

电气工程学位项目，第二阶段，30 学分，斯德哥尔摩，瑞典 2020

面向下一代视频压缩的多分辨率编码与优化

VIGNESH V MENON

arXiv:2301.12191v1 [cs.MM] 28 Jan 2023

KTH皇家理工学院电气工程与计算机科学学院

面向下一代视频压缩的多分辨率编码与优化

VIGNESH V MENON

理学硕士 信息与网络工程 日期：2020年10月16日 导师：Jonatan Samuelsson 审查人：Markus Flierl 电气工程与计算机科学学院 主办公司：Divideon AB 瑞典语标题：面向下一代视频压缩的多分辨率编码与优化

摘要

多路编码是指以多个空间分辨率和多个比特率对相同内容进行编码。本研究评估了同一视频在 2160p、1080p 和 540p 分辨率下采用常规自适应比特率档位进行编码时的编码器分析相关性。我们建立并评估了一种多分辨率分层多自适应比特率编码方案，该方案显著提升了常规自适应比特率编码的计算效率。首先以较低分辨率和对应的中位比特率对视频内容进行编码，然后将运动矢量、CU块结构等编码器分析决策用于同一分辨率层级中的其他编码。随后对分析结果进行外推和优化，以用于更高分辨率的编码。我们使用 x265 HEVC视频编码器对该方案进行了验证。与传统的单实例编码方法相比，所提出的多分辨率分层多比特率编码方案整体最高可实现 2.5 倍的加速。此外，这种加速并未造成显著的编码效率损失。

CPU 中的 SIMD 向量单元已成为加速媒体处理及其他具有并行性的内核的事实标准。本研究还展示了面向硬件的优化对下一代视频编解码器编码速度的影响。研究使用 Arowana XVC 编码器进行了评估。

摘要

以不同分辨率和不同比特率压缩同一视频序列，称为“多路编码”。本文评估了同一视频序列在 2160p、1080p 和 540p 分辨率下的压缩情况，所用比特率适用于自适应视频流传输。本文对一种三级压缩方法进行了建模和评估，该方法可显著提升计算性能。视频序列首先以最低分辨率进行压缩，随后在相同及更高分辨率下进行压缩时，通过外推沿用运动矢量和块结构等编码决策。该模型使用 x265 HEVC 编码器进行了验证。与传统压缩方法相比，所提出的三级模型平均可实现 2.5 倍的整体加速，且不会显著降低编码效率。

现代处理器的 SIMD 指令已成为加速媒体处理及其他可利用并行性计算的事实标准。本文还展示了硬件优化对于加速下一代视频编解码器的效果。相关工作在视频编解码器 Arowana XVC 中完成。

目录

1 引言 **2**
1.1 视频压缩简介.2 1.2 单指令多数据简介.3 1.3 报告结构.4

2 HEVC自适应比特率流媒体 **6**
2.1 高效视频编码背景.6 2.2 x265 背景.8 2.3 自适应比特率流媒体背景.8 2.4 相关工作.11

3 XVC和英特尔单指令多数据背景 13
3.1 XVC视频编解码器.13 3.2 单指令多数据扩展.14 3.2.1 英特尔流式 SIMD 扩展 4.15 3.2.2 英特尔高级向量扩展 2.15 3.3 相关工作.16

4 x265中的分析共享方法 17
4.1 测试方法.17 4.2 x265中的帧内分辨率分析共享.19 4.2.1 最先进方法.19 4.2.2 所提出的方法.20 4.3 x265中的跨分辨率分析共享.21

5 I用于XVC编码的英特尔单指令多数据优化方法 **24**
5.1 SIMD 实现方法.24 5.2 NASM 汇编器支持.25 5.3 使用英特尔 SIMD 加速 Arowana XVC 编码 . . . . . 25 5.4 选择要加速的内核.25 5.4.1 代价函数.26 5.4.2 代价函数的 SIMD 实现.27 5.5 评估周期计数改进的方法.32

6 结果与讨论 33
6.1 x265 中最先进的帧内分辨率分析共享.33 6.2 x265 中提出的帧内分辨率分析共享.34 6.3 x265 中的跨分辨率分析共享.36 6.4 最先进的多编码方案.38 6.5 提出的多编码方案.38 6.6 SIMD 优化对 XVC 编码的改进.40

7 结论与未来方向 **42**
7.1 结论.42 7.2 未来方向.43

参考文献 **44**

A x265 命令行选项 **47**
A.1 x265 编码器分析共享模式预览.47 A.2 x265 编码器分析细化模式预览.48

B SSE4 单指令多数据优化 50

C 语言 AVX2 SIMD优化 58

图表目录

2.1 HEVC编码器框图 [10] .7 2.2 上下文感知编码 [5] .11 2.3 [16], [17] 中提出的多码率实现.12 2.4 [18] 中提出的多码率实现.12

3.1 使用英特尔® 酷睿™ i7-4500U 处理器 [28]，采用主配置文件进行 1080p 编码时，英特尔® 高级向量扩展 2 对 x264 和 x265 的性能提升.16

4.1 对具有最低QP作为主表示的CQP表示进行 WxH 分辨率编码时的帧内分辨率分析共享.19 4.2 对具有中位数QP作为主表示的CQP表示进行 WxH 分辨率编码时的帧内分辨率分析共享.21 4.3 1080p视频帧内分辨率分析共享中所有表示的平均编码时间.22 4.4 CQP表示的帧间分辨率分析共享.23 5.1 英特尔®流式SIMD扩展4内核相对于对应C++ 内核的周期计数增益.26 5.2 英特尔® 高级矢量扩展 2内核相对于对应C++ 内核的周期计数增益.26 6.1 采用当前最先进方法（B_1 > B_2 > B_3 > B_4 > B_5）的ABR表示进行 WxH 分辨率编码时的帧内分辨率分析共享 33 6.2 采用所提出的方法（B_1 > B_2 > B_3 > B_4 > B_5）的ABR表示进行 WxH 分辨率编码时的帧内分辨率分析共享 . . . 34 6.3 当前最先进方法与所提出方法的cVBR独立编码BD-Rate比较.35 6.4 当前最先进方法与所提出方法的cVBR独立编码BD-PSNR比较.35 6.5 ABR表示（B_1 > B_2 > B_3 > B_4 > B_5）的帧间分辨率分析共享.36 6.6 采用分析细化与不采用细化时，帧间分辨率cVBR独立编码的BD-Rate比较.37

6.7 采用分析细化与不采用细化时，帧间分辨率cVBR独立编码的BD-PSNR比较.38 6.8 当前最先进编码器在不同分辨率和比特率之间的编码器分析共享表示.39 6.9 所提出编码器在不同分辨率和比特率之间的编码器分析共享表示.39 6.10 CrowdRun视频采用Arowana XVC预设时的XVC编码加速比（GOP长度=2秒）.40 6.11 BasketBallDrive视频采用Arowana XVC预设时的XVC编码加速比（GOP长度=2 秒）.41 6.12 DucksTakeOff视频采用Arowana XVC预设时的XVC编码加速比（GOP长度=2秒）.41

表格目录

2.1 Apple 在 Apple 技术说明 TN2224 [5] 中提出的固定比特率编码阶梯.9 2.2 Netflix 提出的按标题编码 [5] .10 3.1 通用处理器的 SIMD 扩展 [24] .15 4.1 用于验证所采用方法的视频.17 4.2 不同分辨率恒定量化参数（CQP）模式测试所考虑的 QP . . . . . 18 4.3 不同分辨率所考虑的 cVBR 码率.18 4.4 测试系统配置.19 4.5 以最低 QP（最高质量）表示作为主编码时，帧内分辨率分析共享的加速比和压缩效率测量结果 20 4.6 以中位数 QP 表示作为主编码时，帧内分辨率分析共享的加速比和压缩效率测量结果 . . . . . 21 4.7 以中位数质量表示作为主编码时，帧内分辨率分析共享的 BD-Rate 平均改进值.22 4.8 1080p 跨分辨率分析共享的加速比和 BD-Rate 测量结果：以 540p 表示作为主编码，分析复用级别 10.22 5.1 使用 SSE4 和 AVX2 优化时 SAD 内核的加速比.28 5.2 使用 SSE4 和 AVX2 优化时 SATD 内核的加速比.32 6.1 使用当前最先进的方法进行自适应比特率帧内分辨率分析共享时的加速比和 BD-Rate 测量结果.34 6.2 使用所提出的方法进行自适应比特率帧内分辨率分析共享时的加速比和 BD-Rate 测量结果.35 6.3 1080p 和 2160p 自适应比特率跨分辨率分析共享的加速比和 BD-Rate 测量结果：分别以 540p 中位比特率表示和 1080p 中位比特率表示作为主编码.36 6.4 1080p 和 2160p 自适应比特率跨分辨率分析共享（含细化）的加速比和 BD-Rate 测量结果：分别以 540p 中位比特率表示和 1080p 中位比特率表示作为主编码 . . 37

A.1 x265 分析复用级别及其相应说明 [29] .47 A.2 x265 [29] 中的帧内细化技术.48 A.3 x265 [29] 中的帧间细化技术.49 A.4 x265 [29] 中的运动矢量细化技术.49

缩略语列表

AAC 高级音频编码**ABR** 自适应比特率**AMP** 高级运动预测**AVC** 高级视频编码**AVX** 高级矢量扩展 **AVX2** 高级矢量扩展 2**AVX512** 高级矢量扩展 512**BD-PSNR** Bjøntegaard增量峰值信噪比 **BD-Rate** Bjøntegaard差值码率**CABAC** 上下文自适应二进制算术编码**CB** 编码块**CPU** 中央处理器**C QP** 恒定量化参数**CTB** 编码树块**CTU** 编码树单元 **CU** 编码单元**cVBR** 限幅可变比特率**DVD** 数字多功能光盘**FMA** 融合乘加**HDR** 高动态范围

HEVC 高效视频编码**HTTP** 超文本传输协议**IEC** 国际电工委员会**ISA** 指令集架构**ISO** 国际标准化组织**ITU-T** 国际电信联盟—电信标准化部门

化

联合协作团队视频编码 联合协作团队视频编码 **JVET** 联合视频探索团队**动态图像专家组** 动态图像专家组**运动矢量** 运动矢量**预测单元** 预测单元 **量化参数** 量化参数**率失真** 率失真**率失真优化** 率失真优化**残差四叉树** 残差四叉树**单指令多数据** 单指令多数据**流式SIMD扩展** 流式SIMD扩展**流式SIMD扩展2** 流式SIMD扩展2**SSE4** 流式SIMD扩展4**SSSE3** 补充流式SIMD扩展3**变换块** 变换块**超高清** 超高清**视频编码专家组** 视频编码专家组**视频点播** 视频点播

致谢

我深深感谢我的导师**Jonatan Samuelsson先生**（Divideon首席执行官）以及我的考官**Markus Flierl博士**（瑞典KTH皇家理工学院电气工程与计算机科学系），感谢他们在我的项目期间始终如一的支持、热忱的激励和持久的指导。

我还要感谢 Sundararaman Gopalan 博士、Gayathri N 女士、印度 Amrita School of Engineering Amritapuri Campus 电子与通信系，以及曾与我在 MulticoreWare Inc 共事、关系密切的好友 Kalyan Goswami、Praveen Tiwari、Santhoshini Sekar、Bhavna Hariharan 和 Kavitha Sam-path，感谢他们对我的信任，并帮助我不断提升学习能力。

最后，我要感谢我的父母、我的叔叔 Venugopalan P 博士、我的婶婶 Jyothi PM 女士，以及我的妹妹 Aishwarya，感谢他们给予我的爱、鼓励与支持。我也感谢我在斯德哥尔摩的三位挚友 Anubhab Ghosh、Prajit T Rajendran 和 Shreya K Chari 给予我的支持与鼓励。

第1章

引言

过去几年中，视频流媒体服务上的视频内容大幅增加。右图展示了按应用划分的全球互联网流量，其中 62% 为视频流媒体。根据 Sandwine 发布的 The Global Internet Phenomena Report，在 COVID-19疫情期间，2020年3月的视频播放量相比2020年2月底翻了一番 [1]。数亿订阅用户和交易型视频点播用户构成了互联网流量的一部分。可用的直播线性内容和超高清 [2] 点播内容数量正在显著增长。例如，大型制片公司目前已提供超过 250 部超高清高动态范围电影作品，值得注意的是，2018年国际足联世界杯也曾以超高清高动态范围进行现场直播[3]。这种情况要求采用有效的视频压缩方案，以降低全球网络负载。



全球互联网流量。来源：Sandvine [1]

1.1 视频压缩简介

视频编码是指将原始视频转换为与多种设备兼容的数字格式的过程。这样做是为了满足特定应用和硬件的兼容性与效率需求，例如 DVD/蓝光、移动设备、视频流媒体或一般的视频编辑。编码过程会根据所选编码标准的规范，将视频和音频数据转换为文件并进行压缩。

为了将视频压缩为更易管理的大小，内容分发商会使用一种称为 编解码器 的视频压缩技术。编解码器可以将体积庞大的视频进行紧密压缩，以便传输和存储。编解码器会对视频应用算法，并创建其近似副本。视频会被压缩以便存储和传输，之后再解压缩以供观看。流媒体同时采用音频和视频编解码器。H.264（也称为 AVC）是最常用的视频编解码器。AAC 是最常用的音频编解码器。

压缩后，流中的各个组成部分会被封装到包装格式或文件格式中。这些文件包含音频编解码器、视频编解码器、隐藏式字幕及相关元数据。常见的封装格式包括 .mp4、.mov、.ts 和 .wmv。封装格式通常可以容纳多种类型的编解码器。不过，并非所有播放平台都支持所有封装格式和编解码器。因此，在向多种设备进行流媒体传输时，多格式编码至关重要。

在当今的视频压缩领域，超高清和 HDR 内容需要采用 HEVC视频压缩标准 [4]。为缓解与 HEVC 相关的版税问题，其他编解码器也在不断开发，XVC 编解码器便是其中之一。由于为超高清视频内容开发的编解码器需要进行大量计算，才能达到所需的压缩效率，因此，始终可以通过优化编码流水线，在计算速度、压缩效率和视觉质量之间实现最佳权衡。

另一方面，过去几年中，自适应流媒体方式 [5] 几乎已被普遍采用，以便在非预配置网络（例如互联网）上向异构设备提供内容。从本项目的角度来看，自适应流媒体的关键特性是：向播放器提供同一视频内容的多个比特率和分辨率版本（“比特率阶梯”）。为10位HEVC内容编码比特率阶梯中的所有档位，需要大量计算资源；例如，高质量离线编码的速度低于每秒1帧（帧率）十分常见。第2.1节和第2.3节分别更详细地介绍了 HEVC 和自适应比特率流媒体，并对其进行了背景研究。

1.2 单指令多数据简介

多年来，计算机体系结构取得了巨大的进步。这一领域也对视频编码，尤其是SIMD指令的部署，产生了重大影响。Flynn 分类法中的并行计算机类别包括 SIMD。这里所描述的是能够同时对多个数据点执行相同操作的计算机。它们还具有多个处理单元。开发出仅使用通用处理器即可进行实时解码的软件，是引入 SIMD 扩展的原因 [6][7]。调节数字音频的音量以及调整数字图像的对比度，是 SIMD 扩展的常见应用。

SIMD指令也用于现代 CPU 设计，以提高多媒体应用的性能。提高视频编码效率和标准化是 SIMD能力的其他应用场景。这减少了消除样本依赖关系和定义计算精度所需的工作量。

近年来，指令集架构及其微架构日益多样化，复杂度也不断提高，例如单指令多数据指令集架构中加入了多种指令集。此外，即使采用相同的指令集架构，不同指令也具有各自的性能特点。如今的视频编解码器显著具备利用单指令多数据加速的能力，因此有必要进一步研究这些加速对视频编码器的影响。AVX512 是英特尔推出的新型单指令多数据扩展，是面向 x86 指令集架构、在此前 256 位向量扩展单指令多数据指令基础上扩展至 512 位的指令集。AVX512 可在 512 位向量中累加八个 64 位整数、八个双精度浮点数、十六个 32 位整数或十六个单精度浮点数。这大约是 SSE 处理能力的四倍，也是 AVX/AVX2 使用一条指令可计算的数据点数量的两倍。

本项目旨在重点研究单指令多数据对视频编解码器 XVC 的影响。本研究尝试估算编解码器复杂度与单指令多数据扩展所带来性能收益之间的依赖关系。本项目的研究与分析范围仅限于英特尔 AVX2 SIMD 和 Intel SSE4 指令，以及它们在提升编码器速度方面的潜在收益。

1.3 报告结构

第2章简要介绍 HEVC 自适应比特率流媒体的背景。第 2.1 节详细介绍高效视频编码的背景，包括其作为主流视频编解码器的发展历程以及一些关键技术细节。第 2.2 节简要介绍 x265，这是一款开源的高效视频编码编码器，本研究使用它来评估编码方法。第 2.3 节介绍自适应比特率流媒体的相关术语、发展历程和一些关键概念。第 2.4 节提及了本研究领域的一些前期工作，这些工作被用于本研究的文献综述。

第3章简要介绍 XVC视频编解码器和英特尔 SIMD 优化。第 3.1 节讨论 XVC视频编解码器的背景，第 3.2 节则介绍 SIMD 扩展的背景。第 3.2.1 节和第 3.2.2 节分别介绍 Intel SSE4 和 AVX2 SIMD扩展。第 3.3 节介绍了本项目参考的一些文献。

第4章简要介绍了本项目中用于 x265 多分辨率和多码率分析共享的方法。第4.1 节展示了如何测试本项目中考虑的各种方法。第4.2节解释了分辨率内

分析共享中所使用的方法，其中第4.2.1节详细介绍了最先进方法，第4.2.2节描述了所提出的方法。第4.3节展示了帧间分辨率分析共享中所使用方法的详细信息。

第5章介绍了为了解单指令多数据并行在媒体处理中的影响而开展的工作。第5.1节展示了单指令多数据实现的各种方法，并说明了选择手写汇编方法的原因。第5.2节介绍了本项目中用于编译手写汇编代码的 NASM 汇编器。第5.3节展示了使用 Intel SSE4 和 AVX2 SIMD 加速内核后 CPU 周期数的改进。第5.4节说明了如何选择待优化的函数；第5.4.1节和第5.4.2节则分别介绍了编码器代价函数的实现及由此带来的 CPU 周期数改进。第5.5节介绍了用于评估各个优化函数所带来的 CPU 周期数改进的方法。

第6章在实时流媒体场景中评估了第4节所述的方法。第6.1节展示了当前最先进水平的 cVBR 模式帧内分辨率分析共享方法，并讨论了相关结果。第6.2节讨论了所提出的 cVBR 模式帧内分辨率分析共享方法，以及相较于当前最先进水平的帧内分辨率分析共享方法所取得的改进。第6.3节介绍了 cVBR 中的跨分辨率分析共享方法、进行分析细化的必要性及相关结果。第6.4节和第6.5节分别讨论了当前最先进水平的多编码器结构和所提出的多编码器结构，包括编码器分析共享方案及相关结果。第6.6节讨论了利用 SIMD 扩展提升 Arowana XVC 编码速度的效果。

第7章总结了项目报告，并概述了项目期间的主要发现。第7.1节讨论结论，第7.2节则讨论本项目尚未探索的后续研究方向。

第2章

HEVC自适应比特率流媒体

本章简要介绍 HEVC 自适应比特率流媒体的背景。第2.1节详细介绍高效视频编码的背景，包括其作为领先视频编解码器的确立过程及一些关键技术细节。第2.2节简要介绍 x265，这是一款开源高效视频编码器，本研究使用它来评估各种编码方法。第2.3节介绍自适应比特率流媒体相关术语的发展历程和一些关键概念。第2.4节提及了本研究领域的一些先前研究，这些研究用于本研究的文献综述。

2.1 高效视频编码背景

联合协作团队视频编码（JCTVC） [4]最终将高效视频编码确立为广受欢迎的 AVC 标准的后继编码标准。该标准的制定历时两年半以上，第一个版本于 2013 [8]定稿。JCTVC 最初由致力于制定 HEVC 标准的国际电信联盟电信标准化部门视频编码专家组和国际标准化组织/国际电工委员会运动图像专家组合作成立。最终版 HEVC 规范获国际电信联盟电信标准化部门批准，成为 H.265 建议书的一部分，并获国际标准化组织批准，成为 MPEG-H 第2部分。目前使用的高效视频编码版本为第7版 [9]。高效视频编码的开发重点是高效编码高分辨率视频，从而在编码超高清视频（4K 或 2160p 分辨率）时显著提升编码增益[2]。高效视频编码标准通过纳入提高编码效率的工具来实现这一点。

HEVC 中的基本处理单元是通过将每个视频帧划分为方形编码树块（CTB）来创建的。CTB 有多种尺寸，例如 16x16、32x32 或 64x64。通过将 CTB 与相关语法元素组合起来，可以创建一种称为编码树单元（CTU）的更复杂结构：一个亮度分量和两个相应的色度分量。增大 CTU 的尺寸可以在高分辨率下获得更高的编码效率，但代价是计算复杂度增加。HEVC 使用分层四叉树划分结构，将 CTU 划分为一个或多个不同尺寸的编码单元（CU），尺寸范围为 8x8 至 64x64。此外，还可以沿着编码树边界将 CU 进一步细分为更小的块

，用于帧内（空间）预测和帧间（时间运动补偿）预测。如果候选块位于同一帧中，则称为帧内预测块；如果来自不同帧，则称为帧间预测块。帧内预测块可以表示为预测块与表示预测角度的模式的组合。帧内预测所使用的模式包括 DC、平面模式和角度模式，分别表示相对于预测块的不同角度。另一方面，帧间预测块可以表示为参考块（用于预测的块）和运动矢量（MV）的组合，其中 MV 表示参考块与当前块之间的差异。运动矢量净值为零的块使用合并模式，其他块使用非对称运动分割模式（AMP）。在没有残差的特殊情况下，即预测块与源块完全相同，我们使用跳过模式——这是合并模式的一种特殊情况。为了提供预测数据，每个编码单元（CU）至少定义一个预测单元（PU）。所选预测模式表示该 CU（由编码块、单个亮度分量和两个色度分量组成）采用帧内预测模式还是帧间预测模式进行编码。通过将尺寸范围为 4x4 至 32x32 的编码块（CB）进行划分，可以形成变换块（TB），以对预测残差执行变换编码。因此，每个 CTB 都充当编码树的根节点，而编码块充当编码树的叶节点。编码块是变换树或残差四叉树（RQT）的根节点。

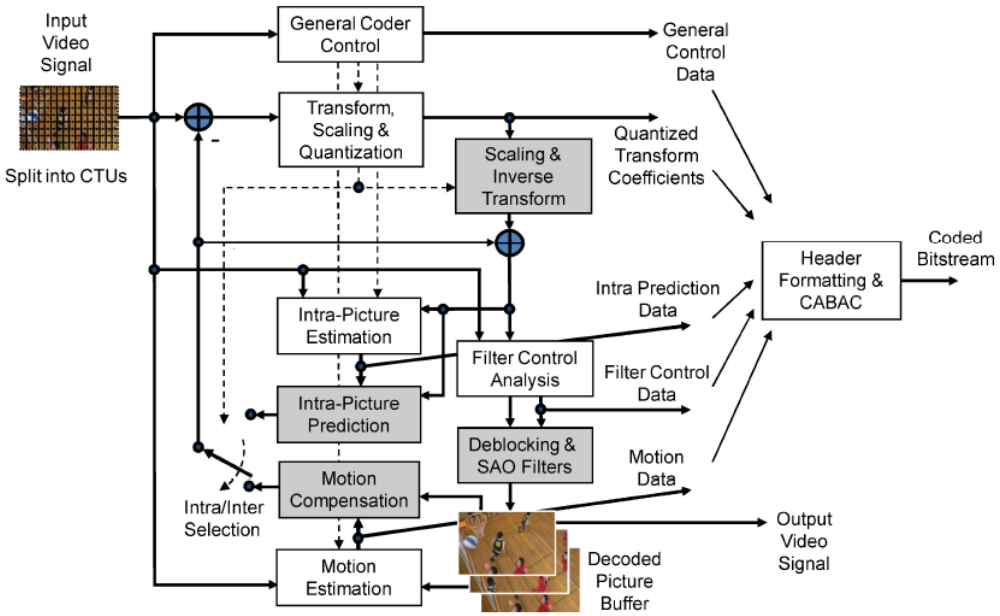

图 2.1：HEVC编码器框图 [10]

在帧内或帧间预测中会使用率失真优化 [11]，包括四叉树和 RQT 划分。率失真优化旨在确保在预设目标比特率下将失真降至最低，或在预设失真对应的目标质量级别下将比特率降至最低。在高分辨率视频中，高效编码需要进行大量块分割决策，因此需要执行大量率失真优化计算。这意味着将消耗更多的

计算资源。因此，降低编码的整体计算复杂度对于以经济的方式为广大观众提供高质量的超高清视觉体验至关重要。

HEVC 标准于 2013 年最终确定后，各种将视频编码为 H.265/MPEG-HEVC 压缩格式的开源实现相继出现，x265 便是其中之一。第 2.2 节将介绍 x265 的背景。

2.2 x265 背景

x265 是一种开源 HEVC编码器，可根据 HEVC 标准对原始视频进行压缩 [4]。该编码器已集成到多个开源框架中，包括 VLC、HandBrake [12], 和 FFMpeg [13] , 并且是事实上的 HEVC 开源视频编码器。x265 编码器针对多个平台提供了汇编优化，包括英特尔架构、ARM 和 PowerPC。x265 编码器采用帧间并行和帧内并行技术，以应对 HEVC编码日益增加的复杂性。对于帧间并行，x265 利用系统级软件线程并行编码多个帧。对于帧内并行，x265 依赖 HEVC 标准提供的波前并行处理（WPP）工具。该功能支持并行编码给定帧中的编码树单元行，同时确保前一行中用于帧内预测的块在给定块开始编码之前已完成；按照标准要求，这意味着在开始编码当前行中的编码树单元之前，必须确保前一行中的下一个编码树单元已完成。与公开可用的参考编码器 HM 相比，这些功能的结合在不损失效率的情况下显著提升了速度。

2.3 自适应比特率流媒体背景

自适应比特率是一种能够通过 HTTP 网络高效传输流媒体文件的技术。系统会向用户的视频播放器提供同一内容的多个不同尺寸文件，客户端则会选择最适合在设备上播放的文件。该技术旨在根据具体的设备和网络状况，在各种情况下传输合适的内容，从而减少重新缓冲的需要。自适应比特率流媒体使视频播放器能够根据可用比特率和设备能力，选择最适合播放的视频分段或数据块。

自适应码率阶梯是由不同质量和分辨率的视频片段组成的数组，这些视频片段可在流媒体服务器上获取。如果比特率提高，视频播放器的逻辑会选择质量更好且文件更大的文件，这类似于沿阶梯向上攀登。如果比特率降低，视频播放器则会切换回质量较低的文件，这类似于沿阶梯向下攀登。自适应比特率流媒体通过调整

