换量化模块优化理论并行度表

| 函数名称 | 功能描述 | 线程块 | 线程数 | 理论并行度 |
| --- | --- | --- | --- | --- |
| CUDA\_FFDST | CUDA快速离散正弦变换 | 1 | 16 | 16 |
| CUDA\_PBF4\_SK | CUDA 4×4蝶形变换 | 1 | 8 | 8 |
| CUDA\_PBF8\_SK | CUDA 8×8单核函数蝶形变换 | 1 | 64 | 64 |
| CUDA\_PBF8\_MS | CUDA 8×8多核函数蝶形变换 | 1 | 64 | 64 |
| CUDA\_PBF16\_SK | CUDA 16×16单核函数蝶形变换 | 8 | 16 | 128 |
| CUDA\_PBF16\_MS | CUDA 16×16多核函数蝶形变换 | 16 | 16 | 256 |
| CUDA\_PBF32\_SK | CUDA 32×32单核函数蝶形变换 | 16 | 32 | 512 |
| CUDA\_PBF32\_MS | CUDA 32×32多核函数蝶形变换 | 32 | 32 | 1024 |

## 本章小结

本章首先分析了HM中主要模块的运行情况，又进一步对其中的帧内预测模块和变换量化模块的各个功能函，做了运行时间占比的测试和分析。接下来对帧内预测模块中的哈达玛失真函数、预测值计算函数和模块整体，变换量化模块中的快速DST变化的蝶形DCT变换函数进行了并行算法的设，并介绍了实现方法。最后对各个功能函数以及同一函数的不同实现思路，分别计算和比较了器理论最大并行度，作为之后实验测试结果的参考。

# 实验结果分析

在完成并行优化方法的设计于实现之后，需要对实际的优化效果进行实验和分析。其中实验内容主要包括算法选择和算法性能。算法选择是指，针对同一个函数，对前文提出的不同实现方式分别进行测试，选择出其中较好的方式作为最终方法。算法性能部分的实验即是将并行优化后的算法与HM中的串行算法进行对比。分析部分主要结合并行算法最大理论并行度和实际优化效果，来分析并行算法的加速效果。本章对帧内预测模块、变换量化模块，分别进行了上述的实验和分析。

## 哈达玛失真函数并行优化实验结果与分析

首先分析帧内预测模块种对于哈达玛失真函数的并行优化结果。使用与前文测试中相同的测试环境和视频测试序列。其中，由于前文提出了2种并行优化方法，因此实验中对两种实现都进行了测试。结果如图4‑1所示：

图4‑1 哈达玛失真函数优化结果

图中HADs函数表示HM中实现的串行哈达玛失真函数，CUDA\_HADs\_Z表示用“依赖整合法”实现的并行函数，CUDA\_HADs\_D表示用“代入展开法”实现的并行函数。通过实验可以发现，两种方法相对于串行函数都有一定的加速效果，且对于不同测试序列优化性能基本稳定。结合对于理论并行度的计算，这两个函数的加速性能如表4‑1所示：

表4‑1 哈达玛失真优化结果

|  |  |  |
| --- | --- | --- |
| 函数名 | 最大并行度 | 平均加速比 |
| CUDA\_HADs\_Z | 32 | 1.158 |
| CUDA\_HADs\_D | 64 | 1.2655 |

可见使用“带入展开法”实现的并行函数有更好的平均加速比，相比于穿行算法能速度能提升26%左右。而这一结果也和并行度分析一致，因此在之后的帧内预测模块整体优化和HM整体测试中，选择使用该方法来进行哈达玛失真的并行计算。

## 预测值函数并行优化实验结果与分析

接下来分析帧内预测模块中对于预测值计算函数的并行优化结果。结果如图4‑2所示：

图4‑2 预测值计算函数优化结果

图中predIntraAng表示HM中的串行预测值计算函数，CUDA\_PIA表示并行优化后的预测值计算函数。如图所示，该部分优化对于不同测试序列也保持性能稳定，其平均加速比约为11%。