流，以选择最合适的比特率，并根据当前情况改变传输条件。这使其非常适合在移动网络上进行流媒体传输。自适应比特率流媒体会动态跟踪 CPU、内存容量和网络状况等参数，随后提供相匹配的视频质量。

大多数现代视频播放器都支持自适应比特率。用户设备上的视频播放器可以从视频清单文件中提供的所有视频片段中进行选择，这些片段经过调整，以提高比特率。视频播放器会根据用户设备当时可用的带宽，选择最合适的视频片段。播放器最初会请求所提供的最低比特率视频片段。如果视频播放器判断下载速度高于当前视频片段所支持的比特率，就会请求下一个更高比特率的视频片段。这个过程会持续进行，直到当前视频片段的比特率与可用带宽达到适当匹配。只要带宽保持在相近水平，视频播放器就会继续以选定的带宽请求视频片段。如果这一过程顺利进行，用户在不同网络状况下都能获得流畅的观看体验。

网络类型	尺寸	帧 rate	总比特-码率	音频比-码率	Key 帧
Cell	480×320	N/A	64	64	N/A
Cell	416×234	10-12	264	64	30-64
Cell	480×270	10-12	464	64	90
WiFi	640x360	29.97	664	64	90
WiFi	640x360	29.97	1264	64	90
WiFi	960x540	29.97	1864	64	90
WiFi	960x540	29.97	2564	64	90
WiFi	1280x720	29.97	4564	64	90
WiFi	1280x720	29.97	6564	64	90
WiFi	1920x1080	29.97	8564	64	90

表 2.1：Apple 在 Apple 技术说明 TN2224 中提出的固定比特率编码阶梯[5]

Apple 于 2010年提出了固定比特率编码阶梯，如表 2.1 所示。在这种方案中，一个视频文件会被编码为十种不同质量的编码档位，范围从比特率为 64 kbps 的纯音频流，到比特率为 8564 kbps 的 1080p 音视频流。表中报告的手机 Apple 尺寸数据适用于纵向观看，而 Wi-Fi 对应的数据则适用于横向观看。

Netflix 于2015年通过按标题编码完善了自适应码率阶梯分析技术。在 Netflix 采用的这一特定方法中，每个视频都会以多分辨率进行编码。数据速率用于确定相应的阶梯档位，从而在每种数据速率下提供最佳质量，

具体如表 2.2 所示。这确保内容提供商可以根据每个特定视频的要求，自定义两个参数——阶梯中的档位数量，以及与之相关的分辨率，如下所示。这一策略有望节省编码和存储成本。

分辨率	固定码率阶梯（kbps）	按标题码率阶梯（kbps）
320x240	235	150
384x288	375	200
512x384	560	290
512x384	750	NA
640x480	1050	NA
720x480	1750	440
720x480	NA	590
1280x720	2350	830
1280x720	3000	1150
1920x1080	4300	1470
1920x1080	5800	2150
1920x1080	NA	3840

表 2.2：Netflix 提出的按标题编码 [5]

上下文感知编码（CAE）大约于 2018年 开始受到关注。它将编码考量扩展到了设备。借助上下文感知编码，内容提供商可以针对每种设备类型分别设置编码参数，并向各种设备类型交付内容，从智能手机到家庭影院。其目标是在保持相同体验的同时，降低播放带宽。播放带宽的这一降低幅度可能相当可观。图 2.2 所示的上下文感知码率阶梯可以通过为每个编码档位使用较低的比特率或更高的分辨率，以较少一半的编码档位提供与固定码率阶梯相同的质量，从而改善播放性能和成本效益。如果编码的目标是节省总带宽，那么内容的消费地点将是一个重要因素。传统的编码方法是为媒体库中的每个视频创建典型设置。对于视频需要流式传输到多种设备的情况，例如通过蜂窝数据网络连接的移动设备和有线机顶盒，建议针对预期使用的每种设备类型分别采用不同的编码方案。上下文感知编码确保内容提供商可以按节目或按场景对参数进行微调。

与本工作相关的自适应流媒体的关键特点是，需要多种表示以扩大覆盖范围。本工作提出了一种方法，用于减少为自适应流媒体分发而以多种分辨率和比特率编码相同视频内容时的计算冗余。



图 2.2：上下文感知编码 [5]

2.4 相关工作

该工作 [14] 提出了一种方法：使用未经修改的高效视频编码对最高比特率/质量表示进行编码。随后，根据编码单元深度信息对依赖表示进行编码。根据观察，在更高质量下，编码单元通常倾向于具有更高的深度值（更小的编码单元大小），因此，我们仅搜索其深度级别小于或等于参考表示中同位编码单元深度级别的编码单元。采用这种方法，可以跳过更大的深度进行搜索，从而节省大量时间。

沿着这一思路， [15] 探讨了一种类似的方法，即采用不同分辨率的表示。由于不同分辨率下的编码树单元涵盖的对应区域各不相同，因此彼此无法匹配。为此，本文提出了一种新颖的匹配算法，旨在从具有任意下采样比率的高分辨率表示中获取块结构信息。随后，利用这些信息跳过较低分辨率下不必要的深度搜索。 [16] 所述研究将上述两种方法结合起来，提出了一种高效的多码率编码方法。除编码树单元的深度值外，该方法还利用参考表示中的其他信息——即预测模式、帧内模式和运动矢量——来加速较低质量和较低分辨率表示的编码。

上述方法利用了高分辨率结果，并将其重新用于较低的“依赖”或派生分辨率。由于低分辨率内容的编码过程本质上更快，这种重新利用降低了对计算资源的需求。文献 [17] 提出了一种在 x265 的不同表示之间共享分析信息的方法，但在该方法中，所有表示都被限制为相同的分辨率。

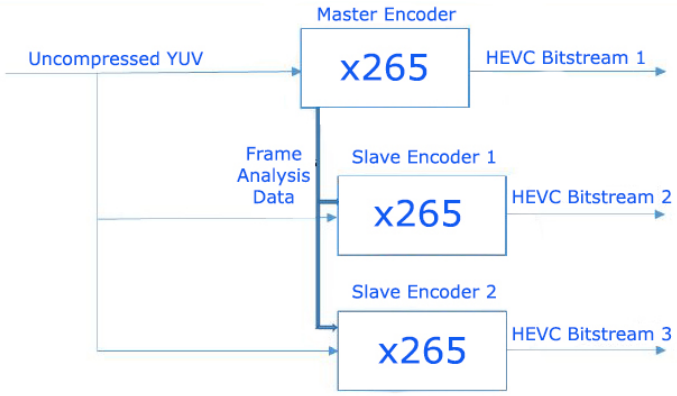

图 2.3： [16], [17]中提出的多码率实现

[18] 介绍了一种基于 x265 的多分辨率编码智能框架的新型实现。利用这一框架，在编码过程中作出的与编码决策相关的信息（例如四叉树结构、预测模式等）可以从低分辨率编码共享至高分辨率编码。这使高分辨率编码过程的速度最高提升至原来的两倍。

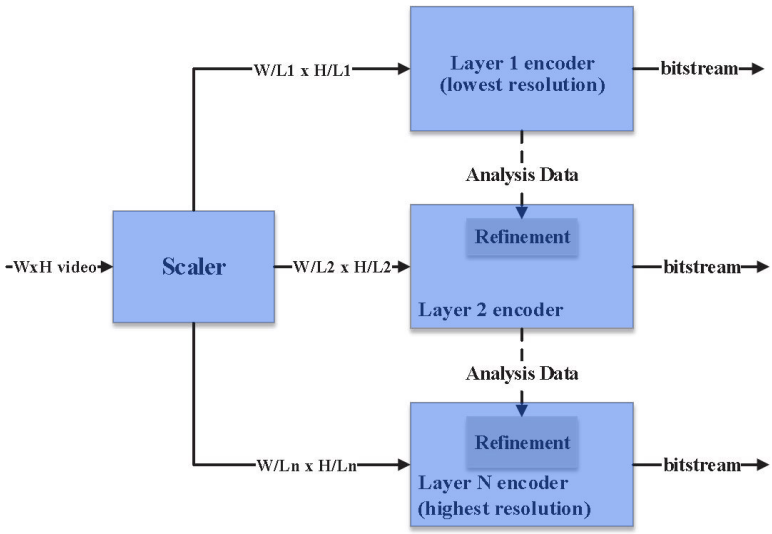

图 2.4： [18]中提出的多码率实现

第3章

XVC 与英特尔 SIMD 背景

本章简要介绍 XVC 视频编解码器和英特尔 SIMD 优化。第 3.1 节讨论 XVC 视频编解码器的背景，第 3.2 节介绍 SIMD 扩展的背景。第 3.2.1 节和第 3.2.2 节分别介绍 Intel SSE4 和 AVX2 SIMD扩展。第 3.3 节介绍本项目参考的部分文献。

3.1 XVC视频编解码器

名为 XVC 的视频编解码器由软件视频压缩公司 Divideon AB 开发，其首个版本于 2017 [19] 年 9 月发布。它的开发主要基于已纳入 AVC [20]或高效视频编码 [4], 的成熟技术，或是在 JVET [21]背景下评估过的技术。XVC 视频编解码器几乎在高效视频编码和 AV1 [22]所代表的现有选项之间提供了一种折中方案。它是一种基于块的编解码器，使用帧内预测和帧间预测，将由 YUV 像素样本构成的原始图像编码为预定义的标准比特流。视频序列中的每幅图像都划分为矩形样本块，尺寸最大为 64×64 像素；这些块根据同一图像中的样本（帧内预测）或先前编码图像中的样本（帧间预测）进行预测。残差采用非方形变换进行变换，编码后的符号则由 CABAC 编码器压缩。块边界经过去块滤波器滤波。

总体而言，XVC 与高效视频编码更为相似，但两者也存在若干明显差异。其中一个最显著的差异是，XVC 使用非正方形编码单元，并在这些编码单元上同时进行预测和变换。因此，XVC 不像高效视频编码那样，分别包含预测单元树和变换单元树。XVC 在多个方面扩展了高效视频编码中的技术，例如帧内预测模式从 35 种增加到 67 种。此外，XVC 还引入了大量全新的编码工具，在高效视频编码中没有对应技术，例如：

- 自适应运动矢量精度——用于指示运动矢量的精度

从而更高效地传输长运动矢量和整数运动矢量。

- 仿射运动预测——为编码单元的每个 4×4 子块计算并应用独立的运动矢量，从而能够更好地表示旋转和缩放等非平移运动。

- 跨分量预测——使用线性模型根据亮度样本预测色度样本。

- 变换选择——评估具有不同特性的变换，以确定哪种变换能够最高效地表示特定编码单元的残差。

- 局部光照补偿——使用线性模型描述从参考图像进行预测时样本值的局部偏移，尤其适用于表示物体或场景光照条件的变化。

XVC 编解码器已使用多种开源应用进行了测试，包括 ExoPlayer、FFmpeg 和 VLC。这些集成使 XVC 能够用于多种设备和平台，包括 Android、iOS、Windows 和 Linux。

3.2 单指令多数据扩展

SIMD 是 Flynn 分类法中的一类并行计算机 [23]。使用 SIMD 编程可以同时对多个数据元素执行相同的操作。这属于数据级并行，而非并发，即可以同时处理多个数据元素，但只执行一种操作。能够以数据级并行方式运行的程序，有望通过 SIMD 获得显著加速。

SIMD 的概念于 20 世纪 70 年代初提出，当时被称为带有单条指令的数据“向量”，但如今它已与 SIMD 分开。随后，SIMD 被引入现代 SIMD 机器中，这些机器包含许多功能受限的处理器。之后，SIMD CPU 的应用领域从台式计算机转向超级计算机。由于台式计算机在 20 世纪 90 年代开始具备足够高的性能，以支持实时游戏和音频/视频处理，SIMD 扩展又重新回到了台式机 CPU。

尽管 SIMD 非常适合用于加速程序，但在现实世界的应用中使用 SIMD 仍存在一些局限性。

- 并非所有程序都能通过单指令多数据加速，因为并非所有程序都能轻松实现向量化。有些程序只能进行部分向量化，但由于非向量化函数的串行执行，会出现性能瓶颈。

单指令多数据 指令集架构	基础指令集 架构	供应商	Year	SIMD 寄存器
MMX	x86	英特尔	1996	8x64b
3DNow!	x86	AMD	1998	8x64b
SSE	x86/x86-64	英特尔	1999	8/16x28b
SSE2	x86/x86-64	英特尔	2000	8/16x28b
SSE3	x86/x86-64	英特尔	2004	8/16x28b
SSSE3	x86/x86-64	英特尔	2006	8/16x128b
SSE4	x86/x86-64	英特尔	2007	8/16x128b
AVX	x86/x86-64	英特尔	2011	16x256b
XOP	x86/x86-64	AMD	2011	8/16x128b
AVX2	x86/x86-64	英特尔	2013	16x256b
AVX-512	x86/x86-64	英特尔	2016	32x512b

表 3.1：通用处理器的单指令多数据扩展 [24]

• 单指令多数据在加载和存储数据方面也存在局限，因为所有单指令多数据指令集架构的向量宽度各不相同。

• 通常，使用单指令多数据加速程序需要投入大量人工精力，因为内置函数代码或手写汇编代码的长度，显著长于 C 语言或 C++代码。

• 由于诸如数据对齐限制等原因，有些单指令多数据指令集架构比其他架构更复杂，而且并非所有处理器都支持所有单指令多数据指令集架构。研究新的单指令多数据扩展对程序的影响，仍需要投入大量人工精力。

3.2.1 英特尔流式SIMD扩展4

Intel SSE4 是一种单指令多数据 CPU 指令集，于 2006 年秋季的英特尔开发者论坛上发布，并与为前几代 Intel 64 和 IA-32 架构微处理器编写的软件完全兼容。所有现有软件都可以在集成 SSE4 的微处理器上继续正确运行，无需修改；即使同时运行集成 SSE4 的现有和新应用程序，也同样如此。Intel SSE4 包含 54 条指令。在一些英特尔文档中，由 47 条指令组成的子集称为 SSE4.1。此外，SSE4.2 是由其余七条指令组成的第二个子集。

3.2.2 英特尔高级向量扩展 2

AVX2 将 SSE4 中大多数已经引入的整数指令扩展至 256 位，并引入融合乘加运算。英特尔首次在 Haswell 处理器中支持这些功能，该处理器

于 2013 年推出。AVX2 使用 16 个 YMM 寄存器执行单指令多数据操作。每个 YMM 寄存器可容纳八个 32 位单精度浮点数或四个 64 位双精度浮点数，并可同时对它们执行运算。SSE 指令仍用于操作 YMM 寄存器的低 128 位。AVX2 引入了三操作数单指令多数据指令格式，其中目标寄存器与两个源操作数彼此不同。

3.3 相关工作

近年来，利用 SIMD 扩展优化程序已十分普遍，这有助于充分利用现有的 CPU 架构。对于英特尔 CPU 架构，不同代的 SIMD 扩展（如 SSE2、SSSE3、SSE4.1 和 AVX2）已用于加速 MPEG-2、MPEG-4 第2部分等不同的视频编解码器。近年来，H.264/AVC 和高效视频编码也通过 SIMD指令进行了优化。 [25] 中详细总结了 H.264/AVC 之前针对编解码器进行 SIMD优化的相关工作。H.264/AVC 也从 SIMD 加速中受益匪浅，包括亮度和色度插值滤波器、逆变换以及去块滤波器的加速。例如，Zhou 等人 [26], 和 Chen 等人 [27] 报告称，使用 SSE2 加速 H.264/AVC，可使应用程序的速度提升 2.0 到 4.0 倍。我此前的一项工作是针对英特尔酷睿 i9 和英特尔® 至强® 可扩展处理器，使用 AVX512 SIMD 扩展加速 x265 HEVC 编码器 [28]。图 3.1 展示了采用英特尔® AVX2 SIMD 优化的 x264 和 x265 编码器的编码器加速比测量结果。这促使我们对更复杂的 Arowana XVC 编码器进行 x86 优化，以获得更好的性能提升。

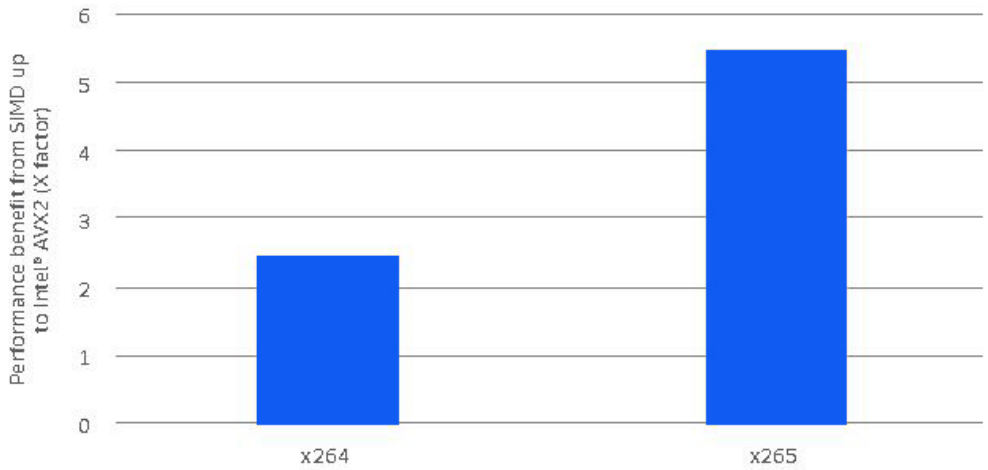

图 3.1：使用英特尔® 高级向量扩展 2，在英特尔® 酷睿™ i7-4500U 处理器上采用主配置文件进行 1080p 编码时，x264 和 x265 的性能提升[28]

第4章

x265 中的分析共享方法

本章简要介绍了本研究中用于 x265 多分辨率多码率分析共享的方法。第 4.1 节说明了如何测试本研究中考虑的各种方法。第 4.2 节介绍了帧内分辨率分析共享所采用的方法，其中第 4.2.1 节详细介绍最先进方法，第 4.2.2 节则阐述所提出的方法。第 4.3 节介绍了帧间分辨率分析共享所采用方法的详细内容。

4.1 测试方法

所使用方法的评估是在内容各异的不同类型序列上进行的。在报告中，我提到了使用表4.1中所述的知名短序列获得的结果。

测试视频				
视频	分辨率	FPS	色彩空间	帧数
BasketBallDrive	960x540 1920x1080 3840×2160	50	YUV420p	500
CrowdRun	960x540 1920x1080 3840×2160	50	YUV420p	500
DucksTakeOff	960x540 1920x1080 3840×2160	50	YUV420p	500

表 4.1：用于验证所采用方法的视频

本文的评估重点是 x265 [29]，的中等预设，它是 FFmpeg [13] 中默认的质量预设，并在编码速度和压缩效率之间实现了最佳权衡。

如第 4.2 节所述，为测试帧内分辨率分析共享，每种分辨率（540p 和 1080p）的测试序列均采用五种不同的量化参数表示进行编码，如表4.2所示。同样，如第 4.3 节所述，为测试跨分辨率分析共享，测试序列也采用五种不同的量化参数表示进行编码。在确定适用于帧内分辨率分析共享和跨分辨率分析共享的方法后，我们采用表4.3所述的 cVBR 码率，研究该方法对实时流媒体应用的影响（见第 6 节）。选择五种不同的量化参数或 cVBR 表示，是为了计算 BD-Rate 和 BD-PSNR 指标 [30] ，从而准确了解这些编码方法对压缩效率的影响。2160p 编码的结果仅针对 cVBR 表示进行计算。

CQP表示					
分辨率	QP				
960x540	22	26	30	34	38
1920x1080	22	26	30	34	38
3840×2160	22	26	30	34	38

表 4.2：不同分辨率下恒定量化参数（CQP）模式测试所采用的量化参数（QP）

码率表示					
分辨率	比特率（kbps）				
960x540	1000	1750	2500	3000	3500
1920x1080	3500	4500	5500	7500	9000
3840×2160	11000	13000	15000	17000	19000

表4.3：不同分辨率下采用的 cVBR 码率

采用 Bjøntegaard增量（BD-Rate 和 BD-PSNR）测量方法 [30]评估编码方法对压缩效率的影响。BD-Rate 计算相同质量下率失真曲线之间的比特率百分比差异。BD-PSNR 计算相同比特率下率失真曲线之间的平均质量差异。负 BD-PSNR 表示与参考方法相比，所提出方法的编码效率有所下降；正 BD-PSNR 则表示编码增益。同样，正 BD-Rate 表示与参考方法相比，所提出方法的编码效率有所下降；负 BD-Rate 则表示编码增益。

本研究的所有性能和质量数据均在一台运行 Windows 10、搭载 Intel® Core (TM) i7-7500U CPU 的系统上收集。表4.4列出了测试系统属性。

测试系统详细信息	
系统属性	值
操作系统名称	Windows 10 家庭版
版本	10.0 版本号 18363
系统型号	联想 80YL
系统类型	基于 x64 的个人计算机
处理器	英特尔 Core (TM) i7-7500U CPU @2.70Ghz（4 个中央处理器），2.9GHz
内存类型	DDR4
内存通道	1
内存大小	16384MB

表4.4：测试系统配置

4.2 x265 中的帧内分辨率分析共享

本实验对一种多质量编码系统进行建模和研究，该系统以多个质量级别对每个视频分段进行编码。第 4.2.1 节详细介绍了当前最先进的方法，其中最高质量表示是主/基准表示。第 4.2.2 节介绍了为改进当前最先进方法而提出的方法。

4.2.1 最先进的方法

首先对最高质量表示进行编码。编码器分析信息会与后续使用不同量化参数的编码共享。这与 [15], [16], 和 [17]中提到的方法所采用的设置相同。图4.1展示了分析共享方案的流程图。在自适应比特率系统中，各码率档位之间的间隔也必须尽可能接近，以避免在不同档位之间切换时出现明显的质量波动。这里，我们分别为 540p 和 1080p 编码设置了五个量化参数表示。对于该模型，将评估整个编码过程的 BD-Rate、BD-PSNR [30], 和编码器加速比。

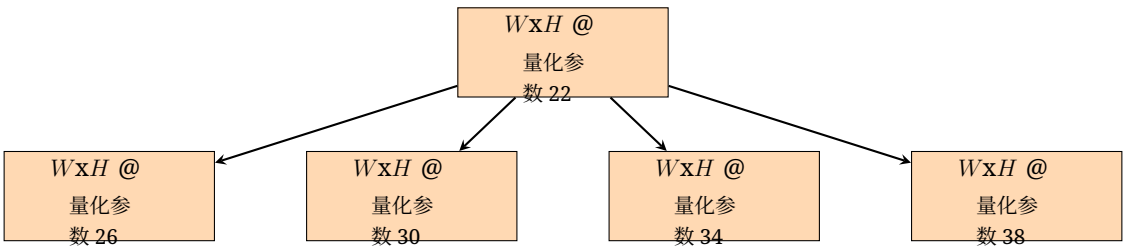

图4.1：在 WxH 分辨率编码中共享帧内分辨率分析，适用于恒定量化参数表示，其中量化参数最低的表示作为主表示

如附录A.1所示，该编码方案针对 x265 中定义的所有分析复用级别进行了评估。从表4.5可以看出，对于540p视频，x265分析复用级别10的平均加速比约为53%

，平均BD-Rate为12%。大多数当前最先进的方法都采用这一复用级别。我们还观察到，对于1080p视频，分析复用级别10带来了类似的加速比，但BD-Rate有所增加。不过，由于测试涵盖了较大的量化参数范围，不同表示中的编码单元深度和模式决策存在更大的差异；因此，BD-Rate的上升是预期之中的。

CQP表示测试				
分辨率	分析级别	ΔT	BD-Rate	BD-PSNR
960x540	4	15.87 %	**1.65 %**	**-0.07 分贝**
960x540	6	17.23 %	**1.65%**	-0.07 分贝
960x540	10	53.06%	**12.27%**	**-0.46 分贝**
1920x1080	4	11.57 %	**2.03 %**	**-0.06 分贝**
1920x1080	6	12.83 %	**2.03%**	**-0.06 分贝**
1920x1080	10	49.99%	**21.69%**	**-0.47 分贝**

表4.5：以最低量化参数（最高质量）表示作为主编码时，分辨率内分析共享的加速比和压缩效率测量值

4.2.2 所提出的方法

在上一节所述的模型中，与大多数文献一致，最高质量编码是主表示。这里有两个需要解决的关键问题：

- 在并行编码系统中，采用这种方法时，整体性能受主编码所需时间的限制。因此，在多质量系统中使用最高质量编码作为主表示会造成性能瓶颈。

- 当我们向较低质量表示遍历时，最高质量表示中的编码单元深度和模式决策并非最优。

为缓解上述问题，如图4.2所示，实验采用中位数质量表示作为主表示。

在测试中的五种量化参数表示中，量化参数 30 是中位数质量表示。因此，首先对量化参数 30 表示进行编码。编码器分析信息与后续的编码——量化参数 22、26、34 和 38——共享。通常，量化参数 22 的独立表示在所有独立编码中速度最慢，从而成为并行编码系统的性能瓶颈。但在这里，预计通过使用量化参数 30 表示的编码器分析信息，可以加快量化参数 22 的编码速度。

表 4.6 展示了所提出的方法在 540p 和 1080p 分辨率内分析共享中的加速比和 BD-Rate。表 4.7 比较了当前最先进水平的方法与所提出的方法在 BD-Rate 和 BD-PSNR 方面的表现。可以看到，

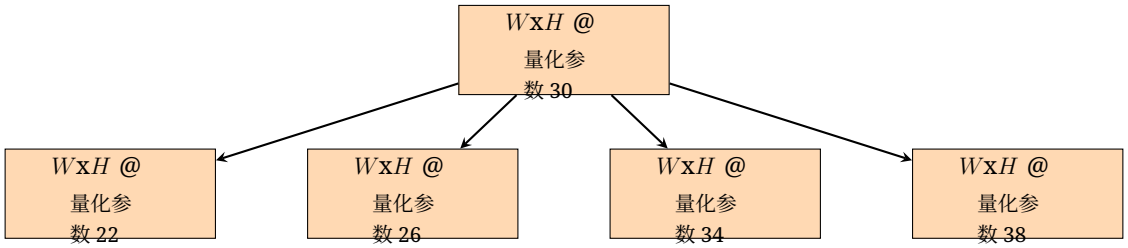

图4.2：对于以中位数QP作为主表示的恒定量化参数表示，在 WxH 分辨率编码之间共享帧内分辨率分析

所提出的方法能够显著降低 BD-Rate。因此，以中位数质量表示作为主表示的方法优于现有当前最先进水平的最高质量表示作为主表示的方法。

CQP表示测试				
分辨率	分析级别	ΔT	BD-Rate	BD-PSNR
960x540	4	16.94 %	0.80 %	-0.03 分贝
960x540	6	18.77 %	0.80 %	-0.03 分贝
960x540	10	65.60 %	6.45 %	-0.26 分贝
1920x1080	4	13.91 %	1.61 %	-0.05 分贝
1920x1080	6	15.07 %	1.61 %	-0.05 分贝
1920x1080	10	63.91 %	8.59 %	-0.25 分贝

表 4.6：以中位数 QP 表示作为主编码时，分辨率内分析共享的加速比和压缩效率测量值

如前所述，当前最先进的方法的性能瓶颈在于，并行编码的总体周转时间受 QP 22 编码速度的限制。因此，需要对每个分辨率层级表示进行详细的加速比分析。图 4.3 展示了对于采用 x265 分析共享模式 10 进行 1080p 编码时，当前最先进的方法和所提出的方法中每个恒定量化参数表示的加速比。所有速度均以 QP 22 独立编码速度进行归一化。在并行编码中，对于当前最先进的方法，编码这五种表示所需的时间等于 QP 22 编码所用的时间。对于所提出的方法，QP 22 表示所用的时间等于编码这五种表示所需的总时间，但可以观察到，该时间减少到了原来的 86%。这意味着并行编码速度提升了 14%。

4.3 x265 中的跨分辨率分析共享

在本实验中，对一种系统进行了建模：该系统以较低分辨率下的最高质量对每个视频片段进行编码，然后重新利用分析信息中的帧级元数据

		最高质量主版本		中等质量主版本	
分辨率	分析级别	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR
960x540	4	1.65 %	-0.07	0.80 %	-0.03 分贝
960x540	6	1.65%	-0.07	0.80%	-0.03 分贝
960x540	10	12.27 %	-0.46	6.45 %	-0.26 分贝
1920x1080	4	2.03 %	-0.06	1.61 %	-0.05 分贝
1920x1080	6	2.03%	-0.06	1.61%	-0.05 分贝
1920x1080	10	21.69 %	-0.47	8.59 %	-0.25 分贝

表 4.7：以中位数质量表示作为主编码的帧内分辨率分析共享的 BD-Rate 平均改进

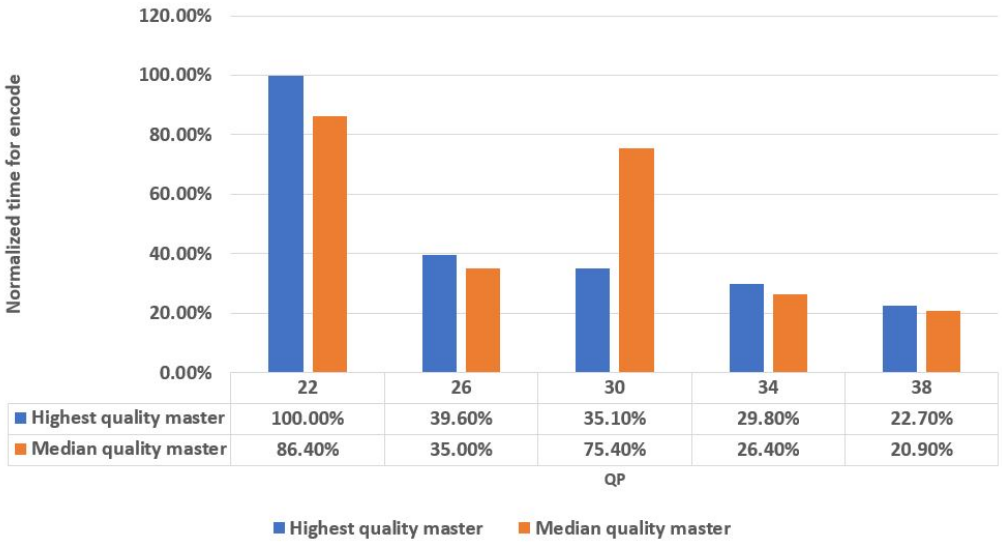

图 4.3：1080p视频帧内分辨率分析共享中所有表示的平均编码时间

对当前分辨率层级中的表示进行编码。该设置以示意图形式呈现在图4.4中。对于此模型，将评估当前分辨率层级编码过程中的BD-Rate、BD-PSNR和整体编码器加速比。

CQP表示测试				
分辨率	细化	ΔT	BD-Rate	BD-PSNR
1920x1080	no	**72.04 %**	**20.08 %**	**-0.62 分贝**
1920x1080	yes	**28.99 %**	**1.61 %**	**-0.07 分贝**

表 4.8：以 540p 表示作为主编码、分析复用级别 10 下，1080p 跨分辨率分析共享的加速比和 BD-Rate 测量结果

从表 4.8 可以看出，通过复用最高

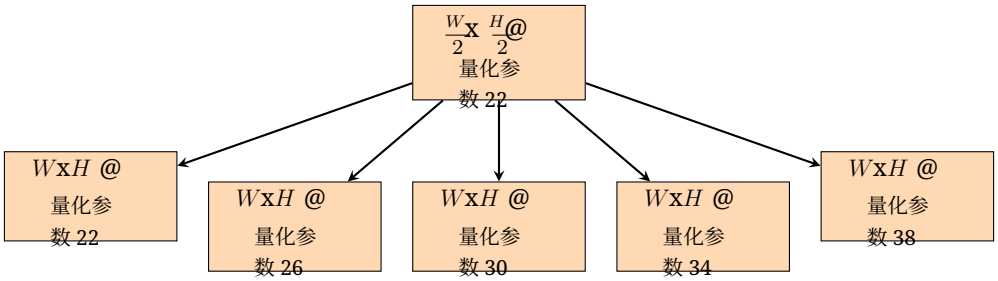

图4.4：CQP表示的帧间分辨率分析共享

质量表示的分析信息（该表示来自未经分析细化的 540p 编码），1080p分辨率层级的编码总体加速比达到 72.04%，BD-Rate 为 20.08%，BD-PSNR 为 -0.62 分贝。还可以看出，通过复用最高质量表示的分析信息（该表示来自经过分析细化的 540p 编码），1080p分辨率层级的编码总体加速比达到 28.99%，BD-Rate 为 1.61%，BD-PSNR 为 -0.07 分贝。因此可以看出，在不同分辨率之间细化分析，是一种能够实现可观加速比且使 BD-Rate 增幅可以忽略的出色策略。

第5章

用于 XVC 编码的英特尔 SIMD 优化方法

本章介绍了为了解 SIMD 并行性对媒体处理影响而开展的探索性工作。第 5.1 节展示了用于实现单指令多数据的各种方法，并说明了选择手写汇编方法的原因。第 5.2 节介绍了本项目用于编译手写汇编代码的 NASM 汇编器。第 5.3 节展示了使用 Intel SSE4 和 AVX2 SIMD 加速的内核在 CPU 周期数方面的改进。第 5.4 节介绍了如何选择要优化的函数，第 5.4.1 节和第 5.4.2 节则分别描述了编码器代价函数的实现以及由此带来的 CPU 周期数改进。第 5.5 节说明了用于评估每个优化函数 CPU 周期数改进的方法。

5.1 单指令多数据实现方法

在英特尔处理器上利用 SIMD指令主要有三种方法 s:

- 编写底层汇编代码
- 使用编译器支持的内在函数
- 编译器自动向量化

通常，编写使用SIMD指令和可用寄存器的底层汇编代码，被认为是实现高性能的最佳方法。然而，这种方法烦琐且容易出错。编译器支持的内在函数提供了更高层次的抽象，其与汇编指令几乎一一对应，但无需处理寄存器分配、指令调度、类型检查和调用栈维护。在这种方法中，内在函数会以内联方式展开，从而消除函数调用开销。它具备与内联汇编相同的优势，同时提高了代码可读性并减少错误 [31]。其代价是，整体性能提升取决于编译器跨多个内在函数调用进行优化的能力。

编译器自动向量化将所有工作交由编译器完成，依赖编译器定位合适的循环结构并自动对其进行向量化。因此，编译器的质量以及程序员编写有助于自动向量化的代码的能力至关重要。 [31]表明，当前最先进水平的编译器只能对实际应用代码中的18-30 %进行向量化。特别值得注意的是，编译器无法执行一些有助于自动向量化的关键代码变换。研究发现，非单位步长内存访问、数据对齐以及数据依赖变换，都是编译器自动向量化中的重要问题。在我们的工作中，我们选择底层汇编代码方法来执行SIMD操作。

5.2 NASM 汇编器支持

必须添加汇编器支持，才能将手写汇编代码编译到视频编码解决方案中。在本项目中，Arowana XVC 编码器解决方案添加了对 Netwide Assembler（NASM）的支持 [32]。NASM 是适用于英特尔 x86 架构的汇编器和反汇编器，可用于编写 16 位、32 位（IA-32）和 64 位（x86-64）程序。它是根据简化版（两条款）BSD 许可证发布的开源软件。我们对解决方案的编译方法进行了必要的更改，使 NASM 能够生成汇编代码的已编译目标文件，并将这些文件链接到编码器解决方案中。

5.3 使用英特尔 SIMD 加速 Arowana XVC 编码

图 5.1 展示了使用 SSE4 英特尔 SIMD 加速的各个内核的周期计数改进。这些内核按照其相对于对应 C++实现的周期计数增益递增顺序排列。附录 B 展示了每个优化内核相对于其对应 C++ 实现的详细性能改进。

图 5.2 展示了使用 AVX2 英特尔 SIMD 加速的各个内核的周期计数改进。附录 C 展示了每个优化内核相对于对应 C 语言++ 实现的详细性能增益。周期计数结果表明，在内核层面，使用英特尔 SIMD 加速 XVC 是有益的。

5.4 选择要加速的内核

从 XVC 的核心计算中选出了超过 1100+ 个内核，采用英特尔 SIMD 指令针对 main10 配置进行优化。这些内核是根据其

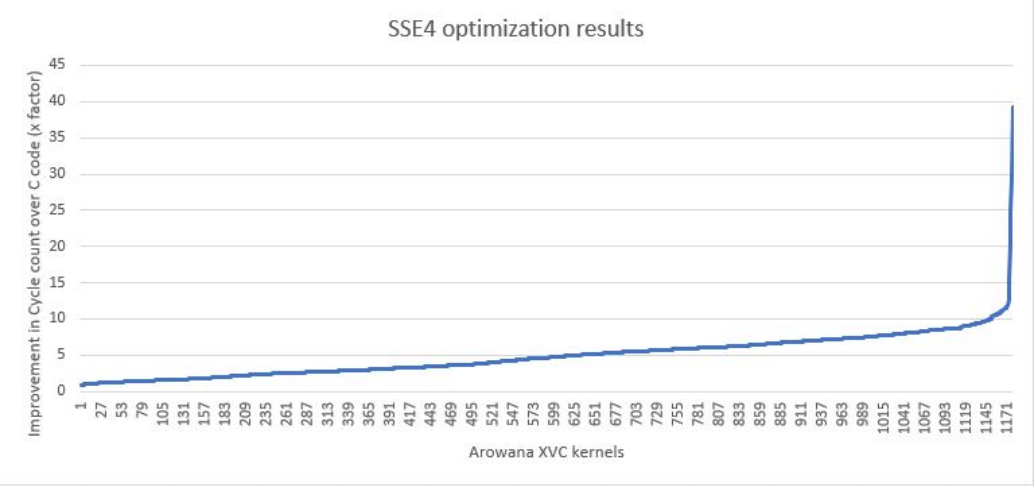

图 5.1：英特尔®流式SIMD扩展4内核相较于对应 C++内核的周期计数增益

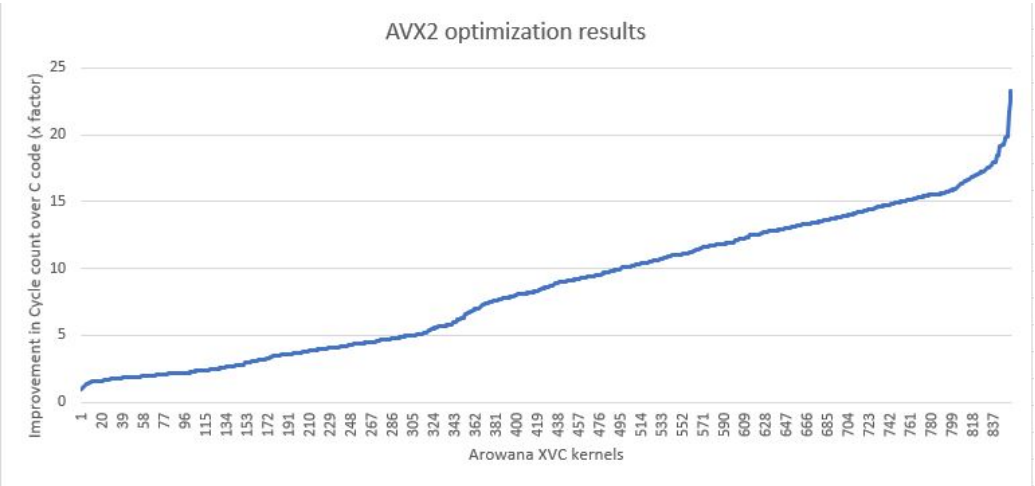

图 5.2：英特尔® 高级矢量扩展 2内核相较于对应 C++内核的周期计数增益

资源需求选出的。有些内核可能需要频繁进行内存访问，例如不同的块复制和块填充内核；另一些内核则可能涉及密集计算，例如 DCT、反离散余弦变换和量化内核。此外，还有第三类内核同时涉及这两种操作，只是所占比例各不相同。使用英特尔 SIMD 指令优化的内核完整列表见附录 B 和附录 C。

5.4.1 代价函数

SAD（绝对差值和）、SATD（绝对变换差值和）和离散余弦变换（DCT）是 Arowana XVC 编码器中最复杂且调用频率最高的一些函数。在视频编码中，会使用许多代价函数来评估并选择最佳编码

模式及其相关参数。SAD 和 SATD 是运动估计中用于寻找整数和四分之一像素运动矢量的两种主要代价函数。SAD 约占 XVC 编码器总 CPU 周期的 11% 至 13%，SATD 则占 14% 至 15%。这两种代价函数的定义如下：

$$SAD = \Sigma_{i,j}^{I,J} \mid O(i,j) - P(i,j) \mid \tag{5.1}$$

$$SATD = \frac{\Sigma_{i,j}^{I,J} \mid H(i,j) \mid}{2} \tag{5.2}$$

其中，i 和 j 是像素索引，其取值范围由块大小决定。O(i,j) 和 P(i,j) 分别是原始像素值和预测像素值。H(i,j) 是预测误差 O(i,j) - P(i,j) [33]的 Hadamard变换。由于代价函数中仅涉及加法和减法运算，与 DCT 相比，SATD 能够以相对较低的复杂度在变换域中得到准确的代价。由于两者都对多个数据应用相同的操作，因此向量指令对于减少所需的时钟周期非常有用。

5.4.2 SIMD实现代价函数

SAD函数

下面给出了 C++代码的 SAD函数实现：

```
1 t e m p l a t e <i n t w i d t h , i n t h e i g h t , ty pen ame SampleT1 >2 i
```

The 下面展示了宽度 16 的块大小对应的 x86手写汇编 :

```
%宏 绝对差值和 16x2 高位深度
```

```
        打包字减法      m2, [r0 + r1]
        打包字绝对值    m1, m1
        打包字绝对值    m2, m2
        打包字加法      m1, m2
        打包字乘法累加 m1, m6
        打包双字整数加法指令 m0, m1
        lea             r0, [r0 + 2 * r1]
        lea             r2, [r2 + 2 * r3]
%endrep
        HADDD           m0, m1
        movd            eax, xm0
%end宏

初始化_YMM avx2
c语言global     绝对差值和_16x2, 4, 4, 7
        SAD_16x2_HBD_AVX2 2
        RET
C 语言全局      绝对差值和_16x4, 4, 4, 7
        SAD_16x2_HBD_AVX2 4
        RET
C 语言全局      绝对差值和_16x8, 4, 4, 7
        SAD_16x2_HBD_AVX2 8
        RET
C 语言全局      绝对差值和_16x16, 4, 4, 7
        SAD_16x2_HBD_AVX2 16
        RET
C 语言全局      绝对差值和_16x32, 4, 4, 7
        SAD_16x2_HBD_AVX2 32
        RET
全局 C 语言     SAD_16x64, 4, 4, 7
        SAD_16x2_HBD_AVX2 64
        RET
```

表 5.1 展示了 SAD函数优化内核相较于 C 语言++ 实现所带来的性能增益。

采用 SSE4 和 AVX2优化时的加速比					
内核	加速比	内核	加速比	内核	加速比
绝对差值和[4x4]	3.01x	绝对差值和[8x64]	2.63x	绝对差值和[32x8]	4.95x
绝对差值和[4x8]	4.41倍	绝对差值和[16x2]	3.39倍	绝对差值和[32x16]	4.99倍
绝对差值和[4x16]	3.67x	绝对差值和[16x4]	2.44x	绝对差值和[32x32]	5.80x
绝对差值和[4x32]	5.54x	绝对差值和[16x8]	4.58x	绝对差值和[32x64]	5.54x
绝对差值和[8x4]	2.90x	绝对差值和[16x16]	6.46x	绝对差值和[64x8]	5.30x
绝对差值和[8x8]	2.40x	绝对差值和[16x32]	5.79x	绝对差值和[64x16]	5.37x
绝对差值和[8x16]	5.72x	绝对差值和[16x64]	6.27x	绝对差值和[64x32]	5.27x
绝对差值和[8×32]	2.78倍	绝对差值和[32×4]	4.47倍	绝对差值和[64×64]	5.14倍

表 5.1：采用 SSE4 和 AVX2优化的 SAD 内核加速比

SATD函数

SATD函数的 C 语言++ 代码如下所示：

```
1 模板 <类型名称 SampleT1，整数 w，整数 h>2 // 计算 8x4 个整数块中的 satd 3 整数
sa
长
数
co
12
s (
符
21
Sa
{ 2
号
像
34
1 |
数
1 |
SU
(
][
=
tm
```

```
总和 +=绝对值2（a0） +绝对值2（a1） +绝对值2（a2） +绝对值2（a3）； }
返

```

The 下面给出了块大小为宽度 16 时对应的 x86手写汇编 :

```
全局 计算 16 x 4 的 SATD_ – ; 函数 用于计算 SATD 成本适用于 16 列 , 4 行
; 行 0-3
movu m0, [行0]
movu m4, [行2]
打包字减法 m0, m4
movu m1, [r0 +r 1]
movu m5, [r2 +r 3]
打包字减法 m1, m5
movu m2, [r0 +r 1 * 2]
movu m4, [r2 +r 3 * 2]
打包字减法 m2, m4
movu m3, [r0 +r 4]
movu m5, [r2 +r 5]
打包字减法 m3 , m5
lea r0 , [r 0 + r1 * 4]
lea r2 , [r 2 + r3 * 4]
打包字加法 m4, m0, m1
打包字减法 m1, m0
打包字加法 m0, m2, m3
打包字减法 m3, m2
高位字交错解包 m2, m4, m1
punpcklwd m4, m1
高位字交错解包 m1, m0, m3
punpcklwd m0, m3
打包字加法 m3, m4, m0
打包字减法 m0, m4
打包字加法 m4, m2, m1
打包字减法 m1, m2
高双四字解包交错 m2, m3, m0
p u n p c k l d q m3, m0
打包字加法 m0, m3, m2
打包字减法 m2, m3
高双四字解包交错 m3, m4, m1
p u n p c k l d q m4, m1
打包字加法 m1, m4, m3
打包字减法 m3 , m4
punpckhqdq 指令 m4 , m0 , m1
p u n p c k l q d q m0, m1
打包字绝对值 m0 , m0
打包字绝对值 m4, m4
pmaxsw m0, m0, m4
punpckhqdq 指令 m1, m2, m3
p u n p c k l q d q m2, m3
打包字绝对值 m2, m2
```

```
45          打包字绝对值                m1, m1
46          pmaxsw                      m2, m1
47          pxor                        m7, m7
48          mova                        m1, m0
49          punpcklwd                   m1, m7
50          打包双字整数加法指令        m6, m1
51          mova                        m1, m0
52          高位字交错解包              m1, m7
53          打包双字整数加法指令        m6, m1
54          pxor                        m7, m7
55          mova                        m1, m2
56          punpcklwd                   m1, m7
57          打包双字整数加法指令        m6, m1
58          mova                        m1, m2
59          高位字交错解包              m1, m7
60          打包双字整数加法指令        m6, m1
61          返回
62
63  %宏 SATD        _AVX2_END 0
64          vextracti128                xm7, m6, 1
65          打包双字整数加法指令        xm6,xm7
66          打包异或指令                xm7,xm7
67          高低位打包单精度浮点数移动指令  xm7,xm6
68          打包双字整数加法指令        xm6,xm7
69          pshufd                      xm7, xm6,  1
70          打包双字整数加法指令        xm6, xm7
71          movd                        eax, xm6
72  %endmacro
73
74  ; 静态      int     satd_16x4(const SampleT1* pix1, intptr_t stride_pix1,
75  ;                               const 示例 * pix2, intptr_t stride_pix2,
76  ;                               int offset)
77  %宏 SATD        _16x4_AVX2 1
78  c 全局      satd_%1_16x4,  4,6,8
79          shl                         r1d, 1
80          shl                         r3d, 1
81          lea                         r4, [3 * r1]
82          lea                         r5, [3 * r3]
83          pxor                        m6, m6
84          调用                        cal _SATD _16x
85          SATD_AVX2_END
86          RET
87  %endmacro
```

表5.2展示了SATD函数优化内核相较于其 C++ 语言实现所带来的性能增益。使用SSE4和AVX2 SIMD扩展时，对于不同的内核大小，可观察到3到10倍的加速比。

采用 SSE4 和 AVX2优化后的加速比					
内核	加速比	内核	加速比	内核	加速比
satd[8x4]	3.45x	satd[16x16]	7.32x	SATD[32x64]	8.72x
SATD[8x8]	3.75x	SATD[16x32]	8.15x	SATD[64x8]	8.43x
SATD[8x16]	4.21x	SATD[16x64]	8.85x	SATD[64x16]	9.03倍
SATD[8x32]	4.01倍	SATD[32x4]	5.61倍	SATD[64x16]	9.19x
SATD[8x64]	4.40x	SATD[32x8]	7.83x	SATD[64x32]	9.81x
SATD[16x4]	5.19x	SATD[32x16]	7.75x	SATD[64x64]	10.22x
SATD[16x8]	6.40x	SATD[32x32]	8.56x		

表5.2：采用 SSE4 和 AVX2优化的 SATD 内核加速比

5.5 评估周期计数改进的方法

实现手写 x86 SIMD 汇编代码时，需要评估两个标准：

- 如果实现的原语正确，即输出相同，则将同一组输入分别馈送给 C++ 原语和优化后的原语。
- 如果实现的原语正确，则可通过优化后的原语相对于 C++ 原语获得加速比。

在这种情况下，Arowana XVC 编码器针对 x86 汇编内核实现了一个正确性与性能测量工具。为评估上述第一项标准，该工具接收给定原语的有效参数，调用 C++ 原语和相应的汇编内核，并比较两个输出缓冲区。该工具使用一组随机分布的值，验证给定输入类型的所有可能边界情况。每个汇编内核都会被调用 100 次，并与其 C++ 原语的输出进行对比，以确保正确性。对于第二项标准，即衡量性能提升，测试基准程序对汇编内核和 C++ 内核运行 1,000 次时的时钟周期差异（由 rdtsc 指令报告）进行测量，并报告两者之间的平均值。

第6章

结果与讨论

本章将在实时流媒体场景中评估前几章所述的方法。在流媒体行业当前最先进的应用中，视频编码采用 cVBR 模式。因此，本节使用 x265 中的 cVBR 模式，介绍第4节所述方法的结果。每种分辨率测试所使用的 cVBR 码率如表4.3所示。第6.1节展示了当前最先进的分辨率内分析共享方法，并讨论针对 540p、1080p 和 2160p 这些分辨率所获得的结果。第6.2节讨论所提出的分辨率内分析共享方法，以及其相较于传统分辨率内分析共享方法的改进。第6.3节解释跨分辨率分析共享方法、分析细化的必要性及相关结果。第6.4节和第6.5节分别讨论当前最先进的多编码器结构和所提出的多编码器结构，以及用于编码器分析共享的方案和相关结果。第6.6节解释利用 SIMD 扩展提升 Arowana XVC 编码速度的改进。

6.1 x265 中最先进的分辨率内分析共享

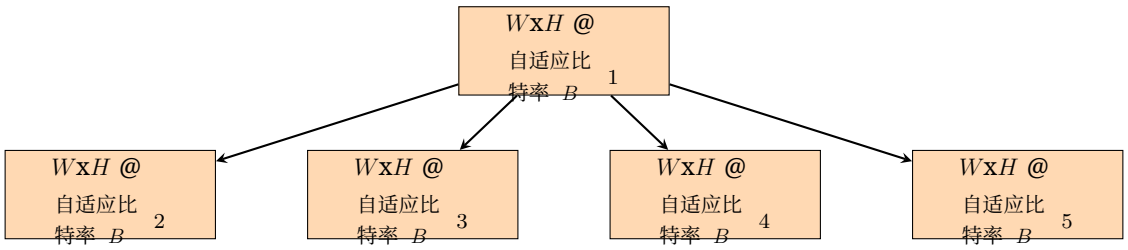

图6.1：采用最先进方法（$B_1 > B_2 > B_3 > B_4 > B_5$）对自适应比特率表示进行跨 WxH 分辨率编码的帧内分辨率分析共享

在此模型中，cVBR B_1 编码实例是主表示。后续

编码使用主表示中的分析信息。实验结果针对表 4.1 中提到的测试视频以及表 4.3 所示的 cVBR 码率进行评估。此处仅使用表 A.1 中提到的 x265分析复用级别 10，因为近期文献均采用该分析共享模式。

可以观察到，结果与恒定量化参数表示的结果非常相似。如表6.1所示，对于540p视频，五种cVBR表示的总体加速比为54.00%，BD-Rate为8.79%，BD-PSNR为 $-0.36dB$。对于1080p视频，总体加速比为54.62%，BD-Rate为9.23%，BD-PSNR为 $-0.27dB$。对于2160p视频，总体加速比为58.00%，BD-Rate为8.22%，BD-PSNR为$-0.25dB$。

自适应码率表示测试			
分辨率	ΔT	BD-Rate	BD-PSNR
960x540	54.00 %	8.79 %	-0.36 分贝
1920x1080	54.62 %	9.23%	-0.27 分贝
3840×2160	58.00%	8.22%	-0.25 分贝