## 帧内预测模块整体并行优化实验结果与分析

对于帧内预测模块的整体优化包含了对于以上2类函数的多线程并行调用，因此在结果统计的过程中，一方面记录了并行优化前后模块的整体的运行时间，另一方面也记录了哈达玛失真函数和预测值计算函数在高并发调用的情况下的优化情况，以便更好地分析优化后的帧内预测模块中具体函数的性能提升。模块整体的运行时间与其中优化函数的运行时间实验数据如图4‑4所示：

图4‑4 帧内预测模块优化结果

根据这一结果，可以发现，经过帧内模块的整体优化之后，哈达玛失真函数和预测值计算函数的加速性能相比于之前有巨大提升。这一性能的提升主要收益于更高的并行度。如前文所述，在帧内预测模块的整体优化中，哈达玛失真函数的并行度最大提高64×35倍，预测值计算函数的并行度最大提高35倍。在模块并行的条件下，这两个函数的平均加速比和理论最大并行度如表4‑2所示：

表4‑2 帧内预测模块优化结果

|  |  |  |
| --- | --- | --- |
| 函数名 | 最大并行度 | 平均加速比 |
| CUDA\_HADs | 64 | 1.2655 |
| CUDA\_HADs\_MA | 143360 | 9.1975 |
| CUDA\_PIA | 4096 | 1.1095 |
| CUDA\_PIA\_MA | 143360 | 1.7836 |
| CUDA\_INTRA\_MA | 143360 | 1.2207 |

表中\_MA后缀表示“模块加速”（module acceleration），即在帧内预测模块整体优化情况下的对应函数，最后一行的CUDA\_INTRA\_MA表示整个帧内预测模块。其中哈达玛失真函数的加速比达到了9.2左右，而预测值计算函数的加速比达到了1.8，相比于之前单一函数的性能提升更为显著。因此整个帧内预测模块的加速比也达到了1.22以上。根据前文对函数运行时间和所占比重的分析，哈达玛失真函数占帧内预测模块运行时间的约13%，预测值计算函数占约12%，可知对模块优化的最大理论加速比为1/（1-25%）≈1.33。对比实际实现的并行方法，已经达到了理论最大加速比的91%，基本符合预期。

## 快速DST函数并行优化实验结果与分析

下面对变换量化模块的优化结果进行实验和分析。首先分析快速DST变换的优化结果，实验结果如图4‑5所示：

图4‑5 HM模块耗时百分比

图中FDST表示HM中实现的串行快速DST变换函数，CUDA\_FDST表示并行优化之后的快速DST函数。可以发现，该算法的优化性能对于不同的测试序列也保持了基本稳定。根据前文分析，该算法的最大理论并行度为16，实际算法的加速比为1.36。

## 蝶形DCT变换函数并行优化实验结果与分析

对于蝶形DCT变换，也进行了两种方法的并行优化，即“单一核函数法”和“多阶段核函数法”，首先对这两种方法进行实验和分析，选择其中较好的的并行实现方法作为最终的优化函数。以32×32大小的蝶形DCT变换函数为例，这两种方法的运行时间分别如图4‑6所示：

图4‑6 两种蝶形DCT优化比较

其中PBF32（Partial Butterfly 32）为HM中的串行蝶形变换函数，CUDA\_PBF32\_SK为单一核函数法优化的并行函数，CUDA\_PBF32\_MS为多阶段核函数法优化的并行函数。从图中可以看出，这两种方法对于原函数都有较好的加速效果，且加速效果对于不同的测试序列基本保证稳定。2种方法的具体加速比和理论最大并行度如表4‑3所示：

表4‑3 2种蝶形DCT并行方法优化结果

|  |  |  |
| --- | --- | --- |
| 函数名 | 最大并行度 | 平均加速比 |
| CUDA\_PBF32\_SK | 512 | 8.1153 |
| CUDA\_PBF32\_MS | 1024 | 5.505 |

从表中可以发现单核函数法的实际优化效果更好，而其理论最大并行度却比多阶段核函数法要低。这是主要是因为在多阶段和函数法种，调用了多次核函数，虽然最大线程数更多，但是单线程的计算方法过于简单，程序在线程同步和设备通信上的耗时反而较高。而单一核函数法只需要调用一个核函数，且每一个线程具有一定的计算深度，从而充分利用了线程计算能力。经过这一实验，选择单一核函数方法来作为最终的蝶形DCT变换并行优化方法。