表 6.1：采用当前最先进水平的方法进行 ABR帧内分辨率分析共享时的加速比和 BD-Rate 测量结果

6.2 x265 中提出的帧内分辨率分析共享

在此模型中，cVBR B_3 编码实例是主表示。后续编码使用来自主表示的分析信息。

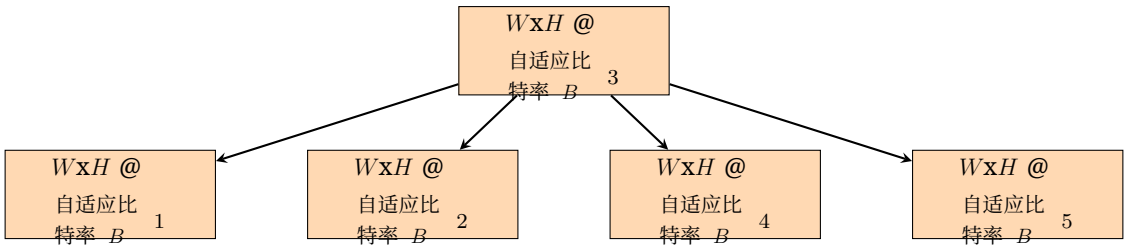

图6.2：采用所提出的方法（$B_1 > B_2 > B_3 > B_4 > B_5$）进行ABR表示的 WxH 分辨率编码中的帧内分辨率分析共享

如表6.2所示，对于540p视频，五种cVBR表示的总体加速比为56.14%，BD-Rate增加7.75%，BD-PSNR为 $-0.27dB$。对于1080p视频，总体加速比为59.15%，BD-Rate增加6.47%，BD-PSNR为 $-0.18dB$。对于2160p视频，总体加速比为60.09%，BD-Rate增加6.01%，BD-PSNR为 $-0.17dB$。

自适应码率表示测试			
分辨率	ΔT	BD-Rate	BD-PSNR
960x540	56.14 %	7.75 %	-0.27 分贝
1920x1080	59.15 %	6.47 %	-0.18 分贝
3840×2160	60.09%	6.01%	-0.17 分贝

表6.2：采用所提出的方法进行ABR帧内分辨率分析共享时的加速比和BD-Rate测量结果

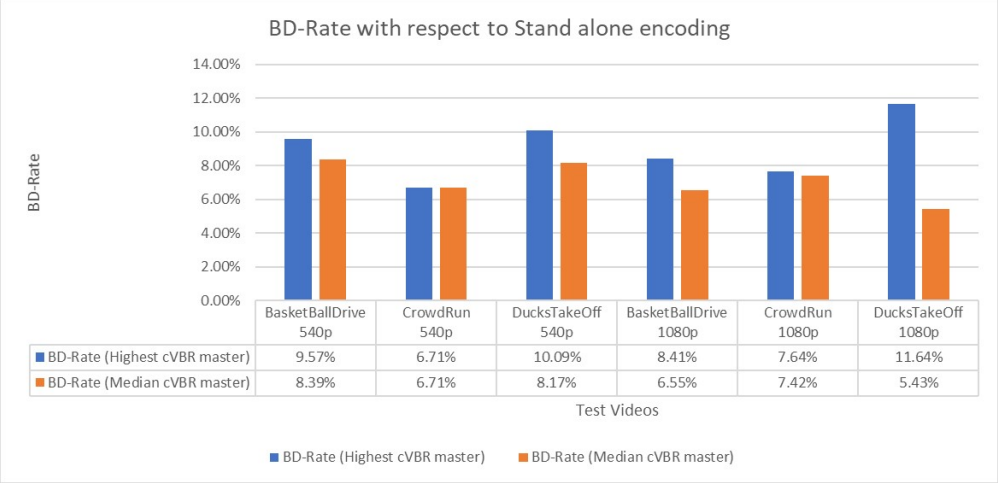

图6.3：现有最先进方法与所提出的方法在 cVBR独立编码中的 BD-Rate 比较

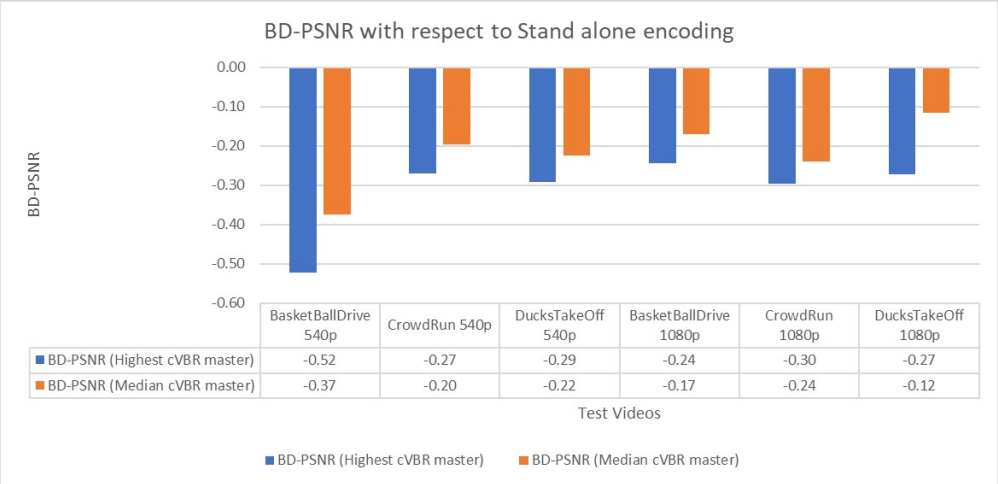

图6.4：现有最先进方法与所提出的方法在 cVBR独立编码中的 BD-PSNR 比较

6.3 x265 中的跨分辨率分析共享

在此模型中，$\frac{W}{2}$ x $\frac{H}{2}$ 分辨率的 cVBR B 编码实例为主表示。随后，对 WxH 分辨率层级进行编码时，会使用主表示中的分析信息。结果针对表 A.1 中提到的 x265分析复用级别10进行评估。在 WxH 分辨率层级使用五种 cVBR 码率来计算 BD-Rate 和 BD-PSNR [30]。

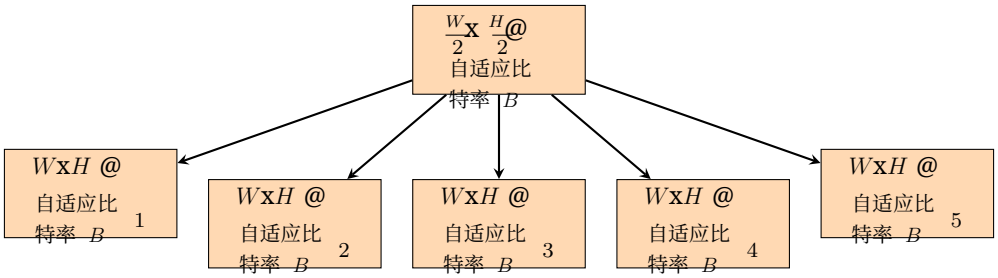

图6.5：自适应码率表示的帧间分辨率分析共享（B_1 > B_2 > B_3 > B_4> B_5）

从表6.3可以看出，以540p中位数恒定可变码率表示作为主编码时，1080p分辨率层级的加速比为65.64%，BD-Rate为20.20%，BD-PSNR为 $-0.64dB$。以1080p中位数恒定可变码率表示作为主编码时，2160p分辨率层级的加速比为63.96%，BD-Rate为25.91%，BD-PSNR为 $-0.40dB$。我们注意到，在不进行细化而使用全部编码单元分析信息时，BD-Rate值过高。因此，我在x265中使用细化选项，以获得合理的BD-Rate。x265中的分析细化选项在A.2节中进行了说明，其中帧内细化选项和帧间细化选项分别在表A.2和表A.3中进行了说明。

自适应码率表示测试			
分辨率	ΔT	BD-Rate	BD-PSNR
1920x1080	65.64 %	20.20 %	-0.64 分贝
3840×2160	63.96 %	25.91 %	-0.40 分贝

表6.3：分别以540p中位比特率表示和1080p中位比特率表示作为主编码时，1080p和2160p自适应比特率跨分辨率分析共享的加速比和BD-Rate测量结果

从表 6.4 可以看出，以 540p 中位数恒定可变码率表示作为主编码时，1080p 分辨率层级的加速比为 50.45%，BD-Rate 为 8.62%，采用分析细化时 BD-PSNR 为 $-0.21dB$ 。以 1080p 中位数恒定可变码率表示作为主编码时，2160p 分辨率层级的加速比为 52.82%，BD-Rate 为 7.45%

自适应码率表示测试			
分辨率	ΔT	BD-Rate	BD-PSNR
1920x1080	50.45 %	8.62 %	-0.21 分贝
3840×2160	52.82 %	7.45 %	-0.22 分贝

表 6.4：以 540p 中位比特率表示和 1080p 中位比特率表示分别作为主编码时，采用细化的 1080p 和 2160p 自适应比特率跨分辨率分析共享的加速比和 BD-Rate 测量结果

以及采用分析细化时的 $-0.22dB$ BD-PSNR。图 6.6 和图 6.7 展示了对采用分析细化和未采用细化的跨分辨率 cVBR 独立编码进行比较时 BD-Rate 和 BD-PSNR 的图形表示。

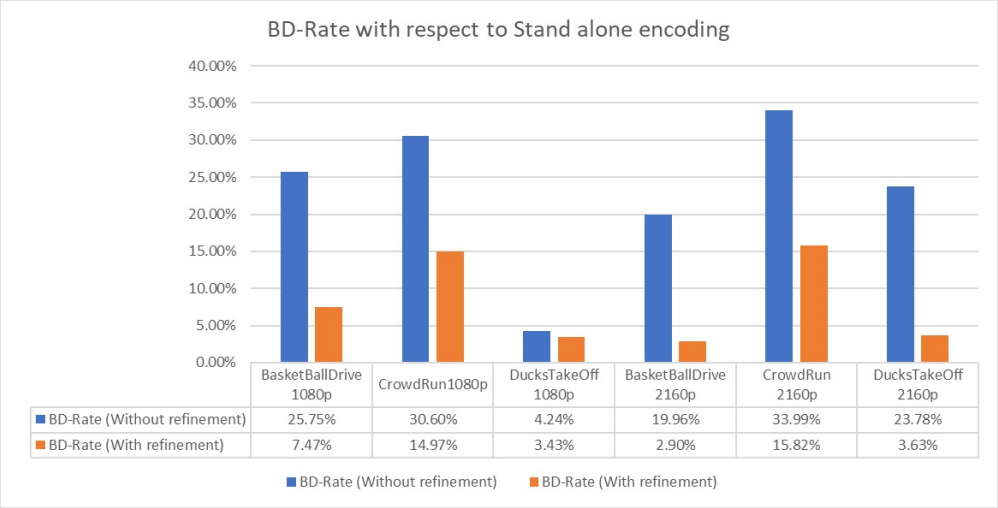

图 6.6：采用分析细化与不采用细化的跨分辨率 cVBR独立编码的 BD-Rate 对比

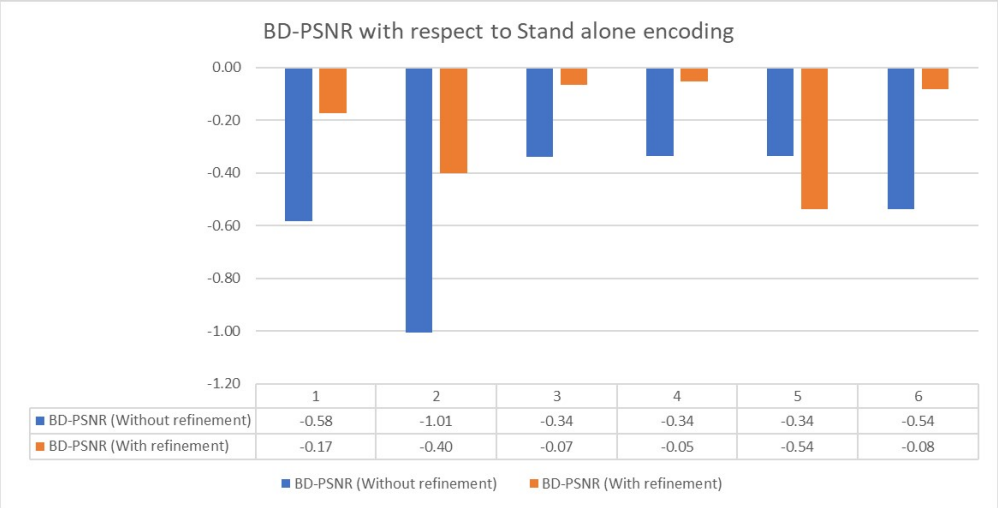

图 6.7：采用分析细化和不采用细化的跨分辨率 cVBR 独立编码的 BD-PSNR 比较

6.4 当前最先进水平的多编码方案

图6.8展示了使用 x265 HEVC 编码器所采用的当前最先进水平的多编码方案。540p 3.5 兆比特每秒 cVBR表示的编码器分析数据被用作主表示，以编码其他 540p 表示。相同的数据经过细化后用于编码 1080p 9.0 兆比特每秒表示。换言之，最低分辨率层级中最高的 cVBR表示被用于编码同一层级中的其余表示，并经过细化后用于编码后续分辨率层级中最高的 cVBR表示。

6.5 所提出的多编码方案

在找到最优的分辨率内和跨分辨率分析共享方案后，提出了一种新型多编码方案，用于生成适用于自适应比特率流媒体的各种分辨率和比特率的 cVBR 比特流。在图 6.9 中，使用最优分析共享方案对所提出的多编码方法进行了示意。540p 2.5 兆比特每秒的 cVBR 表示是主编码表示。编码器分析信息与其他 540p 分辨率表示共享。该信息还与 1080p 5.5 兆比特每秒表示共享，后者对分析数据进行细化并加以复用。来自 1080p 5.5 兆比特每秒表示的分析信息用于编码其他 1080p 分辨率表示以及 2160p 15 兆比特每秒 cVBR 表示。与当前最先进水平的方案相比，所提出的方法平均可减少 7% 的 BD-Rate，并提升 9% 的加速比。

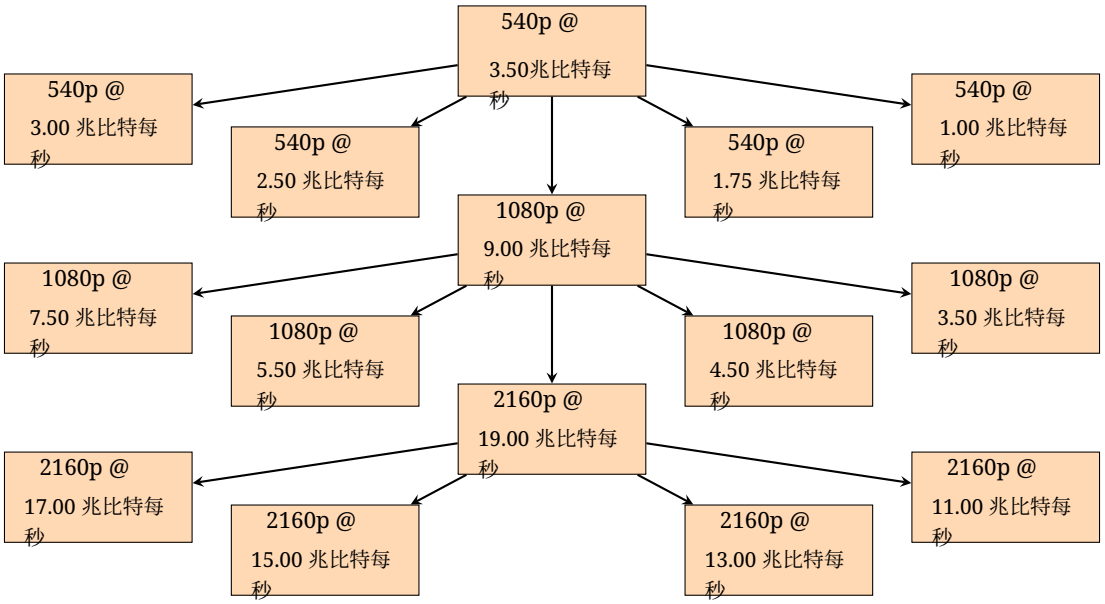

图6.8：跨残差和比特率的最先进的编码器分析共享表示

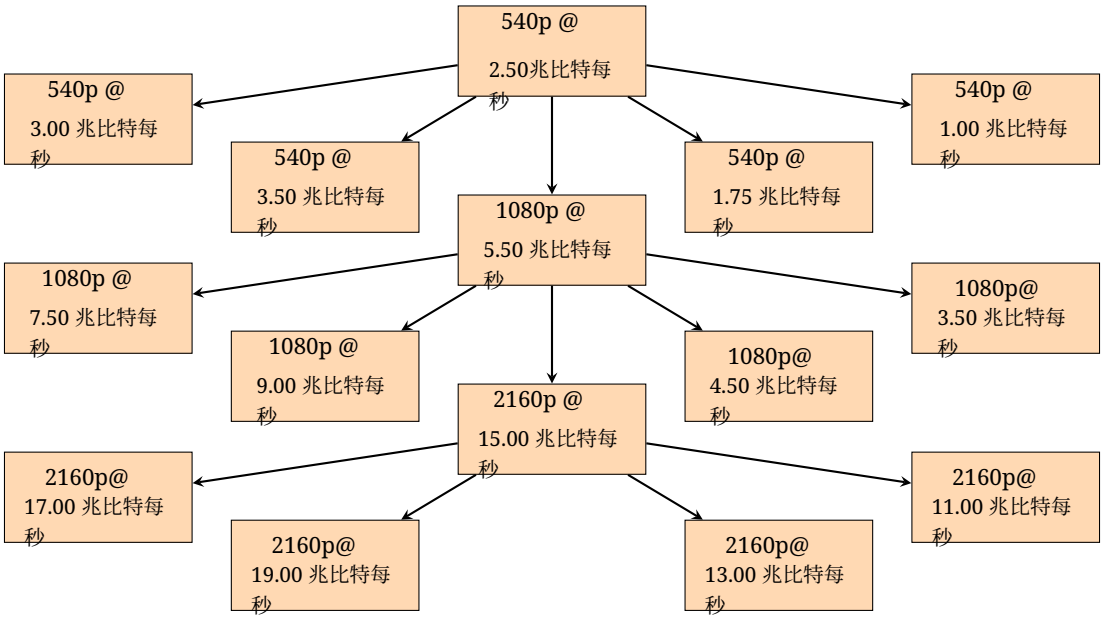

图 6.9：所提出的跨分辨率和比特率的编码器分析共享表示

6.6 通过单指令多数据优化改进 XVC 编码

实验涵盖 Arowana XVC 编码器的所有预设，以展示 XVC 的广泛应用场景，因为这些预设体现了压缩效率与编码速度（以每秒帧数衡量）之间的多种权衡。“慢速”预设生成的编码结果效率最高，但速度最慢；它也是离线编码场景的首选。“超超快”预设是 Arowana XVC 最快的设置，但生成的编码结果效率最低。“非常快”预设则体现了性能与编码器效率之间的折中。通常，效率更高的预设会采用更多 XVC 工具，因此每个像素需要的计算量也比效率较低的预设更多。

图6.10、图6.11 和图6.12 展示了在 540p分辨率下，Arowana 编码器中所有预设对给定测试视频的编码速度提升。我们注意到，在某些情况下，加速比净增约 2.5 倍。这表明，优化视频编码器中频繁使用的函数可以显著提高编码速度。结果主要来自对代价函数和插值滤波函数的优化。变换等关键函数尚未优化。函数向量化程度越高，加速比也越大。

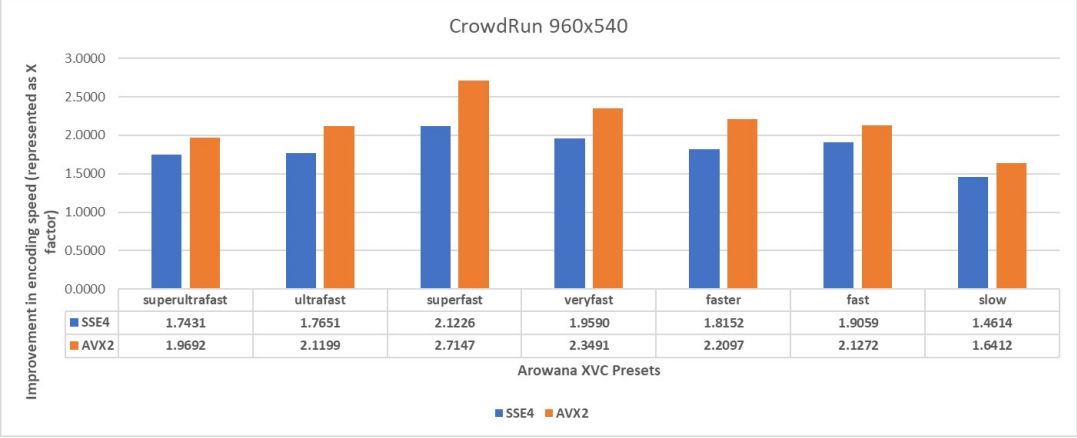

图6.10：CrowdRun视频在Arowana XVC预设下的XVC编码加速比（GOP长度=2 秒）

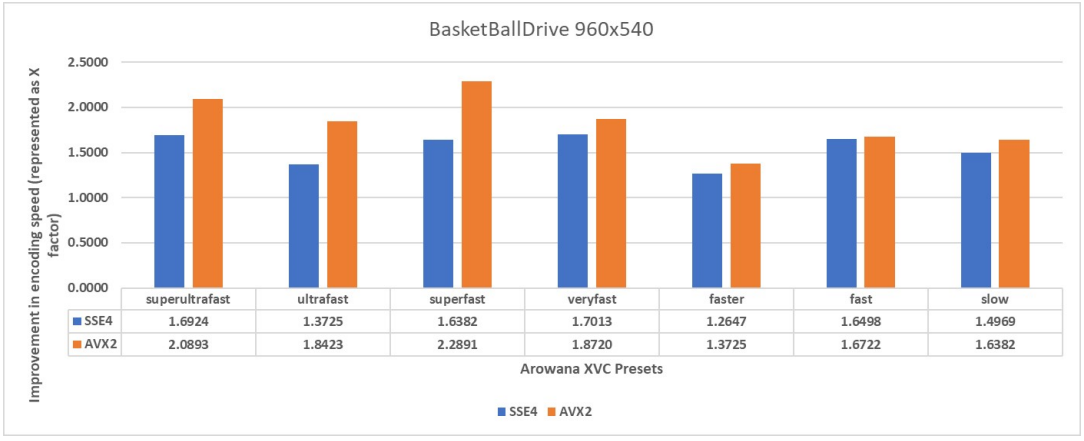

图6.11：Arowana XVC预设下BasketBallDrive视频的XVC编码加速比（GOP长度=2 秒）

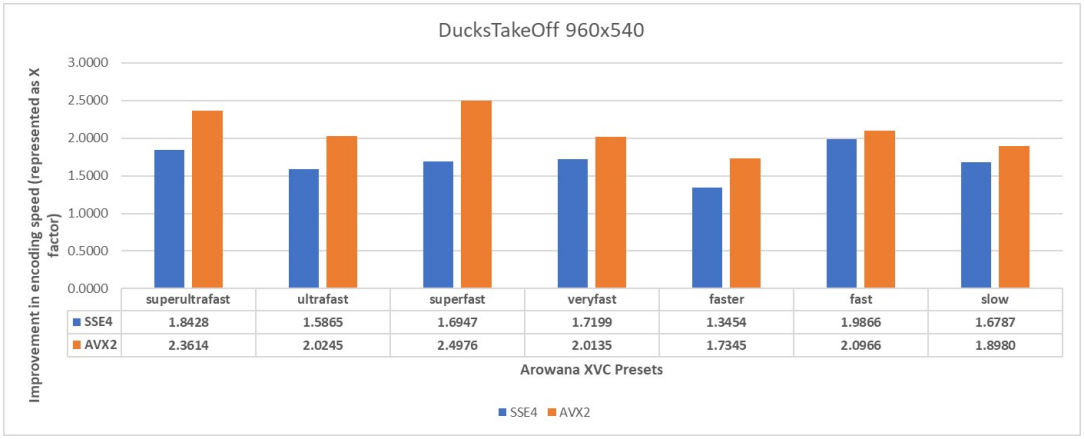

图6.12：Arowana XVC预设下DucksTakeOff视频的XVC编码加速比（GOP长度=2 秒）

第7章

结论与未来方向

本章总结项目报告，并概述项目期间的主要发现。第7.1节讨论结论，第7.2节讨论本项目尚未探索的未来方向。

7.1 结论

本项目对当前最先进的分辨率内多码率 HEVC 编码方案进行了建模和评估。该方案采用最高比特率表示作为主表示。为缓解并行编码中的性能瓶颈并提高整体压缩效率，本文提出了一种用于分辨率内多码率 HEVC 编码的新方案，其中采用中位比特率表示作为主表示。本文还评估了采用跨分辨率分析共享的当前最先进的多分辨率 HEVC 编码方案。基于更新后的优化方案，本文提出了一种用于自适应比特率流媒体的多编码方案，该方案采用多分辨率层级多码率 HEVC 编码。在所提出的方案中，同一分辨率层级的编码实例会复用中位 cVBR 编码实例的编码器决策。随后，这些编码器决策经过细化，并在后续分辨率层级中位 cVBR 编码实例中复用。所有方案均在 x265 HEVC 编码器上进行了评估。实验结果表明，该方案显著提升了加速比，同时对编码效率的影响微乎其微。

本项目评估了 SIMD 扩展优化对下一代视频编解码器编码速度的影响。这些优化在 Arowana XVC 编码器中实现，通过对一些关键代价函数和插值滤波函数进行向量化，使编码速度最高提升至原来的 2.7 倍。此外还发现，手写汇编是实现最佳性能提升的最佳方法，尽管这需要付出大量人力。

7.2 未来方向

由于硬件限制，本项目未考虑和讨论各种编码实例的并行编码结果。在第6.5节讨论的多编码模型中，$^{W}_{4}\mathbf{x}^{H}_{4}$ 分辨率层级的其他表示可以与 $^{W}_{4}\mathbf{x}^{H}_{4}$ 中位数恒定可变码率表示以帧延迟方式并行运行。此外，$^{W}_{2}\mathbf{x}^{H}_{2}$的中位数恒定可变码率表示也可以与 $^{W}_{4}\mathbf{x}\ ^{H}_{4}$中位数恒定可变码率表示以帧延迟方式并行运行。这种编码结构适用于配备多个 CPU 核心和线程的高端服务器。并行编码方法可以扩展本项目中提到的性能优势。

在本项目中，我们只对 Arowana XVC 编码器中的少数关键函数进行了优化。Arowana 中仍有许多可向量化代码，例如 DCT、iDCT、转置及其他函数；对它们进行优化，可以显著提升所有预设下的编码器速度。据[28], x265 在针对英特尔 SIMD 的所有代际技术（最高至 AVX2）实施优化后，加速比达到 5.5 倍。Arowana 是一款更复杂的编码器，通过全面优化可获得更大的性能提升。高端服务器上的性能提升还可通过 AVX512 优化进一步扩大。

参考文献

[1] Sandwine. *The Global Internet Phenomena Report*. 2020年5月。url: https:// www.sandvine.com/phenomena.[2] UltraHD Forum. *Phase A* 实施端到端指南，*v1.4*。2017年9月。url: https : // ultrahdforum . org/ resources / phasea- guidelinesdescription/.[3] T. Fautier. “超高清全球服务部署更新”。In: *NAB 2018*，拉斯维加斯，内华达州（2018年4月）。[4] G. J. Sullivan 等。“高效视频编码（HEVC）标准概述”。In: *IEEE* 电路与系统视频技术汇刊22.12（2012年），第 1649–1668 页。[5] AT&T 开发者计划。自适应码率视频流。url: https:// developer.att.com/video-optimizer/docs/best-practices/adaptive-bitrate-video-streaming.[6] R. Lee。“在 PA-RISC 处理器上通过软件解压缩实现实时 MPEG 视频”。In: 1995年1月，第 186–192 页。doi: 10.1109/CMPCON.1995.512384.[7] K. M. Patel、B. Smith 和 L. A. Rowe。“伯克利软件 MPEG-1 视频解码器”。In: *ACM* 多媒体计算、通信与应用汇刊1（2005年），第 110–125 页。[8] ITU-T/ISO/IEC。*ITU-T，H.265* 建议书（*2013*年*4*月），*H* 系列：视听与多媒体系统，视听服务基础设施——运动视频编码，高效视频编码。url: https://www.itu.int/rec/ TREC-H.265/en.[9] ITU-T/ISO/IEC。*ITU-T，H.265* 建议书（*2019*年*11*月），*H* 系列：视听与多媒体系统，视听服务基础设施——运动视频编码，高效视频编码。url: https://www.itu.int/rec/ TREC-H.265/en.[10] A. Elrowayati 等。“HEVC 视频编码标准中的数字水印算法综述与分析”。In: 2014年12月。doi: 10 . 13140 / RG . 2 . 1 . 1768.8809.[11] G. J. Sullivan 和 T. Wiegand。“视频压缩的率失真优化”。In: *IEEE* 信号处理杂志15.6（1998年），第 74–90 页。

[12] HandBrake。开源视频转码器。网址：https://handbrake. fr/.[13] FFMPEG。用于录制、转换和流式传输音频和视频的完整跨平台解决方案。网址：http://ffmpeg.org/.[14] D. Schroeder、P. Rehm 和 E. Steinbach。“多码率高效视频编码的块结构复用”。载于：*2015*年 *IEEE* 图像处理国际会议（*ICIP*）。2015年9月，第3972–3976页。[15] D. Schroeder、A. Ilangovan 和 E. Steinbach。“具有多分辨率的基于HEVC的自适应HTTP流媒体的多码率编码”。载于：*2015*年 *IEEE* 第*17*届多媒体信号处理国际研讨会（*MMSP*）。2015年10月，第1–6页。[16] D. Schroeder 等。“基于HEVC的自适应HTTP流媒体的高效多码率视频编码”。载于：*IEEE*《视频技术电路与系统汇刊》。2018年1月，第143–157页。[17] D. Nandakumar 等。“用于自适应比特率流媒体的高效多码率HEVC编码”。载于：*IEEE SMPTE*运动成像期刊。2017年，第36–40页。[18] K. Goswami 等。“用于自适应比特率流媒体的自适应多分辨率编码”。载于：*2018*年 *IEEE* 第*25*届图像处理国际会议（*ICIP*）。2018年，第1008–1012页。[19] Divideon。*XVC*简介——采用创新许可模式的新型视频压缩格式。网址：https://www.divideon.com/wp-content/uploads/2017/09/Introducing-xvc-a-Divideon-whitepaper- v1.0.pdf。[20] ITU-T/ISO/IEC。*H.264/14496-10*高级视频编码。网址：https://www.itu.int/rec/T-REC-H.264。[21] JVET。*JEM*软件。网址：https://jvet.hhi.fraunhofer.de/.[22] 开放媒体联盟。*AV1*视频编解码器。网址：https://aomedia.org/ av1/.[23] M. Flynn。“Flynn 分类法”。载于：并行计算百科全书。D. Padua 编。美国马萨诸塞州波士顿：Springer US，2011年，第689–697页。ISBN：978-0-387-09766-4。DOI：10.1007/978-0-387-09766-4_2。网址：https://doi.org/10. 1007/978-0-387-09766-4_2。[24] C. C. Chi 等。“HEVC解码的SIMD加速”。载于：《视频技术电路与系统汇刊》25（2015年5月），第841–855页。DOI：10.1109/TCSVT.2014.2364413。[25] V. Lappalainen 和 T. Hämäläinen。“媒体指令集架构扩展及其在视频编码中的应用研究概述”。载于：《视频技术电路与系统汇刊》12（2002年9月），第660–670页。DOI：10.1109/TCSVT.2002.800865。

[26] X. Zhou、E. Li 和 Y. Chen。“在通用处理器上使用媒体指令实现H.264解码器”。载于：*2003*年电子成像光学与光子学国际学会12（2003年），第224–235页。[27] Y. Chen 等。“在个人计算机上实现H.264编码器和解码器”。载于：《视觉通信与图像表示》期刊17（2006年4月），第509–532页。DOI：10.1016/j.jvcir.2005.05.004。[28] P. K. Tiwari、V. V. Menon 和 J. Murugan。使用英特尔加速*x265*®高级矢量扩展 *512*。网址：https :// software . intel. com / sites/ default/files/managed/d5/f6/mcw-intel-x265-avx512.pdf。[29] MulticoreWare 公司。*x265* 命令行选项。网址：https://x265.readthedocs. io/en/default/cli.html。[30] G. Bjøntegaard。“率失真曲线之间平均PSNR差异的计算”。载于：*VCEG-M33* 技术报告，*ITU-T SG16/Q6*。2001年。[31] S. Maleki 等。“向量化编译器评估”。载于：*2011*年并行架构与编译技术国际会议（*PACT*）。2011年，第372–382页。[32] NASM 开发团队。*NASM*。网址：https://www.nasm.us/。[33] K. MCCANN。“HM4：高效视频编码（HEVC）测试模型4编码器说明”。载于：*JCTVC-F802*（2011年）。网址：https://ci.nii.ac.jp/naid/ 20000680833/en/。

附录 A

X265 命令行选项

A.1 x265 编码器分析共享模式预览 s

x265 命令行参数 *–analysis-save* 用于从主/基础编码生成相应的编码器分析信息，并将其保存为元数据文件。在主/基础编码中，附加参数 *–analysis-save-reuse-level* 定义了元数据文件中保存的信息范围，如表 A.1 所示。随后，依赖表示通过结合 *–analysis-load* 和 *–analysis-load-reuse-level* 参数，重新利用元数据文件中的信息。依赖表示既可以与主/基础表示具有完全相同的分辨率，但针对不同的质量级别；也可以具有不同于主/基础表示的分辨率。*–scale-factor*选项用于将编码器分析数据缩放至所需分辨率。目前，x265 仅允许在二进制倍增分辨率之间进行此类分析共享（即宽度和高度均按两倍因子变化）。共享分析数据包括编码器决策，例如四叉树结构、预测模式（如帧内/帧间预测模式）、运动矢量（运动矢量），以及帧级信息，如切片类型、参考列表和失真信息。分析共享级别的选择取决于压缩效率与加速比之间的权衡。在本文中，我仅考虑级别 4、6 和 10。

x265 分析复用级别	
级别	描述
1	复用前瞻信息
2-4	级别 1 + 复用帧内/帧间模式 , 参考
5-6	级别 2 + 复用矩形非对称分区
7	级别 5 + 复用 AVC 大小 CU 优化
8-9	级别 5 + 复用 AVC 大小完整 CU 分析信息
10	级别 5 + 复用完整 CU 分析信息

表 A.1：x265 分析复用级别及其相应说明 [29]

A.2 x265 编码器分析细化模式预览

对于多分辨率编码，x265 使用特定的细化技术。命令行选项 *–refine-intra* 用于启用当前表示中帧内块的细化。该参数的不同取值及其相应功能如表A.2所示。

帧内细化	
级别	描述
0	强制采用参考编码中的模式和深度
1	评估当前深度(n)和深度(n+1)处的所有帧内模式 当当前块大小比最小 CU 大小大一级时，并且 强制对较大块使用相应模式。
2	除具备级别 1 的功能外，在所有深度强制 (i) 当保存的编码器选择角度模式时，仅使用深度 code. (ii) 当保存的编码器选择其他帧内模式时，使用深度和模式 保存的编码。
3	对从参考编码中复用的深度执行帧内模式分析 编码
4	不复用任何分析信息——重新对该 帧内块

表A.2：x265 [29] 中的帧内细化技术

命令行选项 *–refine-inter* 用于启用当前表示中帧间块的细化。该参数的不同取值及其相应功能如表A.3所示。命令行选项 *–refine-mv* 用于启用缩放视频的运动矢量细化。系统会根据所选级别评估最佳运动矢量。不同级别及其相应功能如表A.4所示。

帧间细化	
级别	描述
0	强制使用参考编码中的模式和深度
1	在当前深度(n)和深度(n+1)下评估所有帧间模式 当当前块大小比最小 CU 大小大一级时，并且 强制对更大的块使用指定模式。
2	除了级别 1 的功能外，还会限制模式评估 当特定模式被 保存的编码 保存的编码中的 2Nx2N —— 禁用对矩形分区和 AMP 的重新评估 保存的编码中的跳过模式——仅重新评估跳过模式、合并模式和 2Nx2N 模式。
3	在复用参考编码中的深度时，执行帧间模式分析 编码

表A.3：x265 [29] 中的帧间细化技术

运动矢量细化	
级别	描述
1	围绕缩放运动矢量搜索
2	级别 1 + 围绕最佳 AMVP 候选项搜索
3	级别 2 + 围绕另一个 AMVP 候选项搜索

表 A.4：x265 [29] 中的运动矢量细化技术

附录 B

SSE4 SIMD 优化

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
sad_ac_p[1][8x4]	0.84	绝对差值和_AC_像素[1][64x8]	1.2	绝对差值和_AC_残差[1][64x16]	1.36
均值_差值_像素[1][4x8]	0.9	减去_残差[32x16]	1.2	子_权重[32x8]	1.38
减去_像素[64x64]	0.92	绝对差值和_ac_残差[0][64x8]	1.21	绝对差值和_ac_残差[0][16x8]	1.4
减去_像素[32x32]	0.95	绝对差值和_ac_p[0][16x2]	1.22	绝对差值和_ac_残差[1][16x32]	1.4
减去_残差[64x32]	0.95	绝对差值和_交流_残差[0][32x4]	1.22	绝对差值和_交流_残差[0][8x4]	1.41
绝对差值和_交流_p[1][8x2]	0.98	减去_像素[64x16]	1.23	绝对差值和_交流_残差[1][32x8]	1.41
复制_双向预测[64x32]	1.02	sub_权重[64x32]	1.23	均值_差异_p[1][8x2]	1.42
sub_权重[64x64]	1.04	绝对差值和_ac_残差[0][64x64]	1.24	均值_差异_残差[1][8x2]	1.43
均值_差异_p[1][4x2]	1.06	复制_双向预测[32x8]	1.24	绝对差值和_ac_p[0][32x16]	1.44
绝对差值和_ac_p[0][32x8]	1.07	减去_残差[64x8]	1.25	减去_像素[64x32]	1.44
复制_双向预测[64x64]	1.07	绝对差值和_交流_残差[1][8x4]	1.26	计算_符号[64x8]	1.44
减去_像素[32x64]	1.08	减去_残差[32x32]	1.26	绝对差值和_交流_p[0][16x32]	1.45
绝对差值和_交流_残差[1][64x8]	1.1	绝对差值和_ac_残差[1][64x32]	1.27	减_权重[64x8]	1.45
绝对差值和_ac_p[0][64x64]	1.1	减去_残差[32x4]	1.27	绝对差值和_ac_残差[1][4x2]	1.46
复制_双向预测[32x16]	1.1	绝对差值和_交流_残差[1][8x8]	1.28	均值_差异_残差[1][4x4]	1.46

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
copy_双向预测[32x64]	1.1	绝对差值和_ac_残差[0][32x64]	1.28	绝对差值和_交流_残差[0][8x64]	1.46
减去_像素[32x8]	1.1	减去_像素[64x8]	1.28	绝对差值和_交流_残差[1][64x64]	1.46
绝对差值和_交流_p[1][16x32]	1.11	绝对差值和_交流_残差[0][16x2]	1.29	绝对差值和_交流_残差[1][8x32]	1.48
均值_差异_残差[1][4x2]	1.12	绝对差值和_交流_残差[0][8x2]	1.3	绝对差值和_交流_残差[1][32x4]	1.48
绝对差值和_交流_残差[1][8x2]	1.12	绝对差值和_交流_残差[0][32x32]	1.3	绝对差值和_ac_残差[1][32x16]	1.48
减去_残差[64x64]	1.12	绝对差值和_ac_p[1][8x8]	1.31	绝对差值和_ac_p[1][32x32]	1.48
绝对差值和_ac_p[1][16x2]	1.14	子项_权重[32x16]	1.31	绝对差值和_ac_残差[1][32x32]	1.48
绝对差值和_ac_残差[1][16x2]	1.14	绝对差值和_ac_残差[0][64x16]	1.32	绝对差值和_ac_p[1][32x64]	1.48
复制_双向预测[64x16]	1.15	减去_残差[32x8]	1.32	绝对差值和_ac_p[1][64x64]	1.48
减去_残差[32x64]	1.15	减去_残差[64x16]	1.32	子_权重[32x4]	1.48
复制_双向预测[32x32]	1.17	计算_符号[16x2]	1.33	绝对差值和_ac_残差[1][8x64]	1.49
绝对差值和_ac_残差[0][64x32]	1.18	sub_权重[32x32]	1.34	绝对差值和_ac_p[0][16x16]	1.49
绝对差值和_ac_p[1][32x4]	1.19	均值_差异_p[1][16x2]	1.35	绝对差值和_ac_p[0][64x32]	1.49
复制_双向预测[32x4]	1.19	绝对差值和_ac_p[1][16x16]	1.36	角度_快速_复制[8x2]	1.49
sub_权重[32x64]	1.19	绝对差值和_ac_残差[0][32x8]	1.36	减去_像素[32x16]	1.49
绝对差值和_交流_p[0][64x16]	1.5	绝对差值和_交流_p[0][8x32]	1.63	平均_差异_p[0][4x4]	1.78
绝对差值和_交流_p[1][4x2]	1.51	绝对差值和_交流_p[1][16x8]	1.63	绝对差值和_交流_残差[0][4x2]	1.79
绝对差值和_交流_p[0][4x4]	1.51	计算_符号[64x32]	1.63	均值_差值_残差[0][8x2]	1.79
绝对差值和_交流_残差[0][4x4]	1.51	绝对差值和_交流_残差[1][8x16]	1.65	绝对差值和_ac_残差[1][16x16]	1.81
绝对差值和_ac_p[1][8x32]	1.52	绝对差值和_ac_p[1][8x64]	1.65	绝对差值和_ac_p[0][16x64]	1.81
绝对差值和_ac_p[0][32x64]	1.52	绝对差值和_ac_残差[0][8x16]	1.66	计算_符号[32x16]	1.81
复制_双向预测[64x8]	1.52	绝对差值和_交流_残差[0][16x32]	1.66	绝对差值和_空间_x3[32x64]	1.83
减法_权重[64x16]	1.52	绝对差值和_交流_帧间[0][4x16]	1.67	复制_双向预测[16x8]	1.83
绝对差值和_ac_p[1][16x4]	1.53	绝对差值和_ac_残差[0][8x32]	1.67	绝对差值和_ac_p[0][8x4]	1.85
减去_像素[4x8]	1.53	平均_差值_p[0][16x4]	1.67	绝对差值和_交流_残差[0][8x8]	1.86
绝对差值和_交流_残差[1][4x8]	1.54	均值_差异_残差[1][16x4]	1.67	绝对差值和_交流_残差[0][16x4]	1.86
绝对差值和_交流_p[0][8x2]	1.54	复制_块[1]	1.67	绝对差值和_交流_功率[0][4x2]	1.87
绝对差值和_交流_功率[0][64x8]	1.54	均值_差值_功率[0][4x2]	1.68	绝对差值和_交流_功率[1][4x4]	1.88
计算_符号[32x4]	1.54	均值_差值_功率[1][4x4]	1.68	均值_差异_残差[1][16x8]	1.88
绝对差值和_交流_p[0][8x16]	1.55	绝对差值和_交流_p[0][32x32]	1.68	减去_像素[16x64]	1.88
绝对差值和_交流_p[1][16x64]	1.55	均值_差异_残差[1][4x8]	1.69	绝对差值和_样本_x3[32x32]	1.89
减去_像素[32x4]	1.55	绝对差值和_交流_像素[1][4x32]	1.69	计算_符号[32x32]	1.89
绝对差值和_交流_像素[0][4x32]	1.56	平均_差值_残差[0][4x8]	1.7	绝对差值和_ac_残差[0][4x16]	1.9
绝对差值和_ac_残差[1][16x8]	1.56	绝对差值和_ac_p[0][16x8]	1.7	copy_双向预测[4x2]	1.9
绝对差值和_ac_p[1][32x8]	1.56	calc_sign[16x4]	1.7	绝对差值和_sp[32x8]	1.91
绝对差值和_ac_p[1][64x32]	1.56	绝对差值和_ac_残差[1][16x64]	1.71	计算_符号[16x32]	1.92
绝对差值和_ac_p[0][16x4]	1.57	复制_双向预测[16x4]	1.72	计算_符号[32x8]	1.92
绝对差值和_累加_残差[0][32x16]	1.57	均值_差值_残差[0][4x2]	1.73	绝对差值和_累加_p[0][8x8]	1.94
绝对差值和_累加_p[1][32x16]	1.58	绝对差值和_累加_残差[1][4x4]	1.73	均值_差值_残差[0][8x16]	1.94
绝对差值和_AC_残差[1][32x64]	1.59	绝对差值和_AC_残差[0][4x32]	1.73	均值_差值_残差[0][4x16]	1.95
计算_符号[32x64]	1.59	绝对差值和_AC_p[0][4x8]	1.74	绝对差值和_sp_x3[8x4]	1.95
计算_符号[64x64]	1.59	绝对差值和_ac_p[1][8x16]	1.74	绝对差值和_ac_p[1][4x8]	1.97
绝对差值和_ac_p[0][8x64]	1.6	绝对差值和_sp[8x2]	1.75	减去_残差[4x2]	1.98
绝对差值和_ac_残差[0][16x16]	1.61	绝对差值和_ac_残差[0][16x64]	1.76	计算_符号[64x16]	1.98
绝对差值和_ac_p[1][64x16]	1.61	复制_双向预测[16x2]	1.77	减去_残差[16x64]	1.99
平均_差异_残差[1][8x4]	2	平均_差异_残差[0][4x4]	2.25	减去_残差[16x32]	2.44
平均_差异_p[0][8x8]	2	平均_差异_p[1][4x16]	2.25	块_零[64x16]	2.44
减去_残差[16x8]	2	均值_差异_残差[1][16x32]	2.25	绝对差值和_pp[64x32]	2.45
绝对差值和_sp_x3[64x64]	2.02	绝对差值和_sp_x3[32x16]	2.25	绝对差值和_sp[32x16]	2.46
绝对差值和_sp_x3[8x2]	2.03	均值_差值_p[1][32x4]	2.26	块_零[64x8]	2.46
均值_差值_p[1][16x64]	2.03	减去_残差[8x32]	2.26	均值_差值_残差[1][4x16]	2.47
均值_差值_残差[0][16x4]	2.06	绝对差值和_ac_p[1][4x16]	2.28	绝对差值和_sp_x3[32x4]	2.47
sub_权重[4x4]	2.06	均值_差值_残差[1][8x8]	2.28	色度_hps_12[5]	2.47
均值_差异_像[1][8素8]	2.07	均值_差异_残差[1][16x2]	2.28	绝对差值和_pp[64x16]	2.48
计算_符号[16x64]	2.07	减去_像素[16x4]	2.28	均值_差值_残差[0][8x64]	2.49

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
绝对差值和_ac_残差[0][4x8]	2.09	计算_符号[16x16]	2.28	绝对差值和_pp[4x16]	2.5
sub_权重[16x8]	2.09	绝对差值和_pp[16x2]	2.32	绝对差值和_pp[8x2]	2.5
绝对差值和_sp[32x64]	2.11	绝对差值和_pp[32x8]	2.33	均值_差异_p[0][32x4]	2.5
减去_残差[4x8]	2.12	子_权重[16x16]	2.33	减去_像素[16x32]	2.51
子_权重[16x64]	2.13	绝对差值和_sp_x3[8x8]	2.35	绝对差值和_sp[4x32]	2.52
subtract_残差[16x4]	2.15	subtract_像素[16x8]	2.37	绝对差值和_sp_x3[16x32]	2.52
subtract_残差[16x16]	2.15	绝对差值和_sp[8x4]	2.39	绝对差值和_sp[64x32]	2.52
平均值_差值_p[1][16x16]	2.16	绝对差值和_sp_x3[64x8]	2.39	绝对差值和_sp_x3[16x2]	2.53
绝对差值和_sp_x3[16x64]	2.16	减去_像素[16x16]	2.39	均值_差值_残差[0][16x64]	2.53
计算_符号[16x8]	2.16	率失真优化_符号_重新应用[4x4]	2.39	减去_像素[4x4]	2.53
绝对差值和_sp_x3[8x64]	2.17	计算_符号[8x4]	2.39	绝对差值和_sp[4x16]	2.54
均值_差异_p[0][4x16]	2.18	子_权重[8x8]	2.4	均值_差异_p[0][16x2]	2.54
均值_差异_p[0][8x2]	2.18	绝对差值和_pp[8x4]	2.42	绝对差值和_pp[64x8]	2.54
均值_差异_p[1][8x4]	2.19	绝对差值和_sp[16x2]	2.42	绝对差值和_sp_x3[64x16]	2.54
均值_差异_残差[1][8x16]	2.2	均值_差异_残差[0][32x4]	2.42	均值_差异_p[0][64x64]	2.54
均值_差异_残差[1][8x64]	2.2	绝对差值和_sp_x3[64x32]	2.42	复制_双向预测[16x32]	2.54
均值_差异_残差[0][16x2]	2.2	色度_垂直搜索_12[4x2]	2.42	色度_垂直后处理_10[4x2]	2.54
绝对差值和_sp_x3[8x16]	2.21	减法_像素[4x2]	2.43	绝对差值和_sp[4x8]	2.55
绝对差值和_sp[8x64]	2.23	sub_权重[16x32]	2.43	绝对差值和_ac_残差[1][4x16]	2.55
绝对差值和_sp_x3[8x32]	2.24	绝对差值和_ac_p[0][32x4]	2.44	均值_差异_残差[1][8x32]	2.55
复制_块[2]	2.55	绝对差值和_ac_残差[1][16x4]	2.7	亮度_hps_12[1]	2.83
绝对差值和_sp_x3[16x4]	2.56	均值_差异_残差[1][32x4]	2.7	绝对差值和_pp[4x32]	2.84
绝对差值和_pp[64x64]	2.56	色度_垂直预处理_10[4x2]	2.7	subtract_像素[8x8]	2.84
绝对差值和_sp[64x8]	2.57	subtract_残差[8x8]	2.7	绝对差值和_sp[32x4]	2.85
色度垂直搜索_10[4x2]	2.57	sub_权重[8x4]	2.7	亮度_水平预测_12[2]	2.87
减去_残差[4x4]	2.57	率失真优化_裁剪乘数[4x32]	2.7	平均_差异_p[1][64x32]	2.88
平均_差异_p[0][32x64]	2.59	绝对差值和_pp[8x64]	2.71	均值_差值_p[0][8x16]	2.89
均值_差值_残差[0][64x64]	2.59	均值_差值_p[0][64x8]	2.71	均值_差值_p[0][8x32]	2.89
复制_双向预测[16x16]	2.6	减去_残差[8x2]	2.72	绝对差值和_sp_x3[32x8]	2.89
绝对差值和_pp[4x4]	2.61	率失真优化_裁剪乘法[4x2]	2.72	色度_hps_12[1]	2.89
平均差值_p[1][64x16]	2.61	绝对差值和_pp[32x16]	2.73	率失真优化_裁剪乘数[4x4]	2.89
绝对差值和_sp[4x2]	2.62	色度_垂直搜索_10[4x16]	2.73	绝对差值和_sp_x3[16x16]	2.9
均值_差值_残差[0][4x32]	2.62	色度_hps_10[5]	2.74	平均值_差值_残差[1][16x16]	2.91
绝对差值和_sp[64x16]	2.62	绝对差值和_pp[32x4]	2.75	绝对差值和_sp[32x32]	2.91
平均值_差值_p[0][16x64]	2.63	亮度_hpp_10[0]	2.75	均值_差异_p[1][64x64]	2.91
均值_差异_残差[1][4x32]	2.64	减去_像素[4x32]	2.75	复制_双向预测[8x2]	2.92
绝对差值和_sp_x3[16x8]	2.64	均值_差异_p[0][4x8]	2.76	色度_hps_10[3]	2.92
绝对差值和_pp[8x16]	2.66	均值_差异_残差[1][32x16]	2.76	减去_残差[4x32]	2.92
加上_裁剪_10[8x2]	2.66	色度_hps_10[1]	2.76	均值_差分_p[1][4x32]	2.93
均值_差分_p[0][32x16]	2.67	绝对差值和_sp[4x4]	2.77	绝对差值和_sp[16x8]	2.94
减去_残差[16x2]	2.67	绝对差值和_pp[8x32]	2.77	色度_垂直处理_12[4x2]	2.94
绝对差值和_ac_残差[1][4x32]	2.68	绝对差值和_pp[4x2]	2.79	色度_垂直处理_12[8x2]	2.94
均值_差异_残差[0][8x4]	2.68	率失真优化_裁剪乘数[4x8]	2.8	绝对差值和_pp[16x8]	2.95
平均_差异_p[1][16x4]	2.68	平方差和_ss[4x2]	2.81	率失真优化_符号_重新应用[4x32]	2.95
率失真优化_符号_重新应用[4x2]	2.68	绝对差值和_pp[32x32]	2.81	率失真优化_裁剪乘法[16x2]	2.95
均值_差值_p[1][8x64]	2.69	加法_裁剪_10[4x2]	2.81	均值_差值_残差[1][32x8]	2.96
绝对差值和_sp[8x8]	2.7	亮度_hps_12[3]	2.82	平均_差异_残差[0][8x32]	2.97
平均_差异_残差[0][8x8]	2.7	子项_权重[16x4]	2.82	绝对差值和_sp[8x32]	2.98
绝对差值和_sp[8x16]	2.7	平均_差异_p[0][8x4]	2.83	复制_双向预测[4x4]	2.98
平均_差值_p[1][8x16]	2.7	平均_差值_p[1][16x8]	2.83	减去_像素[4x16]	2.98