选择较佳的方法之后，对于不同矩阵大小的蝶形DCT变换函数分别实现其并行函数并进行试验，具体结果如图4‑7所示：

图4‑7 蝶形DCT优化结果

从图中可以发现，单一核函数方法对于不同尺寸的蝶形DCT变换函数都有比较明显的优化效果，并且随着输入矩阵的尺寸从4到32，并行函数的加速加速比也逐渐增加。具体每种尺寸的变换函数优化之后的实际加速比和理论最大并行度如表4‑4所示：

表4‑4 哈达玛失真优化结果

|  |  |  |
| --- | --- | --- |
| 函数名 | 最大并行度 | 平均加速比 |
| CUDA\_PBF32\_SK | 512 | 8.1153 |
| CUDA\_PBF16\_SK | 128 | 1.7448 |
| CUDA\_PBF8\_SK | 64 | 1.4106 |
| CUDA\_PBF4\_SK | 8 | 1.2092 |

从中可以发现，对于尺寸最大的32×32矩阵的蝶形DCT变换，其加速比达到了8.1以上，而对于较小的尺寸则加速效果相对较小，这一结果与理论并行度也是一致的。

完成DCT变换相关函数的优化和实验之后，可以对整个变换量化模块的优化结果进行分析。根据前文所述，变换量化函数中的快速DST变换和蝶形DCT变换总计占到了整个模块运行时间的50%以上，因此文中涉及的优化部分，理论最大加速比为1/（1-50%）=2。而实际实现的优化加速比为1.25，达到了最大加速比的62%，基本符合预期。

## HM整体并行优化实验结果与分析

完成对于帧内预测模块和变化量化模块的实验与分析之后，可以看出，并行优化的实现对于这两个模块分别取得了符合预期的效果，因而下面接着测试HM整体在经过并行优化之后的运行情况。在测试时，通过特性宏CUDA\_ACC来控制工程中各处的CUDA函数启用，其中经过前文实验的选择，哈达玛失真函数使用“依赖展开法”实现，蝶形DCT变换使用“单核函数法”实现，且两个模块的优化同时开启。具体运行情况如图4‑8所示：

图4‑8 HM整体优化结果

从中可以看出，在不同测试序列下，并行优化对于HM整体的加速效果是比较稳定的，平均加速比达到了1.19。根据文中所优化部分，HM整体的理论最大加速比为1.33，实际加速比达到了理论加速比的89%，基本达到了并行优化设计的要求。

## 本章小结

本章对功能函数、模块和HM整体分别进行了实验与分析，并结合理论最大并行度与理论最大加速比，对实际优化结果进行了评价。对于哈达玛失真函数，最高达到了9倍以上的加速，对预测值计算函数达到了1.7倍以上加速，对DST变换函数达到了1.3倍以上加速，对蝶形DCT变换达到了8倍以上加速，对HM整体达到了1.19倍加速，实验结果基本达到优化设计需求。

# 总结与展望

新一代视频编码标注HEVC为了提高编码视频的压缩率，而引入了更为复杂的图像划分方式和编码流程，因此带来的计算量制约了这一标准的广泛推广。基于CUDA的并行优化方案充分利用了目前越来越普遍用的CPU+GPU多核异构平台，针对HM中的主要耗时模块分别进行了并行优化的设计与实现，并对优化结果进行了实验和分析。主要工作有以下几点：

1）设计并实现了HM编码器中帧内预测模块的CUDA并行优化。其中，对帧内预测模块中的哈达玛失真函数，分别设计和实现了使用带入展开方法和依赖整合方法的并行优化方案，并对两种方式分别进行了详细测试和分析。其中较好的带入展开方法加速比达到9.19。对于预测值计算函数，通过展开复杂度较大的多重循环，实现了其计算过程的并行优化，加速比达到1.7。对于帧内预测模块整体，通过同时运行35种预测模式实现了并行优化，模块整体加速比达到1.22。

2）设计并实现了HM编码器中变换量化模块的CUDA并行优化。其中，对快速DST变换函数，实现了使用循环展开的并行优化，加速比达到1.36。对蝶形DCT变换函数，分别设计和实现了单一核函数和多阶段核函数两种并行优化方案，并进行了实验对比分析。其中优化效果较好的单一核函数方法，加速比达到了8.11。