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
色度_垂直处理_10[4x2]	2.99	均值_差异_残差[1][32x64]	3.16	均值_差值_残差[1][64x32]	3.32
绝对差值和_pp[8x8]	3	子_权重[4x16]	3.17	色度_垂直处理_12[8x4]	3.32
均值_差值_残差[1][16x64]	3	均值_差值_残差[0][32x64]	3.18	亮度_反量化_量化_10b[4x4]	3.32
色度_垂直搜索_10[8x8]	3	色度_垂直处理_10[8x2]	3.18	率失真优化_裁剪乘数[4x16]	3.32
子_权重[4x8]	3	率失真优化_符号_重新应用[4x8]	3.18	率失真优化_符号_重新应用[4x16]	3.32
均值_差异_残差[0][16x8]	3.01	率失真优化_裁剪乘数[8x32]	3.18	色度_垂直后处理_12[8x2]	3.33
均值_差异_p[1][32x8]	3.01	均值_差异_p[0][32x32]	3.19	计算_符号[8x2]	3.33
绝对差值和_sp[16x32]	3.02	亮度_逆_量化_12b[8x4]	3.19	平均_差_残差[0][64x8]	3.34
平均_差_p[0][16x8]	3.03	绝对差值和_sp[16x64]	3.2	减去_残差[8x64]	3.34
平均_差异_p[0][64x32]	3.03	平均_差异_p[0][32x8]	3.2	添加_裁剪_12[4x4]	3.35
复制_双向预测[16x64]	3.03	平均_差异_残差[0][32x32]	3.2	率失真优化_裁剪乘数[32x64]	3.35
率失真优化_非零数量_x2]	3.03	亮度_逆_量化_12b[4x4]	3.2	平均_差值_残差[0][32x8]	3.36
绝对差值和_sp[64x64]	3.04	率失真优化_裁剪乘数[8x2]	3.2	率失真优化_裁剪乘法[8x64]	3.36
减去_像素[16x2]	3.04	平均_差异_p[1][32x32]	3.22	色度_垂直搜索_10[8x4]	3.37
平均_差异_p[0][4x32]	3.05	平均_差异_p[1][32x64]	3.22	色度_垂直搜索_12[8x4]	3.37
绝对差值和_sp[16x4]	3.05	减去_像素[8x64]	3.22	SATD_sp[8x4]	3.38
绝对差值和_sp[16x16]	3.05	均值_差异_p[1][16x32]	3.24	色度_垂直预处理_10[4x4]	3.41
块_零[64x32]	3.05	均值_差异_p[1][64x8]	3.24	均值_差异_残差[1][64x16]	3.42
均值_差异_p[0][8x64]	3.06	色度_垂直搜索_12[4x4]	3.25	减去_像素[8x4]	3.42
色度_垂直后处理_12[4x2]	3.06	色度_垂直预处理_12[8x2]	3.25	角度_快速_复制[8x64]	3.42
率失真优化_裁剪乘法[16x4]	3.07	均值_差值_残差[1][64x8]	3.27	减去_像素[8x32]	3.43
减去_残差[8x4]	3.09	率失真优化_裁剪乘法[16x32]	3.27	率失真优化_符号_重新应用[32x32]	3.45
均值_差异_残差[0][16x16]	3.1	率失真优化_裁剪乘法[8x16]	3.28	色度_hps_12[4]	3.46
均值_差异_残差[0][32x16]	3.11	率失真优化_裁剪乘数[32x4]	3.28	亮度_逆_量化_10b[4x16]	3.46
绝对差值和_pp[16x32]	3.13	均值_差异_p[0][64x16]	3.29	色度_垂直后处理_12[4x4]	3.47
均值_差异_残差[0][16x32]	3.14	率失真优化_裁剪乘数[8x8]	3.29	减去_像素[8x2]	3.47
均值_差异_p[1][32x16]	3.14	均值_差异_残差[0][64x32]	3.31	sub_权重[4x32]	3.47
绝对差值和_pp[16x64]	3.15	亮度_hps_10[4]	3.31	率失真优化_sign_reapply[8x64]	3.47
mean_diff_残差[1][32x32]	3.15	subtract_残差[8x16]	3.31	率失真优化_裁剪乘法[16x8]	3.47
均值_差值_p[0][16x32]	3.16	均值_差值_p[1][8x32]	3.32	率失真优化_裁剪乘法[16x64]	3.47
复制_双向预测[4x8]	3.48	色度_垂直预处理_12[8x32]	3.65	绝对差值和_pp[16x4]	3.87
角度_快速_复制[8x32]	3.49	色度_垂直后处理_10[8x32]	3.65	绝对差值和_pp[32x64]	3.89
绝对差值和_pp[4x8]	3.5	添加_裁剪_12[4x8]	3.66	add_裁剪_10[8x4]	3.9
亮度_hps_12[4]	3.51	率失真优化_clipmul[32x16]	3.66	块_zero[32x16]	3.9
色度_hps_10[2]	3.52	率失真优化_clipmul[32x32]	3.68	平方差和_sp[4x2]	3.92
色度_垂直搜索_10[8x2]	3.52	块_零[64x64]	3.69	SATD_sp[32x16]	3.94
率失真优化_裁剪乘数[16x16]	3.52	亮度_逆_量化_10b[16x4]	3.69	亮度_hps_12[0]	3.94
色度_hps_12[3]	3.53	率失真优化_clipmul[32x8]	3.69	色度_垂直处理_12[4x8]	3.94
色度_垂直搜索_10[4x8]	3.53	SATD_pp[8x4]	3.7	SATD_SP[64x32]	3.98
色度_垂直搜索_12[8x2]	3.53	色度_垂直搜索_10[4x4]	3.7	复制_双向预测[4x16]	3.98
减法_权重[8x64]	3.53	加法_裁剪_10[4x8]	3.7	率失真优化_符号_重新应用[8x2]	3.98
色度_垂直后处理_10[8x4]	3.55	亮度_逆_量化_12b[4x8]	3.7	添加_平均值_10[8x2]	3.99
减去_残差[4x16]	3.55	SATD_sp[32x64]	3.71	色度_垂直处理_10[8x8]	3.99
子_权重[8x16]	3.56	色度_垂直预处理_10[8x4]	3.71	亮度_逆_量化_12b[8x32]	4
色度_垂直处理_12[4x4]	3.57	均值_差异_p[0][16x16]	3.72	添加_裁剪_12[4x16]	4.03
亮度_反_量化_12b[4x16]	3.57	色度_垂直处理_10[4x4]	3.73	色度_垂直处理_10[8x4]	4.05
添加_裁剪_10[4x4]	3.58	亮度_反_量化_10b[8x4]	3.73	加法_裁剪_10[8x64]	4.06
块_归零[32x8]	3.58	加法_裁剪_12[4x2]	3.75	加法_裁剪_12[8x64]	4.07
率失真优化_裁剪乘法[64x16]	3.6	减法_权重[8x32]	3.75	色度_垂直预处理_10[8x16]	4.08
色度_hps_12[2]	3.61	亮度_反量化_b[4x8]	3.76	SATD_sp[8x64]	4.09
均值_差异_残差[1][64x64]	3.62	色度_垂直后处理_12[4x8]	3.77	亮度_hpp_12[0]	4.1
复制_块[3]	3.62	色度_垂直预处理_10[8x2]	3.78	色度_垂直处理_10[4x8]	4.12
色度_垂直预处理_12[4x4]	3.63	SATD_pp[16x4]	3.8	色度_垂直处理_12[8x8]	4.12
率失真优化_符号_重新应用[8x4]	3.63	率失真优化_裁剪乘法[64x64]	3.8	色度_垂直后处理_12[8x8]	4.12
率失真优化_裁剪乘法[64x8]	3.63	计算_符号[8x32]	3.82	色度_垂直处理_12[8x16]	4.12
色度_水平处理_10[4]	3.64	绝对差值和_pp[16x16]	3.83	色度_垂直搜索_10[8x32]	4.12
减去_像素[8x16]	3.64	色度_垂直预处理_12[4x2]	3.84	率失真优化_数量_非零[8x4]	4.16
平均值_差异_残差[0][64x16]	3.65	平方差和_pp[4x16]	3.85	色度_垂直处理_10[8x16]	4.17
复制_双向预测[8x4]	3.65	亮度_hpp_10[3]	3.85	平方差和_pp[4x4]	4.18
亮度_hpp_10[1]	3.65	色度_垂直后处理_10[4x4]	3.85	色度_垂直搜索_12[8x8]	4.19

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
色度_垂直预处理_12[8x4]	4.2	亮度_逆_量化_10b[16x64]	4.56	率失真优化_符号_重新应用[16x64]	4.77
色度_垂直预处理_12[4x8]	4.23	色度_垂直后处理_10[8x2]	4.57	添加_平均值_10[8x16]	4.79
率失真优化_裁剪乘法[8x4]	4.24	色度_垂直后处理_10[8x16]	4.57	色度_垂直搜索_10[8x64]	4.79
色度_垂直搜索_12[4x8]	4.25	角度_快速_复制[16x64]	4.57	亮度_逆_量化_12b[16x64]	4.8
添加_裁剪_10[4x32]	4.25	亮度_逆_量化_10b[8x32]	4.57	色度_垂直预处理_12[4x32]	4.82
加法_平均值_12[8x2]	4.26	平方差和_sp[4x16]	4.58	SATD_sp[8x16]	4.84
加法_裁剪_12[4x32]	4.27	SATD_sp[8x8]	4.58	SATD_pp[8x32]	4.84
亮度_inv_quant_10b[16x8]	4.27	亮度_hpp_12[1]	4.58	亮度_hps_10[1]	4.84
平方差和_pp[8x2]	4.28	add_裁剪_12[8x32]	4.58	色度_垂直后处理_12[8x64]	4.84
色度_垂直搜索_12[8x16]	4.28	色度_垂直处理_10[16x8]	4.59	色度_垂直搜索_10[8x16]	4.85
色度_垂直预处理_12[8x8]	4.3	率失真优化_符号_重新应用[8x32]	4.59	色度_垂直处理_10[16x32]	4.85
率失真优化_clipmul[64x32]	4.31	SATD_pp[8x8]	4.6	add_裁剪_10[8x8]	4.86
色度_垂直搜索_12[4x16]	4.34	SATD_pp[8x16]	4.6	亮度_hpp_12[2]	4.87
复制_块[4]	4.34	色度_垂直搜索_10[16x64]	4.6	亮度_inv_quant_12b[4x32]	4.87
色度_垂直后处理_10[4x8]	4.35	添加_裁剪_12[8x2]	4.6	亮度_hpp_12[3]	4.88
平方差和_ss[4x16]	4.39	SATD_pp[32x8]	4.61	亮度_hps_10[3]	4.89
亮度_inv_quant_12b[16x4]	4.39	add_裁剪_10[8x32]	4.61	SATD_pp[32x16]	4.9
add_裁剪_12[8x4]	4.4	色度_垂直预处理_10[8x8]	4.63	色度_垂直预处理_12[8x16]	4.9
平方差和_pp[4x2]	4.42	亮度_inv_quant_10b[16x16]	4.63	复制_双向预测[4x32]	4.92
SATD_pp[64x64]	4.42	色度_垂直预处理_10[4x8]	4.65	平方差和_ss[8x2]	4.93
计算_符号[8x8]	4.43	平方差和_sp[4x4]	4.66	色度_垂直后处理_12[8x4]	4.93
SATD_pp[32x64]	4.44	亮度_垂直预处理_10[4x16]	4.67	色度_垂直后处理_10[32x8]	4.95
SATD_sp[64x64]	4.45	亮度_hps_10[0]	4.68	平方差和_sp[4x8]	4.96
色度_垂直后处理_10[8x8]	4.45	亮度_inv_quant_10b[8x8]	4.69	亮度_垂直处理_10[4x4]	4.97
add_裁剪_10[4x16]	4.46	色度_垂直处理_12[8x32]	4.72	SATD_pp[16x16]	4.99
亮度_垂直处理_12[16x8]	4.49	复制_块[5]	4.72	色度_垂直搜索_10[16x4]	4.99
SATD_pp[32x4]	4.5	角度_快速_复制[8x4]	4.72	块_零[32x64]	4.99
平方差和_ss[4x4]	4.52	率失真优化_符号_重新应用[32x64]	4.73	色度_垂直预处理_12[16x32]	5.01
亮度_逆_量化_12b[8x8]	4.52	色度_垂直预处理_10[16x2]	4.75	亮度_hps_10[2]	5.02
色度_垂直预处理_10[4x16]	4.53	亮度_垂直搜索_10[4x4]	4.75	亮度_垂直预处理_12[8x4]	5.02
色度_垂直搜索_12[8x32]	5.03	SATD_sp[64x8]	5.24	色度_垂直搜索_10[4x32]	5.45
亮度_垂直处理_12[4x4]	5.04	色度_垂直处理_12[32x64]	5.26	SATD_sp[8x32]	5.46
复制_样本_字节	5.05	加法_平均值_10[8x4]	5.27	添加_平均值_12[8x4]	5.46
角度_快速_复制[8x16]	5.06	亮度_反_量化_12b[16x8]	5.27	亮度_垂直处理_10[4x8]	5.46
亮度_反_量化_10b[4x32]	5.06	平方差和_pp[4x8]	5.29	块_零[32x32]	5.46
SATD_pp[16x8]	5.08	色度_垂直搜索_12[16x4]	5.29	添加_平均值_10[8x8]	5.47
亮度_hpp_12[4]	5.09	添加_平均值_10[8x32]	5.32	添加_裁剪_10[16x2]	5.48
亮度_反_量化_10b[8x64]	5.09	色度_垂直处理_10[8x32]	5.33	率失真优化_符号_重新应用[16x4]	5.48
亮度_反_量化_10b[64x16]	5.09	亮度_垂直后处理_12[4x4]	5.33	色度_垂直处理_12[4x16]	5.51
块_零[16x8]	5.1	添加_裁剪_10[8x16]	5.33	添加_裁剪_12[16x2]	5.51
SATD_sp[16x32]	5.11	添加_裁剪_12[8x16]	5.33	ang_快速_复制[64x16]	5.51
添加_裁剪_12[8x8]	5.11	亮度_逆_量化_12b[8x16]	5.34	率失真优化_符号_重新应用[64x16]	5.52
率失真优化_符号_重新应用[8x16]	5.11	色度_垂直搜索_10[16x8]	5.35	亮度_垂直搜索_10[16x4]	5.53
亮度_垂直搜索_12[16x8]	5.12	色度_垂直搜索_10[16x2]	5.37	亮度_逆_量化_10b[32x4]	5.53
ang_快速_复制[16x32]	5.13	SATD_pp[16x64]	5.39	亮度_逆_量化_12b[64x16]	5.53
SATD_pp[32x32]	5.14	色度_垂直处理_12[16x2]	5.39	色度_垂直预处理_12[8x64]	5.54
亮度_垂直处理_10[8x4]	5.15	复制_双向预测[8x8]	5.4	色度_垂直搜索_12[64x64]	5.55
SATD_sp[32x8]	5.16	色度_垂直处理_10[16x4]	5.4	亮度_垂直预处理_10[16x16]	5.55
亮度_逆_量化_10b[32x64]	5.16	亮度_逆_量化_10b[8x16]	5.4	率失真优化_数量_非零[8x8]	5.55
SATD_pp[64x8]	5.17	色度_垂直处理_10[4x16]	5.41	色度_垂直处理_10[16x64]	5.56
SATD_pp[64x16]	5.17	色度_垂直处理_10[16x2]	5.41	色度_垂直预处理_12[16x2]	5.57
色度_垂直搜索_12[16x2]	5.17	色度_垂直搜索_12[16x8]	5.41	亮度_垂直预处理_12[4x4]	5.57
亮度_hpp_10[2]	5.18	亮度_垂直预处理_10[4x4]	5.41	块_零[32x4]	5.57
色度_垂直后处理_10[8x64]	5.19	亮度_垂直后处理_10[8x64]	5.41	色度_垂直后处理_12[64x16]	5.58
色度_垂直后处理_12[8x16]	5.2	色度_垂直后处理_12[8x32]	5.42	色度_垂直处理_12[16x8]	5.59
SATD_sp[32x4]	5.21	色度_垂直预处理_10[8x32]	5.43	色度_垂直预处理_10[64x16]	5.59
亮度_垂直搜索_12[4x4]	5.21	色度_垂直处理_10[16x16]	5.43	亮度_垂直处理_12[4x8]	5.59
SATD_pp[8x64]	5.22	色度_垂直后处理_12[32x4]	5.43	亮度_垂直搜索_12[4x8]	5.6
SATD_sp[16x16]	5.22	色度_垂直搜索_12[8x64]	5.44	色度_垂直搜索_12[4x32]	5.62
色度_垂直预处理_12[16x8]	5.22	色度_垂直后处理_12[16x2]	5.44	色度_垂直后处理_12[64x64]	5.62

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
satd_sp[64x16]	5.63	ang_fast_copy[32x64]	5.81	色度_垂直后处理_12[16x8]	5.99
色度_垂直处理_12[64x16]	5.63	SATD_sp[16x8]	5.83	色度_垂直预处理_12[64x64]	5.99
ang_fast_copy[32x8]	5.64	亮度_inv_quant_12b[32x32]	5.83	亮度_垂直后处理_10[8x4]	5.99
平方差和_pp[8x4]	5.65	平方差和_ss[4x8]	5.84	calc_sign[8x16]	5.99
色度_垂直预处理_12[4x16]	5.65	亮度_垂直处理_12[8x8]	5.84	亮度_垂直后处理_12[16x32]	6
亮度_逆_量化_12b[32x16]	5.65	亮度_垂直后处理_12[8x32]	5.84	亮度_逆_量化_12b[32x4]	6
亮度_逆_量化_12b[64x32]	5.66	率失真优化_符号_重新应用[32x8]	5.84	计算_符号[8x64]	6
率失真优化_符号_重新应用[16x8]	5.66	率失真优化_符号_重新应用[32x16]	5.85	色度_垂直预处理_12[16x64]	6.01
平方差和_sp[8x2]	5.67	色度_垂直处理_12[8x64]	5.86	率失真优化_符号_重新应用[16x32]	6.01
亮度_反向_量化_12b[8x64]	5.67	色度_垂直预处理_12[32x8]	5.86	色度_垂直后处理_12[4x16]	6.02
亮度_反向_量化_10b[16x32]	5.67	色度_垂直后处理_12[32x8]	5.86	色度_垂直预处理_10[16x4]	6.02
色度_垂直后处理_12[32x64]	5.68	亮度_垂直预处理_10[8x4]	5.86	色度_垂直预处理_12[16x4]	6.02
SATD_sp[32x32]	5.69	亮度_垂直后处理_12[8x4]	5.86	色度_垂直后处理_12[16x64]	6.02
添加_平均值_10[8x64]	5.69	率失真优化_符号_重新应用[8x8]	5.86	平方差和_pp[4x32]	6.03
色度_垂直处理_10[8x64]	5.69	平方差和_ss[4x32]	5.87	色度_垂直处理_12[32x4]	6.03
亮度_垂直预处理_10[8x8]	5.7	亮度_垂直预处理_10[4x8]	5.88	率失真优化_符号_重新应用[16x16]	6.03
亮度_垂直后处理_10[8x8]	5.7	亮度_逆_量化_10b[32x32]	5.89	色度_垂直处理_10[4x32]	6.04
亮度_垂直预处理_10[16x32]	5.7	色度_垂直后处理_10[32x16]	5.9	色度_垂直搜索_10[32x4]	6.04
亮度_垂直处理_12[8x4]	5.71	亮度_垂直预处理_12[4x8]	5.9	亮度_垂直后处理_12[16x64]	6.04
角度_快速复制_x[32	5.71	亮度_垂直搜索_10[4x8]	5.9	复制_双向预测[8x16]	6.05
率失真优化_符号_重新应用[64x64]	5.71	亮度_垂直搜索_10[8x4]	5.91	色度_垂直处理_10[32x4]	6.05
平方差和_ss[8x4]	5.73	率失真优化_num_非零[16x4]	5.91	亮度_垂直后处理_10[4x4]	6.05
色度_垂直后处理_10[16x2]	5.73	SATD_sp[16x64]	5.92	亮度_逆_量化_10b[64x8]	6.05
角度_快速_复制[16x4]	5.74	亮度_垂直处理_12[32x16]	5.92	添加_裁剪_12[16x4]	6.07
亮度_逆_量化_12b[16x32]	5.74	亮度_逆_量化_10b[64x32]	5.92	色度_垂直预处理_12[32x32]	6.08
亮度_水平预处理_10[4]	5.76	亮度_垂直搜索_12[8x4]	5.94	色度_垂直搜索_10[64x8]	6.08
率失真优化_符号_重新应用[16x2]	5.77	率失真优化_数量_非零[16x2]	5.94	SATD_垂直后处理[16x4]	6.09
亮度_垂直搜索_10[8x8]	5.78	亮度_反量化_b[16x16]	5.95	亮度_垂直搜索_12[8x8]	6.1
色度_垂直后处理_10[32x4]	5.8	色度_垂直预处理_12[64x32]	5.96	添加_裁剪_10[16x8]	6.1
亮度_逆_量化_10b[64x64]	5.8	角度_快速_复制[16x2]	5.96	亮度_垂直预处理_12[16x4]	6.11
块_清零[4x2]	6.11	亮度_垂直处理_10[64x32]	6.32	色度_垂直处理_12[16x32]	6.57
色度_垂直后处理_10[64x32]	6.12	色度_垂直后处理_10[16x16]	6.33	色度_垂直预处理_10[16x16]	6.58
色度_垂直搜索_12[32x8]	6.13	平方差和_sp[4x32]	6.34	色度_垂直处理_10[64x64]	6.58
色度_垂直处理_12[4x32]	6.14	亮度_垂直后处理_10[32x64]	6.34	添加_平均值_12[8x32]	6.6
亮度_垂直处理_10[8x64]	6.14	角度_快速_复制[64x64]	6.35	色度_垂直搜索_10[64x64]	6.61
色度_垂直后处理_10[4x16]	6.15	亮度_垂直后处理_10[4x8]	6.38	亮度_垂直处理_10[16x32]	6.61
色度_垂直预处理_10[16x64]	6.15	亮度_垂直预处理_10[32x64]	6.38	亮度_垂直搜索_12[64x64]	6.61
添加_平均值_12[8x8]	6.17	亮度_垂直预处理_10[64x32]	6.38	色度_垂直搜索_12[32x4]	6.62
添加_平均值_12[16x2]	6.17	色度_垂直后处理_12[16x16]	6.39	亮度_垂直预处理_12[64x8]	6.63
角度_快速_复制[64x32]	6.17	亮度_垂直预处理_12[8x8]	6.39	添加_平均值_12[8x16]	6.64
色度_垂直后处理_10[16x8]	6.18	亮度_垂直后处理_10[8x16]	6.41	add_裁剪_10[16x64]	6.64
角度_快速_复制[32x32]	6.19	亮度_逆_量化_12b[64x8]	6.41	亮度_垂直预处理_10[8x64]	6.7
add_裁剪_12[32x64]	6.19	率失真优化_符号_重新应用[32x4]	6.41	色度_垂直处理_10[32x32]	6.71
亮度_垂直处理_12[64x32]	6.2	色度_垂直搜索_10[16x16]	6.43	亮度_垂直后处理_12[4x32]	6.71
亮度_反量化_b[32x8]	6.2	添加_裁剪_10[16x4]	6.43	色度_垂直处理_12[16x16]	6.72
率失真优化_符号_重新应用[64x8]	6.2	亮度_垂直预处理_10[4x32]	6.45	色度_垂直搜索_10[32x8]	6.72
亮度_垂直搜索_10[4x16]	6.23	亮度_垂直后处理_12[4x8]	6.46	率失真优化_数量_非零[16x8]	6.73
亮度_垂直处理_12[16x4]	6.23	亮度_垂直处理_12[8x16]	6.46	亮度_垂直后处理_10[64x16]	6.76
亮度_垂直后处理_12[64x8]	6.26	色度_垂直搜索_12[32x64]	6.47	率失真优化_数量_非零[32x4]	6.77
色度_垂直后处理_10[16x4]	6.27	亮度_垂直后处理_10[32x32]	6.47	亮度_垂直预处理_10[8x16]	6.78
色度_垂直预处理_12[32x4]	6.27	色度_垂直后处理_10[4x32]	6.48	亮度_垂直预处理_10[64x16]	6.78
色度_垂直处理_10[32x8]	6.27	色度_垂直后处理_10[32x32]	6.48	复制_双向预测[8x64]	6.79
亮度_垂直后处理_12[8x8]	6.27	SATD_pp[64x32]	6.49	色度_垂直预处理_12[16x16]	6.79
亮度_垂直处理_10[8x8]	6.28	亮度_垂直预处理_12[8x32]	6.49	亮度_垂直后处理_12[8x64]	6.79
亮度_垂直后处理_10[16x32]	6.28	添加_平均值_12[8x64]	6.52	亮度_垂直搜索_12[32x32]	6.79
色度_垂直搜索_12[32x32]	6.29	亮度_逆_量化_10b[32x16]	6.53	亮度_垂直预处理_12[64x32]	6.79
块_零[16x4]	6.29	角度_快速_复制[16x16]	6.54	平方差和_ss[16x8]	6.8
色度_垂直后处理_12[4x32]	6.3	色度_垂直处理_12[32x16]	6.55	亮度_垂直后处理_12[16x4]	6.81
亮度_inv_quant_12b[32x8]	6.3	色度_垂直预处理_10[32x32]	6.56	色度_垂直处理_10[64x16]	6.84
add_裁剪_12[32x32]	6.31	亮度_垂直后处理_10[64x32]	6.56	平方差和_ss[16 x4]	6.85