3）实现了CUDA函数静态链接库与HM整体工程的整合，并对整体优化流程进行了实验和分析。最终实际优化加速比为1.19，达到了理论最大加速比的89%。

4）根据各并行算法的实现与调用细节，分别计算并比较了各个优化函数的理论最大并行度。并在对于不同清晰度和帧率的视频测试序列，分别验证和测试了并行函数的实际优化结果。在这二者基础上，又进一步分析和评价了并行算法的优化结果与其设计预期之间的关联与差距。

虽然优化结果基本符合预期，但以上工作仍存在很多可以改进的地方。在变换量化模块中，占到整个模块50%时间的量化函数，没有对其进行并行优化；在帧内预测模块中相比于粗选阶段更加耗时的细选阶段，同样没有进行并行优化；在设计与实现了并行优化的模块和函数之中，GPU线程初始化、内存申请，GPU与CPU间通信仍占用了大量的时间，没有被很好地优化；而最终，对于HEVC编码器地整体优化虽然达到预期，但相比于H.264/AVC标准仍然过慢。

希望随着HEVC编码标准的修订升级，GPU性能与CUDA编程框架的日渐强大，以及相关领域研究的不断深入，HEVC标准编码器能最终成为兼具编码压缩率和编码速度的优秀编码器，在超高清、3D视频愈加流行的时代里发挥其应有的作用。

致 谢

在完成本次毕业设计之际，我想感谢郭红星老师半年以来对毕业设计工作的指导和敦促。对我的课题方向，郭老师有有独到而深入的见解；对我的学习习惯，郭老师有严格而细致的要求；对我的生活心态，郭老师有适时而温暖的关照。这一切的帮助，我才能较为顺利地完成毕业论文。

其次，我想感谢我的父母和家人，在这半年多的毕业设计和毕业论文完成过程中，我同时还承受了升学和工作方面的不小的压力，是父母在我心情糟糕时给我源源不断的支持，成为我学习、生活中最坚实的后盾，让我不论遇到什么困难和挫折，都能勇敢向前和坦然应对。

另外我还想感谢我的同学们。“关山难越，谁悲失路之人；萍水相逢，尽是他乡之客”，对于第一次面对较大体量和较高要求的论文课题的我，同样在迷茫中探索的同学们是最为可靠的同路人。漫长而曲折的毕业论文对于我们都是一次不小的历练，也正你们耐心和及时的帮助，让我克服了一个个难以解决的“坑”。你们的陪伴与同行，也让我有了不断前进的动力。

我还想感谢我的朋友和伙伴们，虽然你们中的很多人同样面临相似的，甚至更大的生活难题，但每当我向你们倾诉和抱怨，你们都会耐心开导和鼓励我，或者跟我一起娱乐减压，这些陪伴让我的生活没有被压力淹没，也让我每次都能在振作之后更加努力地投入毕设之中。

最后我想感谢我自己，虽然懒惰、散漫和脆弱一直是像幽灵一样缠绕着我的每一天，但是这难道不就是人之所以奋斗、刻苦和坚强的意义所在么？毕业论文是对一个课题的研究，也是对在华中科技大学4年学习的检验，更是对自己4年中成长历程的审视。感谢自己没有在漫长的旅程中掉队离去，也没有在最后的考验中败下阵来；感谢自己，没有在骄纵里迷失，也没有在挫败中沉沦；感谢自己，在现实里没有服从现状，在平凡中没有甘于平庸。