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
添加_裁剪_12[16x32]	6.85	亮度_垂直处理_12[4x32]	7.03	平方差和_pp[32x4]	7.27
add_平均值_12[16x8]	6.86	亮度_垂直后处理_12[64x32]	7.03	亮度_垂直搜索_10[8x16]	7.27
色度_垂直后处理_10[64x8]	6.86	色度_垂直预处理_10[16x32]	7.05	角度_快速_复制[16x8]	7.27
角度_快速_复制[64x8]	6.86	亮度_垂直处理_10[4x16]	7.05	色度_垂直处理_10[64x8]	7.29
SATD_pp[16x32]	6.87	色度_垂直预处理_10[8x64]	7.07	复制_双向预测[8x32]	7.32
色度_垂直预处理_10[32x16]	6.87	亮度_垂直后处理_10[16x8]	7.07	亮度_垂直预处理_12[4x32]	7.32
色度_垂直预处理_10[32x8]	6.88	色度_垂直后处理_10[64x16]	7.08	add_裁剪_10[16x16]	7.33
亮度_垂直搜索_12[16x4]	6.88	色度_垂直后处理_10[16x32]	7.11	色度_垂直处理_10[32x16]	7.34
亮度_垂直处理_10[32x8]	6.88	色度_垂直搜索_12[16x32]	7.13	添加_平均值_10[32x64]	7.36
平方差和_pp[16x2]	6.89	色度_垂直搜索_12[32x16]	7.13	色度_垂直预处理_10[32x4]	7.36
色度_垂直处理_12[64x8]	6.89	色度_垂直搜索_12[16x64]	7.14	亮度_垂直处理_10[16x4]	7.36
亮度_垂直搜索_10[32x4]	6.9	添加_裁剪_10[64x64]	7.14	亮度_垂直预处理_10[16x8]	7.36
色度_垂直处理_12[32x8]	6.91	亮度_反量化_b[32x64]	7.14	亮度_垂直预处理_10[32x32]	7.36
亮度_垂直处理_10[4x32]	6.92	亮度_垂直处理_12[64x16]	7.15	添加_平均值_12[32x64]	7.37
色度_垂直处理_12[32x32]	6.93	添加_裁剪_12[16x8]	7.15	色度_垂直预处理_10[64x32]	7.37
亮度_垂直搜索_12[8x16]	6.93	色度_垂直后处理_12[16x32]	7.16	添加_平均值_12[16x4]	7.38
率失真优化_符号_重新应用[64x32]	6.93	色度_垂直搜索_10[32x32]	7.16	亮度_垂直搜索_10[8x32]	7.38
色度_垂直预处理_12[32x16]	6.94	色度_垂直搜索_12[64x16]	7.16	色度_垂直后处理_12[32x32]	7.39
亮度_垂直处理_12[4x16]	6.95	亮度_垂直预处理_12[4x16]	7.16	亮度_垂直搜索_12[4x16]	7.39
亮度_垂直搜索_10[8x64]	6.96	亮度_垂直搜索_10[16x8]	7.17	亮度_垂直搜索_10[16x64]	7.39
亮度_垂直后处理_10[16x4]	6.96	亮度_垂直搜索_12[16x64]	7.17	色度_垂直搜索_10[16x32]	7.42
添加_裁剪_10[16x32]	6.96	添加_裁剪_12[16x64]	7.17	色度_垂直预处理_12[64x8]	7.42
亮度_垂直处理_12[64x64]	6.99	亮度_垂直处理_10[8x16]	7.18	亮度_垂直后处理_12[16x8]	7.42
角度_快速_复制[32x16]	6.99	亮度_垂直预处理_12[8x16]	7.19	亮度_垂直搜索_10[16x32]	7.44
色度_垂直搜索_10[32x64]	7	添加_平均值_10[16x2]	7.2	亮度_垂直搜索_10[64x32]	7.44
率失真优化_数量_非零[8x16]	7	亮度_垂直后处理_12[8x16]	7.2	色度_垂直预处理_10[32x64]	7.45
添加_平均值_10[16x32]	7.01	亮度_垂直处理_10[16x16]	7.24	色度_垂直预处理_10[64x8]	7.45
色度_垂直搜索_12[64x8]	7.01	平方差和_sp[8x8]	7.25	亮度_垂直后处理_10[8x32]	7.45
亮度_垂直预处理_10[32x16]	7.01	色度_垂直后处理_12[32x16]	7.25	亮度_垂直处理_10[16x8]	7.46
亮度_垂直搜索_10[32x16]	7.01	亮度_垂直预处理_12[16x8]	7.26	亮度_垂直处理_10[64x16]	7.47
平方差和_sp[8x4]	7.48	亮度_垂直处理_12[16x16]	7.77	亮度_垂直处理_12[64x8]	8.09
添加_平均值_10[16x4]	7.48	平方差和_ss[8x16]	7.79	色度_垂直后处理_10[16x64]	8.1
亮度_垂直预处理_10[16x4]	7.48	添加_平均值_12[32x4]	7.79	亮度_垂直搜索_12[32x4]	8.1
色度_垂直处理_12[16x64]	7.49	色度_垂直预处理_12[64x16]	7.79	添加_裁剪_12[16x16]	8.1
亮度_垂直处理_12[8x32]	7.5	添加_裁剪_12[32x4]	7.79	亮度_垂直搜索_12[64x16]	8.16
亮度_垂直处理_12[16x32]	7.51	平方差和_sp[16x8]	7.8	亮度_垂直搜索_10[32x8]	8.17
色度_垂直处理_10[64x32]	7.53	亮度_垂直搜索_12[8x32]	7.84	添加_平均值_10[16x16]	8.18
平方差和_sp[16x2]	7.54	亮度_垂直预处理_12[32x4]	7.85	添加_平均值_10[64x64]	8.18
添加_裁剪_10[32x64]	7.54	平方差和_ss[32x4]	7.87	亮度_垂直处理_12[32x8]	8.18
亮度_垂直处理_12[16x64]	7.57	亮度_垂直后处理_10[16x16]	7.87	亮度_垂直后处理_10[64x8]	8.21
亮度_垂直处理_10[8x32]	7.58	添加_裁剪_10[32x16]	7.88	亮度_垂直后处理_12[32x64]	8.24
色度_垂直搜索_10[64x32]	7.59	率失真优化_数量_非零[8x32]	7.88	率失真优化_数量_非零[8x64]	8.25
色度_垂直后处理_12[64x8]	7.62	色度_垂直搜索_10[32x16]	7.9	色度_垂直预处理_10[4x32]	8.26
亮度_垂直处理_10[32x4]	7.62	添加_平均值_12[16x64]	7.91	亮度_垂直后处理_12[32x4]	8.28
亮度_垂直后处理_12[32x8]	7.64	色度_垂直后处理_12[16x4]	7.93	平方差和_pp[16x4]	8.3
平方差和_sp[32x4]	7.65	色度_垂直预处理_10[16x8]	7.93	平方差和_sp[32x8]	8.3
亮度_inv_quant_12b[64x64]	7.65	add_平均值_10[16x8]	7.96	平方差和_ss[16x2]	8.33
率失真优化_数量_非零[32x8]	7.66	色度_垂直预处理_12[32x64]	7.96	添加_平均值_12[16x32]	8.33
亮度_垂直搜索_12[4x32]	7.68	色度_垂直搜索_10[64x16]	8	亮度_垂直搜索_10[4x32]	8.35
亮度_垂直搜索_12[32x8]	7.68	色度_垂直处理_12[64x32]	8	亮度_垂直预处理_12[32x8]	8.37
亮度_垂直后处理_12[32x16]	7.68	色度_垂直后处理_10[32x64]	8.01	平方差和_pp[16x16]	8.38
亮度_垂直预处理_12[64x16]	7.68	add_平均值_10[16x64]	8.02	亮度_垂直后处理_12[32x32]	8.38
平方差和_sp[16x4]	7.69	add_平均值_10[32x16]	8.03	亮度_垂直处理_10[32x32]	8.41
亮度_垂直搜索_12[8x64]	7.69	亮度_垂直后处理_10[32x4]	8.03	亮度_垂直搜索_12[16x32]	8.42
亮度_垂直预处理_12[16x64]	7.69	添加_裁剪_10[32x8]	8.05	亮度_垂直搜索_10[64x64]	8.44
平方差和_ss[8x8]	7.7	率失真优化_num_非零[16x16]	8.05	add_平均值_12[64x16]	8.45
亮度_垂直后处理_12[4x16]	7.7	亮度_垂直预处理_12[16x16]	8.06	亮度_垂直处理_10[64x8]	8.46
亮度_垂直预处理_12[16x32]	7.74	亮度_垂直预处理_12[32x64]	8.06	色度_垂直搜索_12[16x16]	8.47
亮度_垂直后处理_10[32x16]	7.76	亮度_垂直预处理_10[32x4]	8.08	添加_裁剪_12[32x8]	8.48
添加_裁剪_10[32x32]	7.76	亮度_垂直预处理_10[8x32]	8.09	平方差和_pp[16 x8]	8.49

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
色度_垂直搜索_12[64x32]	8.49	亮度_垂直搜索_12[32x64]	8.74	率失真优化_数量_非零[32x32]	9.62
色度_垂直处理_12[64x64]	8.49	添加_裁剪_12[64x16]	8.77	角度_快速_复制[8x8]	9.65
亮度_垂直处理_10[32x64]	8.49	亮度_垂直搜索_12[32x16]	8.83	add_裁剪_12[64x8]	9.7
块_zero[16x32]	8.5	add_平均值_12[32x16]	8.89	亮度_垂直搜索_10[32x32]	9.71
亮度_垂直预处理_10[32x8]	8.51	亮度_垂直预处理_10[64x8]	8.93	添加_裁剪_10[64x8]	9.75
色度_垂直预处理_10[64x64]	8.52	亮度_垂直预处理_12[32x32]	8.98	率失真优化_数量_非零[64x32]	9.77
平方差和_pp[32x8]	8.53	亮度_垂直搜索_12[64x8]	9.02	块_零[8x2]	9.86
平方差和_sp[8x16]	8.55	率失真优化_num_非零[16x64]	9.02	add_裁剪_12[64x64]	9.88
平方差和_ss[32x8]	8.55	亮度_垂直后处理_10[16x64]	9.03	加_平均值_12[32x8]	9.9
块_零[16x16]	8.55	亮度_垂直搜索_10[16x16]	9.09	加_平均值_12[64x32]	9.9
加_裁剪_10[32x4]	8.55	亮度_垂直预处理_12[32x16]	9.09	add_平均值_10[32x4]	10.06
平方差和_ss[8x32]	8.59	add_平均值_10[32x8]	9.12	块_zero[16x2]	10.39
亮度_垂直搜索_10[64x16]	8.59	亮度_垂直预处理_12[8x64]	9.12	加_平均值_12[64x8]	10.43
色度_垂直后处理_12[64x32]	8.6	加_平均值_10[32x32]	9.13	亮度_垂直处理_10[64x64]	10.45
亮度_垂直后处理_10[4x32]	8.61	色度_垂直处理_12[16x4]	9.14	亮度_垂直处理_10[16x64]	10.48
块_零[8x8]	8.62	亮度_垂直后处理_10[4x16]	9.2	加法_平均值_10[64x16]	10.51
块_零[16x64]	8.62	亮度_垂直后处理_12[16x16]	9.2	率失真优化_数量_非零[64x16]	10.56
色度_垂直处理_10[32x64]	8.63	亮度_垂直后处理_12[64x16]	9.2	率失真优化_数量_非零[32x64]	10.58
添加_平均值_12[16x16]	8.65	亮度_垂直搜索_12[64x32]	9.24	平方差和_垂直后处理[32x16]	10.65
平方差和_ss[16x16]	8.66	亮度_垂直搜索_10[64x8]	9.26	亮度_垂直后处理_12[64x64]	10.73
平方差和_pp[32x16]	8.67	平方差和_pp[8x16]	9.28	add_平均值_12[32x32]	10.82
率失真优化_数量_非零[16x32]	8.67	率失真优化_数量_非零[32x16]	9.4	add_平均值_10[64x8]	10.87
亮度_垂直搜索_12[16x16]	8.68	add_裁剪_10[64x16]	9.42	亮度_垂直处理_10[32x16]	10.99
平方差和_pp[8x8]	8.69	平方差和_sp[16x16]	9.46	添加_平均值_10[64x32]	11.13
添加_裁剪_12[32x16]	8.69	亮度_垂直预处理_10[16x64]	9.46	亮度_垂直预处理_12[64x64]	11.13
添加_裁剪_10[64x32]	8.69	添加_平均值_12[64x64]	9.5	块_零[8x16]	11.16
平方差和_sp[8x32]	8.71	亮度_垂直处理_12[32x4]	9.51	亮度_垂直后处理_10[64x64]	11.22
亮度_垂直搜索_10[32x64]	8.72	亮度_垂直处理_12[32x64]	9.51	块_零[8x32]	11.39
平方差和_pp[8x32]	8.73	色度_垂直后处理_10[64x64]	9.6	亮度_垂直处理_12[8x64]	11.44
亮度_垂直预处理_10[64x64]	8.73	率失真优化_数量_非零[64x8]	9.6	亮度_垂直处理_12[32x32]	11.46
亮度_垂直后处理_10[32x8]	11.6	块_零[4x4]	15.31	块_零[4x32]	29.09
块_零[8x64]	11.85	块_零[8x4]	20.03	块_零[4x8]	39.21
添加_裁剪_12[64x32]	12.56	块_零[4x16]	24.86		

附录 C

AVX2 SIMD 优化

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
目标4[2]	1.02	绝对差值和_ac_p[0][8x4]	1.75	绝对差值和_ac_残差[0][64x32]	1.96
目标4[1]	1.04	绝对差值和_ac_p[0][16x4]	1.77	减去_残差[32x64]	1.97
绝对差值和_ac_p[1][8x2]	1.22	绝对差值和_交流_残差[1][64x16]	1.77	绝对差值和_交流_残差[1][32x32]	1.99
均值_差值_残差[1][8x2]	1.24	绝对差值和_交流_p[0][32x4]	1.79	绝对差值和_交流_残差[1][8x16]	2
绝对差值和_交流_残差[1][16x2]	1.26	复制_双向预测[32x4]	1.79	绝对差值和_ac_残差[1][64x64]	2
绝对差值和_ac_残差[0][8x4]	1.35	减法_残差[32x16]	1.79	减法_像素[32x8]	2
绝对差值和_ac_p[0][8x2]	1.4	绝对差值和_交流_残差[1][64x8]	1.81	减法_权重[32x16]	2
复制_双向预测[16x2]	1.44	减去_残差[32x4]	1.81	绝对差值和_交流_p[1][16x8]	2.02
绝对差值和_交流_残差[1][8x2]	1.48	绝对差值和_绝对差值和_绝对差值和[0][64 绝对差值和8]	1.83	绝对差值和_绝对差值和_绝对差值和[0][8 绝对差值和16]	2.03
绝对差值和_绝对差值和_绝对差值和[1][16 绝对差值和2]	1.49	绝对差值和_绝对差值和_绝对差值和 [1][8绝对差值和4]	1.85	绝对差值和_绝对差值和_绝对差值和[0][32绝 对差值和32]	2.04
均值_差值_p[1][8x2]	1.53	绝对差值和_交流_p[1][32x8]	1.85	绝对差值和_交流_p[1][32x16]	2.07
均值_差异_残差[1][16x2]	1.56	绝对差值和_交流_残差[0][16x4]	1.87	减去_像素[64x8]	2.07
减去_像素[32x16]	1.57	绝对差值和_ac_残差[1][32x16]	1.87	绝对差值和_ac_残差[0][32x64]	2.08
减去_像素[64x16]	1.58	复制_双向预测[32x32]	1.87	减去_残差[32x32]	2.08
绝对差值和_ac_p[1][32x4]	1.59	减去_像素[32x32]	1.87	子权重_x[32	2.09
绝对差值和_交流_残差 [1][8x8]	1.62	减去_残差[64x8]	1.88	绝对差值和_交流_p[0][64 x16]	2.11

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
绝对差值和_绝对差值和_p[0][16x2]	1.62	绝对差值和_ac_残差[0][8x2]	1.91	绝对差值和_ac_p[1][32x64]	2.12
绝对差值和_ac_p[1][16x4]	1.62	绝对差值和_ac_残差[1][8x4]	1.91	减去_残差[32x8]	2.12
绝对差值和_ac_p[0][32x64]	1.62	绝对差值和_ac_p[0][16x8]	1.91	绝对差值和_交流_残差[1][32x64]	2.13
复制_双向预测[64x64]	1.62	绝对差值和_交流_p[1][64x8]	1.91	减法_权重[64x32]	2.13
减去_像素[32x4]	1.62	绝对差值和_交流_残差[1][64x32]	1.91	绝对差值和_交流_残差[1][32x8]	2.14
复制_双向预测[32x16]	1.63	减法_像素[32x64]	1.91	减法_权重[64x16]	2.14
绝对差值和_交流_p[1][64x64]	1.65	绝对差值和_交流_残差[1][16x16]	1.92	绝对差值和_交流_残差[0][32x8]	2.15
子_权重[32x4]	1.67	绝对差值和_交流_残差[0][32x4]	1.92	绝对差值和_交流_残差[0][8x32]	2.17
绝对差值和_交流_残差[0][16x2]	1.68	绝对差值和_交流_残差[0][64x64]	1.92	绝对差值和_交流_残差[1][16x32]	2.17
绝对差值和_交流_p[0][64x32]	1.69	绝对差值和_交流_p[1][8x32]	1.93	绝对差值和_交流_p[0][32x16]	2.17
绝对差值和_交流_p[0][64x64]	1.72	绝对差值和_交流_残差[0][8x16]	1.94	绝对差值和_交流_残差[1][32x4]	2.18
均值_差值_p[1][16x2]	1.74	绝对差值和_交流_残差[0][16x64]	1.94	绝对差值和_交流_残差[0][64x8]	2.18
绝对差值和_交流_残差[1][16x4]	1.74	复制_双向预测[16x4]	1.95	subtract_像素[64x64]	2.18
copy_双向预测[64x16]	1.74	绝对差值和_ac_p[1][64x16]	1.96	绝对差值和_ac_p[0][8x32]	2.19
绝对差值和_ac_p[1][64x32]	2.19	绝对差值和_ac_p[0][8x64]	2.48	subtract_残差[16x4]	2.96
sub_权重[32x64]	2.19	绝对差值和_ac_p[0][16x64]	2.49	绝对差值和_ac_p[0][32x8]	2.98
绝对差值和_ac_p[1][8x16]	2.21	绝对差值和_ac_p[0][16x16]	2.5	计算_符号[16x2]	2.99
绝对差值和_交流_残差[0][16x16]	2.21	减去_权重[16x4]	2.5	均值_差异_残差[1][16x32]	3
绝对差值和_交流_残差[0][32x16]	2.21	减去_像素[16x2]	2.52	sub_权重[16x16]	3
subtract_像素[64x32]	2.21	mean_diff_p[1][8x4]	2.53	mean_diff_残差[0][16x2]	3.03
sub_权重[64x8]	2.21	subtract_像素[16x4]	2.53	计算_符号[32x32]	3.06
减去_残差[64x64]	2.22	平均_差值_p[0][8x2]	2.54	平均_差值_残差[1][8x16]	3.07
绝对差值和_pp[16x2]	2.23	绝对差值和_ac_p[1][8x64]	2.57	均值_差异_残差[0][8x8]	3.08
绝对差值和_交流_残差[0][32x32]	2.23	均值_差异_残差[1][16x4]	2.57	均值_差异_p[1][8x8]	3.11
绝对差值和_交流_残差[1][16x8]	2.25	绝对差值和_交流_残差[0][16x8]	2.59	计算_符号[32x8]	3.11
绝对差值和_交流_点[1][32x32]	2.26	绝对差值和_交流_点[1][16x32]	2.59	结构 平方差和块 s 4_ _ _ _	3.12
绝对差值和_交流_残差[0][8x8]	2.27	复制_双向预测[32x8]	2.61	绝对差值和_sp_x3[16x8]	3.14
sub_权重[32x32]	2.31	sub_权重[64x64]	2.64	calc_sign[16x4]	3.14
copy_双向预测[64x8]	2.33	绝对差值和_ac_残差[1][16x64]	2.65	复制_双向预测[16x16]	3.17
减去_残差[16x2]	2.34	结构体_平方差和_块_p_4	2.65	均值_差异_p[0][16x2]	3.2
减去_残差[64x32]	2.34	绝对差值和_ac_p[0][8x8]	2.66	绝对差值和_sp_x3[64x32]	3.2
均值_差异_残差[0][8x2]	2.37	均值_差异_残差[1][8x4]	2.68	绝对差值和_sp_x4[16x4]	3.21
绝对差值和_ac_残差[1][8x32]	2.37	绝对差值和_sp_x4[64x64]	2.69	平均_差值_残差[0][16x4]	3.21
减去_残差[64x16]	2.37	平均_差值_残差[1][8x8]	2.71	绝对差值和_pp[16x4]	3.23
绝对差值和_sp_x4[16x2]	2.38	复制_双向预测[32x64]	2.72	减去_像素[16x64]	3.24
绝对差值和_ac_p[1][16x16]	2.38	绝对差值和_ac_残差[1][8x64]	2.74	减去_残差[16x64]	3.26
绝对差值和_ac_残差[0][64x16]	2.38	绝对差值和_sp_x4[32x64]	2.75	计算_符号[64x16]	3.3
复制_双向预测[64x32]	2.38	均值_差异_p[0][8x4]	2.76	均值_差异_p[1][16x4]	3.34
减去_像素[16x8]	2.39	计算_符号[32x16]	2.77	绝对差值和_空间_x4[32x4]	3.41
绝对差值和_空间_x3[16x2]	2.4	均值_差异_残差[0][8x4]	2.79	均值_差异_p[1][8x16]	3.42
绝对差值和_交流_p[1][8x8]	2.41	均值_差异_p[0][8x8]	2.8	绝对差值和_sp[16x4]	3.45
绝对差值和_ac_残差[0][16x32]	2.41	复制_双向预测[16x8]	2.8	均值_差异_残差[1][16x8]	3.45
绝对差值和_ac_p[0][16x32]	2.44	绝对差值和_ac_p[1][16x64]	2.83	均值_差值_点[0][16乘8]	3.47
绝对差值和_交流_残差[0][8乘64]	2.45	计算_符号[32乘4]	2.83	绝对差值和_稀疏_乘3[32乘4]	3.47

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
绝对差值和_sp_x3[32x64]	3.48	亮度_逆_量化_10b[8x4]	3.84	均值_差分_p[0][8x16]	4.19
绝对差值和_sp_x4[32x16]	3.52	均值_差分_残差[0][8x32]	3.87	减去_残差[16x32]	4.19
子_权重[16x32]	3.53	绝对差值和_sp_x4[64x8]	3.88	均值_差异_残差[1][64x16]	4.2
均值_差异_p[1][16x16]	3.55	块_零[64x8]	3.9	均值_差异_p[0][8x32]	4.22
子_权重[16x64]	3.55	均值_差异_残差[1][8x32]	3.92	复制_双向预测[16x64]	4.23
绝对差值和_sp_x4[16x8]	3.56	均值_差异_残差[0][32x8]	3.92	绝对差值和_sp_x3[16x16]	4.25
均值_差异_p[0][64x16]	3.57	率失真优化_裁剪乘数[8x4]	3.93	块_零[64x64]	4.29
绝对差值和_sp_x4[32x8]	3.58	计算_符号[64x64]	3.94	平均值_差值_p[0][8x64]	4.3
减去_像素[16x16]	3.59	子_权重[16x8]	3.97	均值_差异_像素[1][64x32]	4.31
绝对差值和_sp_x3[16x4]	3.6	角度_快速_复制[16x2]	3.98	均值_差异_残差[0][16x64]	4.34
减去_像素[16x32]	3.61	率失真优化_裁剪乘法[8x64]	3.99	均值_差值_残差[0][16x8]	4.35
计算_符号[32x64]	3.61	绝对差值和_sp_x3[64x8]	4	均值_差值_残差[0][64x64]	4.35
均值_差值_残差[0][8x16]	3.62	均值_差值_残差[0][64x32]	4	均值_差值_p[1][16x64]	4.38
计算_符号[16x8]	3.63	率失真优化_裁剪乘法[16x64]	4	均值_差值_p[1][8x32]	4.4
均值_差值_p[1][16x8]	3.65	绝对差值和_pp[32x4]	4.02	率失真优化_裁剪乘法[64x32]	4.4
均值_差值_残差[1][32x4]	3.67	均值_差值_残差[1][16x16]	4.03	均值_差值_残差[0][32x64]	4.41
绝对差值和_sp_x3[32x16]	3.67	均值_差值_残差[0][32x4]	4.05	率失真优化_裁剪乘法[8x8]	4.41
均值_差值_p[1][32x4]	3.68	绝对差值和_sp_x4[64x32]	4.05	率失真优化_数量_非零[16x2]	4.42
计算_符号[64x8]	3.69	绝对差值和_pp[16x8]	4.06	复制_双向预测[16x32]	4.43
亮度_逆_量化_12b[8x4]	3.7	绝对差值和_sp_x4[32x32]	4.06	绝对差值和_sp[64x32]	4.44
绝对差值和_sp_x3[16x64]	3.71	平均_差值_p[0][64x64]	4.08	平均_差值_残差[0][32x16]	4.45
相减_残差[16x8]	3.71	计算_符号[16x32]	4.08	率失真优化_裁剪乘法[32x16]	4.46
率失真优化_裁剪乘法[8x2]	3.71	绝对差值和_空间[32x4]	4.09	绝对差值和_间隔_乘3[32乘32]	4.47
绝对差值和_间隔_乘4[16乘64]	3.75	均值_差异_残差[1][8乘64]	4.11	绝对差值和_间隔[32乘8]	4.49
计算_符号[64乘32]	3.75	绝对差值和_间隔_乘4[16乘16]	4.12	绝对差值和_间隔_x3[64x64]	4.5
计算_符号[16x64]	3.76	平均_差值_残差[1][32x8]	4.12	平均_差值_p[1][32x8]	4.51
减去_残差[16x16]	3.77	平均_差值_残差[0][16x16]	4.13	率失真优化_裁剪乘法[16x2]	4.51
计算_符号[16x16]	3.78	绝对差值和_空间_x3[32x8]	4.16	率失真优化_裁剪乘法[16x4]	4.52
复制_块[3]	3.83	均值_差异_残差[0][8x64]	4.17	绝对差值和_pp[64x64]	4.53
均值_差值_p[0][16x4]	3.84	均值_差值_p[1][64x64]	4.18	块_零[64x16]	4.53
均值_差值_残差[1][32x16]	4.57	率失真优化_裁剪乘法[16x16]	4.99	结构体 平方差和块 s 8_ _ _ _	5.7
均值_差异_p[0][64x32]	4.57	均值_差异_p[0][32x32]	5.02	亮度_逆_量化_12b[16x8]	5.7
均值_差异_残差[0][32x32]	4.59	均值_差异_p[1][32x32]	5.02	struct_平方差和_块_p_8	5.72
绝对差值和_sp[64x8]	4.6	率失真优化_裁剪mul[64x16]	5.02	ang_fast_copy[64x8]	5.72
绝对差值和_sp_x4[16x32]	4.66	mean_diff_残差[0][64x8]	5.03	绝对差值和_sp[32x32]	5.73
绝对差值和_sp_x3[64x16]	4.66	率失真优化_裁剪mul[32x64]	5.03	率失真优化_裁剪mul[8x32]	5.76
均值_差异_p[0][32x4]	4.68	均值_差异_p[0][32x16]	5.04	绝对差值和_每像素[16乘16]	5.77
平均_差值_像素[0][16乘64]	4.7	亮度_反向_量化_10块[8乘8]	5.07	SATD_每像素[16乘4]	5.78
亮度_反向_量化_12块[8乘8]	4.71	率失真优化_裁剪乘法[8乘16]	5.08	绝对差值和_pp[32x32]	5.79
平均_差值_p[1][8x64]	4.72	绝对差值和_sp[16x16]	5.1	绝对差值和_sp[64x64]	5.79
绝对差值和_pp[64x32]	4.72	率失真优化_裁剪乘法[64x64]	5.11	均值_差异_点[1][32乘64]	5.81
率失真优化_裁剪乘数[16乘8]	4.72	绝对差值和_稀疏点[64乘16]	5.12	率失真优化_非零数量_乘[16	6.02
均值_差异_点[0][32乘8]	4.73	均值_差异_残差[1][64乘8]	5.13	均值_差异_残差[1][64x32]	6.03
绝对差值和_sp[16x2]	4.74	亮度_hpp_12[0]	5.14	率失真优化_符号_重新应用[16x4]	6.04
绝对差值和_sp_x3[16x32]	4.75	亮度_hpp_10[0]	5.17	亮度_反_量化_10b[16x4]	6.08
均值_差异_p[0][16x32]	4.75	绝对差值和_pp[64x8]	5.2	绝对差值和_pp[32x64]	6.16
绝对差值和_pp[32x8]	4.75	均值_差异_p[0][32x64]	5.23	绝对差值和_空间[32x64]	6.21
平均值_差值_残差[0][64x16]	4.75	平均值_差值_预测[1][64x8]	5.33	亮度_反向_量化_12块[32x4]	6.24
平均值_差值_预测[0][16x16]	4.76	绝对差值和_空间[16x8]	5.37	绝对差值和_空间[16乘32]	6.31
均值_差值_残差[1][32乘32]	4.78	亮度_逆_量化_10块[8乘16]	5.37	角度_快速_复制[16乘64]	6.34
均值_差值_残差[1][32乘64]	4.82	均值_差值_残差[0][16乘32]	5.44	绝对差值和_pp[16x32]	6.35
块_零[64x32]	4.83	亮度_逆_量化_12b[8x16]	5.46	率失真优化_裁剪乘数[32x32]	6.37
均值_差异_p[1][64x16]	4.85	绝对差值和_pp[64x16]	5.53	亮度_垂直后处理_10[32x4]	6.56
均值_差值_残差[1][16x64]	4.89	绝对差值和_pp[32x16]	5.54	亮度_垂直搜索_10[4x4]	6.57
均值_差值_残差[1][64x64]	4.9	率失真优化_裁剪乘数[32x8]	5.55	复制_块[4]	6.71
均值_差异_p[1][16x32]	4.92	平方差和_sp[16x2]	5.62	亮度_反_量化_12b[16x4]	6.73
均值_差异_p[1][32x16]	4.94	率失真优化_符号_重新应用[16x2]	5.63	add_裁剪_12[16x2]	6.74
绝对差值和_sp_x4[64x16]	4.95	亮度_垂直后处理_12[4x4]	5.67	亮度_垂直搜索_12[4x4]	6.78
率失真优化_clipmul[32x4]	4.95	均值_差值_p[0][64x8]	5.68	绝对差值和_pp[16x64]	6.8
率失真优化_裁剪乘法[16x32]	4.96	率失真优化_裁剪乘法[64x8]	5.69	率失真优化_符号_重新应用[64x32]	6.9

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
亮度_逆_量化_12b[8x32]	6.95	亮度_逆_量化_12b[32x32]	7.8	亮度_逆_量化_12b[8x64]	8.4
亮度_逆_量化_10b[8x32]	6.97	添加_平均值_12[16x2]	7.84	复制_块[5]	8.43
平方差和_sp[16x4]	7	亮度_垂直搜索_10[8x4]	7.84	亮度_垂直搜索_10[4x8]	8.49
亮度_垂直处理_12[4x8]	7	亮度_垂直处理_10[4x16]	7.87	亮度_逆_量化_12b[16x16]	8.49
平方差和_pp[32x4]	7.02	亮度_垂直后处理_10[4x4]	7.88	亮度_逆_量化_10块[32x16]	8.51
亮度_垂直预处理_10[4x4]	7.15	亮度_逆_量化_10块[64x64]	7.93	色度_垂直搜索_12[16x2]	8.64
亮度_垂直预处理_12[4x4]	7.17	添加_平均值_10[16x2]	7.95	亮度_反_量化_12b[16x64]	8.65
SATD_pp[32x4]	7.21	亮度_垂直后处理_12[8x4]	7.97	亮度_反_量化_12b[32x16]	8.66
绝对差值和_sp[32x16]	7.26	块_零[32x16]	7.98	亮度_逆_量化_10b[32x32]	8.66
亮度_垂直后处理_10[4x8]	7.32	亮度_逆_量化_10b[16x32]	8.06	亮度_逆_量化_10b[16x64]	8.69
亮度_垂直预处理_12[4x16]	7.42	亮度_垂直后处理_10[4x16]	8.08	亮度_逆_量化_12块[16x32]	8.71
亮度_垂直预处理_12[4x32]	7.42	亮度_逆_量化_10块[64x32]	8.09	色度_垂直预处理_10[16x4]	8.74
块_零[32x8]	7.42	亮度_垂直处理_10[8x4]	8.11	亮度_垂直搜索_10[4x32]	8.76
率失真优化_符号_重新应用[64x64]	7.43	SATD_pp[16x8]	8.12	率失真优化_符号_重新应用[64x16]	8.84
SATD_pp[64x8]	7.48	率失真优化_符号_重新应用[32x4]	8.12	亮度_hpp_12[1]	8.88
亮度_inv_quant_10b[16x8]	7.48	亮度_垂直后处理_12[4x16]	8.13	SATD_sp[64x16]	8.91
SATD_pp[32x8]	7.52	平方差和_pp[16x2]	8.14	亮度_垂直搜索_12[4x8]	8.96
率失真优化_符号_重新应用[16x64]	7.55	率失真优化_符号_重新应用[32x32]	8.14	亮度_垂直搜索_12[8x8]	8.96
亮度_垂直搜索_12[32x4]	7.57	平方差和_ss[16x2]	8.15	亮度_垂直后处理_12[4x8]	8.98
色度_hps_10[2]	7.61	亮度_垂直预处理_10[4x8]	8.17	色度_垂直搜索_10[16x2]	8.99
绝对差值和_sp[16x64]	7.65	平方差和_pp[16x8]	8.18	亮度_逆_量化_10b[64x16]	9.01
块_零[32x4]	7.65	率失真优化_符号_重新应用[32x8]	8.18	SATD_sp[16x4]	9.02
亮度_垂直后处理_12[16x4]	7.71	色度_垂直处理_10[16x2]	8.23	亮度_逆_量化_10b[32x64]	9.04
率失真优化_符号_重新应用[16x16]	7.72	色度_hps_10[5]	8.24	率失真优化_符号_重新应用[16x8]	9.04
亮度_垂直处理_12[4x4]	7.74	SATD_sp[32x4]	8.25	ang_fast_copy[16x16]	9.06
add_裁剪_10[16x2]	7.75	add_裁剪_12[16x32]	8.34	亮度_垂直搜索_10[8x8]	9.09
亮度_inv_quant_10b[8x64]	7.76	率失真优化_sign_reapply[32x64]	8.34	添加_裁剪_10[16x4]	9.09
亮度_逆_量化_10b[64x8]	7.77	SATD_sp[32x64]	8.35	添加_裁剪_10[16x64]	9.1
亮度_垂直处理_10[4x8]	7.79	亮度_逆_量化_10b[16x16]	8.35	块_零[32x64]	9.11
亮度_垂直处理_10[4x4]	7.8	亮度_垂直预处理_12[8x4]	8.4	亮度_垂直后处理_10[8x16]	9.13
色度_垂直搜索_10[32x4]	9.15	色度_垂直后处理_10[16x2]	9.7	亮度_垂直预处理_12[4x8]	10.33
亮度_inv_quant_12b[64x16]	9.16	亮度_inv_quant_12b[64x8]	9.73	亮度_垂直处理_12[16x4]	10.33
亮度_垂直搜索_12[4x32]	9.21	亮度_垂直处理_12[16x64]	9.75	add_裁剪_10[16x8]	10.33
亮度_垂直搜索_12[8x4]	9.22	ang_快速_复制[16x4]	9.75	add_裁剪_12[64x8]	10.38
亮度_垂直搜索_12[4x16]	9.23	亮度_inv_量化_12b[64x64]	9.76	亮度_垂直后处理_10[4x32]	10.39
亮度_垂直预处理_12[16x8]	9.23	SATD_pp[32x32]	9.79	SATD_sp[32x8]	10.4
亮度_inv_量化_12b[32x64]	9.24	SATD_pp[32x16]	9.81	亮度_垂直搜索_12[16x4]	10.4
亮度_垂直处理_10[32x4]	9.25	亮度_垂直处理_10[16x4]	9.81	色度_垂直预处理_10[64x32]	10.41
亮度_垂直预处理_10[8x4]	9.29	角度_快速_复制[32x64]	9.81	亮度_垂直预处理_12[32x4]	10.44
亮度_垂直后处理_10[16x4]	9.29	平方差和_pp[16x4]	9.89	SATD_pp[16x64]	10.46
亮度_垂直处理_10[4x32]	9.3	亮度_inv_quant_10b[32x8]	9.9	亮度_垂直预处理_10[16x64]	10.5
块_零[16x2]	9.3	率失真优化_符号_重新应用[32x16]	9.92	亮度_hpp_10[1]	10.51
亮度_逆_量化_12b[64x32]	9.3	亮度_垂直预处理_10[4x32]	9.97	亮度_垂直预处理_12[32x8]	10.52
添加_平均值_12[16x4]	9.32	亮度_垂直处理_10[8x8]	9.97	亮度_垂直搜索_10[8x64]	10.53
亮度_垂直处理_12[4x16]	9.39	亮度_垂直处理_12[32x32]	10.01	加_平均值_10[32x16]	10.61
角度_快速_复制[16x32]	9.39	亮度_垂直处理_12[32x64]	10.01	亮度_垂直搜索_10[8x16]	10.61
角度_快速_复制[32x32]	9.39	加_裁剪_12[16x64]	10.1	亮度_垂直预处理_12[16x4]	10.61
添加_平均值_10[16x4]	9.42	亮度_垂直预处理_10[4x16]	10.11	亮度_hps_10[1]	10.65
亮度_垂直处理_12[4x32]	9.43	亮度_垂直处理_12[8x4]	10.11	ang_快速_复制[32x4]	10.66
亮度_垂直后处理_10[16x8]	9.44	色度_垂直处理_10[16x4]	10.12	SATD_sp[64x64]	10.67
SATD_sp[64x8]	9.45	亮度_垂直后处理_12[32x8]	10.12	亮度_垂直预处理_12[64x32]	10.68
块_零[32x32]	9.47	亮度_垂直后处理_10[8x4]	10.13	亮度_垂直搜索_10[16x64]	10.7
亮度_垂直搜索_10[4x16]	9.51	亮度_垂直后处理_12[32x32]	10.13	亮度_垂直预处理_10[16x4]	10.75
率失真优化_数量_非零[32x4]	9.51	角度_快速_复制[64x32]	10.16	SATD_sp[16x16]	10.76
率失真优化_符号_重新应用[16x32]	9.52	亮度_垂直搜索_10[32x8]	10.21	亮度_垂直后处理_10[64x16]	10.81
亮度_垂直预处理_10[32x64]	9.55	率失真优化_数量_非零[16x16]	10.23	色度_垂直处理_10[32x4]	10.85
率失真优化_符号_重新应用[64x8]	9.55	亮度_垂直搜索_10[32x4]	10.24	亮度_垂直搜索_10[16x4]	10.85
亮度_反量化_b[32x4]	9.59	亮度_垂直预处理_10[32x4]	10.27	亮度_垂直预处理_10[16x8]	10.85
色度_垂直预处理_10[16x2]	9.62	角度_快速_复制[32x16]	10.3	SATD_pp[16x16]	10.89
add_裁剪_10[16x32]	9.68	SATD_pp[64x64]	10.32	平方差和_ss[16x4]	10.9

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
率失真优化_数量_非零[16x8]	10.94	色度_垂直处理_10[32x32]	11.58	平方差和_ss[32x4]	12.13
SATD_sp[32x16]	11	色度_垂直预处理_12[16x2]	11.59	亮度_垂直后处理_10[64x32]	12.15
色度_hps_10[3]	11	亮度_hps_10[4]	11.6	亮度_垂直预处理_10[8x8]	12.17
亮度_垂直后处理_10[16x64]	11	色度_垂直后处理_10[32x16]	11.63	添加_裁剪_10[16x16]	12.21
角度_快速_复制[64x16]	11.01	亮度_垂直后处理_12[16x64]	11.64	色度_垂直搜索_12[16x16]	12.24
平方差和_sp[16x8]	11.04	SATD_pp[32x64]	11.66	色度_垂直预处理_12[32x64]	12.26
SATD_sp[16x32]	11.04	亮度_垂直处理_10[32x32]	11.68	亮度_垂直搜索_12[8x32]	12.27
色度_垂直预处理_12[16x4]	11.04	亮度_垂直后处理_12[8x8]	11.69	平方差和_pp[32x32]	12.28
亮度_垂直后处理_12[32x4]	11.04	亮度_垂直处理_10[8x32]	11.71	亮度_垂直预处理_12[32x64]	12.28
亮度_垂直后处理_12[8x64]	11.05	平方差和_pp[32x8]	11.72	添加_平均值_12[16x8]	12.3
亮度_inv_quant_12b[32x8]	11.05	色度_垂直搜索_12[16x4]	11.75	色度_垂直搜索_10[16x4]	12.33
SATD_pp[16x32]	11.08	亮度_垂直处理_12[16x32]	11.75	亮度_垂直后处理_10[8x8]	12.38
亮度_垂直处理_10[32x64]	11.08	亮度_垂直处理_10[16x64]	11.79	平方差和_sp[64x16]	12.48
亮度_垂直后处理_10[16x16]	11.1	率失真优化_num_非零[32x8]	11.8	亮度_垂直处理_12[64x64]	12.5
ang_fast_copy[64x64]	11.1	率失真优化_num_非零[64x8]	11.8	色度_垂直后处理_10[16x32]	12.51
SATD_pp[64x32]	11.17	亮度_垂直处理_12[32x4]	11.81	亮度_垂直搜索_12[16x64]	12.51
色度_垂直预处理_12[64x16]	11.17	添加_平均值_10[16x8]	11.82	平方差和_sp[32x4]	12.53
SATD_pp[64x16]	11.18	色度_垂直后处理_10[16x4]	11.82	色度_垂直预处理_12[32x8]	12.53
色度_垂直预处理_12[64x32]	11.2	亮度_垂直处理_12[32x8]	11.82	块_零[16x32]	12.53
添加_裁剪_12[16x4]	11.2	块_零[16x16]	11.85	亮度_垂直搜索_12[32x8]	12.55
SATD_sp[16x8]	11.23	亮度_垂直搜索_12[16x16]	11.87	SATD_sp[16x64]	12.58
亮度_垂直处理_10[64x32]	11.3	亮度_垂直搜索_12[64x64]	11.88	亮度_垂直搜索_10[64x8]	12.58
亮度_垂直处理_10[64x64]	11.35	亮度_垂直搜索_12[8x64]	11.89	色度_垂直预处理_12[16x16]	12.64
亮度_垂直搜索_12[32x16]	11.42	色度_垂直预处理_10[32x64]	11.9	添加_裁剪_12[32x16]	12.64
亮度_垂直后处理_12[4x32]	11.43	添加_裁剪_12[16x16]	11.91	色度_垂直后处理_10[32x4]	12.68
亮度_垂直后处理_12[16x8]	11.44	添加_平均值_12[32x4]	11.92	色度_垂直预处理_10[16x64]	12.69
添加_裁剪_12[16x8]	11.47	亮度_垂直搜索_12[8x16]	11.93	亮度_垂直搜索_12[16x8]	12.72
色度_hps_10[4]	11.48	亮度_垂直搜索_10[16x8]	11.93	色度_垂直搜索_12[32x4]	12.73
add_裁剪_10[64x16]	11.48	色度_垂直处理_10[64x16]	11.95	亮度_垂直预处理_12[16x16]	12.74
添加_裁剪_10[32x4]	11.52	添加_裁剪_12[64x32]	12.1	亮度_垂直处理_12[8x32]	12.78
亮度_垂直搜索_10[32x16]	12.8	亮度_垂直处理_12[64x32]	13.27	亮度_垂直处理_12[16x16]	13.76
SATD_sp[64x32]	12.81	亮度_hpp_12[3]	13.3	亮度_垂直处理_12[16x8]	13.78
亮度_hpp_12[2]	12.82	色度_垂直后处理_10[32x32]	13.31	亮度_垂直后处理_10[32x16]	13.81
亮度_垂直搜索_12[16x32]	12.83	亮度_垂直处理_10[32x8]	13.31	添加_平均值_10[16x64]	13.84
色度_垂直处理_10[16x64]	12.84	亮度_垂直搜索_10[32x64]	13.31	色度_垂直预处理_12[32x4]	13.84
添加_裁剪_10[32x64]	12.84	亮度_垂直预处理_12[8x8]	13.32	亮度_垂直搜索_12[64x32]	13.84
角度_快速_复制[16x8]	12.86	SATD_sp[32x32]	13.34	平方差和_pp[16x32]	13.87
色度_垂直后处理_10[32x64]	12.87	亮度_垂直处理_10[8x16]	13.37	亮度_垂直后处理_12[16x32]	13.87
add_裁剪_12[32x32]	12.87	亮度_垂直搜索_10[64x16]	13.37	亮度_垂直搜索_10[64x64]	13.91
add_平均值_12[16x32]	12.89	亮度_垂直处理_12[8x64]	13.41	亮度_垂直后处理_10[32x8]	13.94
add_平均值_12[64x64]	12.93	亮度_垂直搜索_10[64x32]	13.41	亮度_垂直后处理_10[64x8]	13.94
色度_垂直搜索_10[16x16]	12.95	色度_垂直搜索_12[64x64]	13.43	色度_垂直后处理_10[16x8]	13.95
色度_垂直搜索_12[64x8]	12.97	率失真优化_数量_非零[16x32]	13.44	亮度_垂直搜索_12[64x8]	13.95
平方差和_sp[32x8]	12.98	色度_垂直搜索_10[64x64]	13.46	色度_垂直预处理_12[16x8]	14.01
add_裁剪_12[32x4]	12.98	亮度_垂直预处理_12[16x32]	13.46	亮度_垂直预处理_12[32x32]	14.01
亮度_垂直搜索_10[16x32]	13	色度_垂直搜索_12[64x32]	13.47	色度_垂直后处理_10[32x8]	14.07
亮度_垂直处理_10[16x16]	13.03	亮度_垂直预处理_10[32x32]	13.49	亮度_垂直后处理_10[8x32]	14.09
亮度_垂直处理_12[8x8]	13.04	亮度_垂直搜索_12[32x32]	13.52	亮度_垂直后处理_12[16x16]	14.09
色度_垂直预处理_10[64x64]	13.05	块_零[16x64]	13.56	add_裁剪_10[32x8]	14.1
亮度_垂直搜索_12[64x16]	13.05	亮度_垂直搜索_10[8x32]	13.57	平方差和_pp[64x64]	14.16
亮度_垂直后处理_12[8x16]	13.07	亮度_垂直处理_10[16x8]	13.61	色度_垂直预处理_10[16x8]	14.19
亮度_垂直搜索_10[16x16]	13.1	亮度_垂直处理_12[32x16]	13.62	色度_垂直预处理_10[32x16]	14.2
亮度_垂直预处理_10[64x8]	13.13	块_零[16x8]	13.63	平方差和_pp[32x16]	14.21
平方差和_ss[16x8]	13.14	亮度_垂直后处理_10[32x32]	13.64	色度_垂直处理_10[16x32]	14.23
add_平均值_10[32x4]	13.14	亮度_垂直预处理_12[8x16]	13.65	亮度_垂直后处理_12[8x32]	14.23
亮度_hpp_10[2]	13.15	ang_fast_copy[32x8]	13.68	亮度_垂直处理_12[64x16]	14.25
add_裁剪_10[32x32]	13.22	色度_垂直后处理_10[64x64]	13.69	平方差和_ss[32x16]	14.26
色度_垂直后处理_10[16x64]	13.24	色度_垂直后处理_10[16x16]	13.72	亮度_垂直预处理_10[64x16]	14.33
亮度_垂直处理_10[32x16]	13.26	色度_垂直搜索_10[16x8]	13.73	色度_垂直搜索_10[64x16]	14.34
色度_垂直处理_10[64x64]	13.27	亮度_垂直处理_10[8x64]	13.74	亮度_垂直预处理_10[8x32]	14.35

原语	IPC Gain	原语	IPC Gain	原语	IPC Gain
亮度_垂直后处理_12[64x32]	14.35	亮度_垂直预处理_10[32x8]	14.98	add_裁剪_10[64x64]	15.52
平方差和_ss[32x8]	14.41	亮度_hps_10[3]	15.01	add_裁剪_12[32x8]	15.54
亮度_垂直预处理_10[16x16]	14.43	亮度_hpp_10[4]	15.01	平方差和_ss[32x32]	15.56
平方差和_sp[32x16]	14.44	色度_垂直处理_10[32x8]	15.02	色度_垂直处理_10[16x16]	15.56
色度_垂直处理_10[16x8]	14.44	添加_裁剪_10[64x8]	15.07	色度_垂直搜索_10[32x16]	15.58
色度_垂直预处理_10[32x4]	14.45	色度_垂直预处理_12[32x32]	15.09	色度_垂直搜索_12[32x64]	15.58
亮度_垂直预处理_12[8x32]	14.49	亮度_垂直后处理_12[64x64]	15.09	色度_垂直预处理_12[32x16]	15.59
亮度_垂直处理_10[64x16]	14.55	色度_垂直搜索_12[32x32]	15.1	色度_垂直后处理_10[64x16]	15.59
添加_平均值_12[32x8]	14.56	色度_垂直处理_10[32x16]	15.13	平方差和_sp[32x32]	15.61
亮度_hpp_10[3]	14.59	平方差和_sp[16x16]	15.16	亮度_hpp_12[4]	15.66
色度_垂直搜索_10[32x8]	14.6	亮度_hps_10[2]	15.16	平方差和_ss[64x32]	15.68
添加_平均值_12[16x16]	14.61	亮度_垂直预处理_10[8x16]	15.18	色度_垂直后处理_10[64x32]	15.68
平方差和_pp[16x16]	14.66	色度_垂直搜索_10[32x64]	15.19	亮度_垂直预处理_10[32x16]	15.7
添加_平均值_12[32x32]	14.66	色度_垂直搜索_12[64x16]	15.24	色度_垂直预处理_10[32x8]	15.72
亮度_垂直后处理_12[64x8]	14.66	色度_垂直处理_10[32x64]	15.27	平方差和_sp[64x32]	15.75
色度_垂直搜索_12[16x8]	14.7	亮度_垂直后处理_10[32x64]	15.29	率失真优化_num_非零[16x64]	15.77
add_裁剪_12[32x64]	14.71	平方差和_ss[64x16]	15.33	色度_垂直处理_10[64x8]	15.83
亮度_垂直后处理_12[32x16]	14.72	色度_垂直预处理_10[16x32]	15.33	添加_平均值_10[32x32]	15.86
亮度_垂直处理_10[64x8]	14.74	平方差和_ss[16x16]	15.38	率失真优化_num_非零[32x16]	15.89
亮度_垂直预处理_12[64x16]	14.74	平方差和_sp[64x8]	15.39	色度_垂直搜索_10[64x8]	15.94
色度_垂直搜索_10[64x32]	14.75	平方差和_pp[64x16]	15.39	亮度_垂直预处理_10[16x32]	15.94
色度_垂直预处理_12[64x64]	14.76	亮度_垂直预处理_10[64x64]	15.39	亮度_垂直后处理_12[64x16]	15.94
亮度_垂直预处理_12[32x16]	14.8	亮度_垂直后处理_10[16x32]	15.4	色度_垂直预处理_10[32x32]	16.05
色度_垂直预处理_12[16x64]	14.84	添加_平均值_12[32x16]	15.41	添加_裁剪_12[64x64]	16.12
亮度_垂直预处理_10[8x64]	14.87	添加_平均值_10[16x16]	15.43	添加_平均值_12[16x64]	16.14
亮度_垂直处理_10[16x32]	14.89	色度_垂直搜索_12[32x8]	15.44	平方差和_pp[64x32]	16.27
亮度_垂直处理_12[64x8]	14.91	色度_垂直预处理_10[64x16]	15.44	平方差和_sp[32x64]	16.35
色度_垂直预处理_10[64x8]	14.97	色度_垂直后处理_10[64x8]	15.5	平方差和_pp[64x8]	16.38
色度_垂直预处理_12[64x8]	14.97	色度_垂直预处理_10[16x16]	15.52	add_平均值_10[16x32]	16.42
add_裁剪_10[32x16]	14.97	add_裁剪_12[64x16]	15.52	亮度_垂直搜索_10[32x32]	16.48
亮度_垂直后处理_12[32x64]	16.57	平方差和_sp[16x64]	17.14	率失真优化_数字_非零[64x32]	17.96
色度_垂直搜索_10[32x32]	16.58	平方差和_pp[16x64]	17.21	色度_垂直搜索_12[16x64]	18.43
平方差和_sp[16x32]	16.66	亮度_垂直后处理_10[8x64]	17.23	添加_平均值_12[64x8]	18.44
添加_平均值_10[32x8]	16.68	添加_平均值_12[64x16]	17.25	色度_垂直处理_10[64x32]	18.49
色度_垂直预处理_12[16x32]	16.73	平方差和_ss[16x64]	17.31	添加_平均值_10[64x64]	19.13
色度_垂直搜索_12[32x16]	16.75	率失真优化_数量_非零[64x16]	17.44	色度_垂直搜索_10[16x32]	19.2
平方差和_pp[32x64]	16.83	添加_平均值_10[64x8]	17.51	率失真优化_数量_非零[32x64]	19.28
亮度_垂直搜索_12[32x64]	16.84	亮度_垂直预处理_12[8x64]	17.57	添加_平均值_10[64x32]	19.31
亮度_垂直预处理_12[64x8]	16.92	亮度_垂直后处理_10[64x64]	17.59	亮度_垂直预处理_12[16x64]	19.67
添加_平均值_10[32x64]	16.97	率失真优化_数量_非零[32x32]	17.62	块_零[16x4]	19.78
添加_裁剪_10[64x32]	16.98	亮度_垂直预处理_12[64x64]	17.71	平方差和_ss[16x32]	19.84
亮度_垂直处理_12[8x16]	17.02	添加_平均值_12[32x64]	17.91	率失真优化_num_非零[64x64]	19.85
色度_垂直搜索_12[16x32]	17.05	色度_垂直搜索_10[16x64]	17.94	平方差和_平方和[32x64]	21.72
添加_平均值_12[64x32]	17.11	添加_平均值_10[64x16]	17.96	亮度_垂直预处理_10[64x32]	22.48
平方差和_平方和[64x8]	23.33				