参考文献

1. 朱秀昌, 李欣, 陈杰. 新一代视频编码标准——HEVC[D]. , 2013.
2. 余兆明, 查日勇, 黄磊, 等. 图像编码标准 H. 264 技术[J]. 2006.
3. 刘昱, 胡晓爽, 段继忠. 新一代视频编码技术 HEVC 算法分析及比较[D]. , 2012.
4. 刘辉, 刘立程, 郝禄国, 等. 一种场景切换下的 HEVC 码率控制算法[J]. 电视技术, 2017, 41(6): 1-5.
5. 叶强. HEVC 视频编码算法的 CUDA 优化与应用[D]. 西安电子科技大学, 2015.
6. 王泽寰, 王鹏. GPU并行计算编程技术介绍[J]. 科研信息化技术与应用, 2013, 4(1): 81-87.
7. Nvidia C. Cuda[J]. 2013-05-01]. http://www. nvidia. com/object/cuda\_home\_new. html, 2008.
8. Kao H C, Wang I C, Lee C R, et al. Accelerating hevc motion estimation using gpu[C]//2016 IEEE Second International Conference on Multimedia Big Data (BigMM). IEEE, 2016: 255-258.
9. Wang B, Alvarez-Mesa M, Chi C C, et al. Efficient HEVC decoder for heterogeneous CPU with GPU systems[C]//2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP). IEEE, 2016: 1-6.
10. Takano F, Igarashi H, Moriyoshi T. 4K-UHD real-time HEVC encoder with GPU accelerated motion estimation[C]//2017 IEEE International Conference on Image Processing (ICIP). IEEE, 2017: 2731-2735.
11. Luo F, Wang S, Ma S, et al. Fast intra coding unit size decision for HEVC with GPU based keypoint detection[C]//2017 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2017: 1-4.
12. Ting Y C, Chang T S. Gradient-based PU size selection for HEVC intra prediction[C]//2014 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2014: 1929-1932.
13. Zhao L, Zhang L, Ma S, et al. Fast mode decision algorithm for intra prediction in HEVC//2011 Visual Communications and Image Processing (VCIP). IEEE, 2011: 1-4.
14. Yan S, Hong L, He W, et al. Group-based fast mode decision algorithm for intra prediction in HEVC[C]//2012 Eighth International Conference on Signal Image Technology and Internet Based Systems. IEEE, 2012: 225-229.
15. Sun H, Zhou D, Goto S. A low-complexity HEVC intra prediction algorithm based on level and mode filtering[C]//2012 IEEE International Conference on Multimedia and Expo. IEEE, 2012: 1085-1090.
16. Chi C C, Alvarez-Mesa M, Juurlink B, et al. Parallel scalability and efficiency of HEVC parallelization approaches[J]. IEEE Transactions on circuits and systems for video technology, 2012, 22(12): 1827-1838.
17. de Souza D F, Ilic A, Roma N, et al. Towards GPU HEVC intra decoding: Seizing fine-grain parallelism[C]//2015 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2015: 1-6.
18. Zhao Y, Song L, Wang X, et al. Efficient realization of parallel HEVC intra encoding[C]//2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW). IEEE, 2013: 1-6.
19. Jiang W, Long M, Jin H, et al. Fine-grained CUDA-based parallel intra prediction for H. 264/AVC[C]//Proceedings of Network and Operating System Support on Digital Audio and Video Workshop. ACM, 2014: 97.
20. 凌勇. 视频压缩编码中帧间参考关系的研究[D].浙江大学,2014.
21. 杨光旭. 基于HEVC的高效视频编码算法设计与实现[D].西安电子科技大学,2014.
22. 张和仙. 下一代视频编码标准 HEVC 帧间预测优化算法研究 [D]. , 2013.
23. 张维龙. HEVC 关键模块并行算法的设计与基于 GPU 的实现[D]. 大连理工大学, 2016.
24. 黄晗. HEVC 帧间/帧内预测及优化技术研究[D]. 北京: 北京交通大学, 2013.
25. 赵耀, 黄晗, 林春雨, 等. 新一代视频编码标准 HEVC 的关键技术[J]. 数据采集与处理, 2014, 29(1): 1-10.
26. 张真, 黄登山, 汤加跃. 视频编码中帧内预测算法研究及性能比较[D]. , 2007.
27. 胥加旺. 基于 HEVC 帧内预测算法的优化[D]. 南京理工大学, 2014.
28. 赵亮. HEVC 帧内模式决策及其核心技术的研究[D]. 哈尔滨: 哈尔滨工业大学, 2011.
29. 朱秀昌, 刘峰, 胡栋. 数字图像处理与图像通信[M]. 北京邮电大学出版社, 2002
30. 杨敏捷. 视频编码的帧内及帧间预测技术研究[D].电子科技大学,2018.
31. 苏斌, 刘畅, 潘志刚. 基于 FPGA 的高速浮点 FFT/IFFT 处理器设计与实现[J]. 中国科学院大学学报, 2015, 32(2): 259-263.