基于倚天 710 通用计算芯片的 H265 高性能视频编解码

复旦大学 VIP-lab
团队成员：刘超、侯秉镜、李思芮
指导老师：范益波教授

摘要：视频编解码技术是互联网流媒体时代非常重要且基础的技术。在本设计中，我们基于倚天 710 通用计算芯片，对 X265 开源软件编码器进行优化。具体的优化包含了两个方面，其一是对 X265 编码算法的速度提升，通过所设计的问题分析方法来改善编码器的复杂度。其二则是对 X265 的编码效率进行提升，通过所设计的前处理滤波使得重要参考帧的压缩可以兼顾到被参考帧的纹理信息，从而获得更高的压缩效率。实验表现上，在相同的压缩效率下，快速算法部分带来了大约 10%-20%的复杂度降低，其中对 veryslow 档次可以获得 **39%**的复杂度减少并仅带来了 **1.47%**的压缩效率损失。对高性能版本，前处理滤波对不同的档次可以带来 2.40%-4.34%的压缩效率提升，其中所设计方法在 ultrafast 档次下对 BQTerrace 序列带来了显著 **15.75%**的压缩效率提升。
关键字：视频编码、快速算法、前处理、X265、HEVC

Abstart: Video coding technology is a very essential and basic technology in the era of Internet streaming media. In this design, we make optimization for open source software encoder X265 based on the Yitian 710 chip. The specific optimization consists of two aspects, one is the fast algorithm design for X265, and the codec complexity performance is reduced by the proposed texture-analysis method. The second is to improve the coding efficiency of X265 by designing a pre-processing filter that allows the compression of important reference frames to consider the texture information of the referenced frames, thus obtaining a higher compression efficiency. Experimentally, the proposed fast algorithm brings about 10%-20% complexity reduction under the same compression efficiency. About **39%** complexity reduction and only **1.47%** compression efficiency loss is achieved under very-slow preset. For the high performance version, the preprocessing filtering brings a 2.40%-4.34% improvement in compression efficiency for different presets. Besides, the proposed method brings a significant **15.75%** improvement in BD-rate reduction for BQTerrace sequences in ultrafast preset.
Key words: Video coding, fast algorithms, pre-processing, X265, HEVC

一、X265 的快速编码算法设计

1 引言

H.265/HEVC 相对于其前一代视频编码标准 H.264/AVC，可以在相同视频质量下再节省 50%的码率。与此同时带来的是编码复杂度的急剧上升，包括了新增加的编码单元四叉树划分、35 种帧内预测模式等等。高性能编码器的要求是以低的复杂度完成高的率失真性能的视频编码任务，如此高的编码复杂度给高性能编码器的设计实现带来了巨大的挑战。因此，学术界和工业界针对编码器的率失真优化算法进行了大量的研究。

帧内预测是 H.265/HEVC 编码器中的重要组成部分。在随机接入编码中帧内编码帧是重要的参考帧，在全帧内编码中帧内编码帧是唯一一种编码帧。因此，帧内预测的率失真优化算法也是当前广泛研究的热点之一。帧内预测过程中需要对编码单元、预测单元、变换单元的划分进行决策，以及需要对帧内预测模式进行决策等。帧内预测的率失真优化算法的研究就是在保证率失真性能的前提下，去降低包括划分决策和模式决策在内的算法的复杂度，以提高编码速度。例如，论文[1]在 HM 上应用了一种基于分析视频内容进行快速划分决策的算法。此外，还可以基于编码过程中的时空信息来进行快速划分决策以及模式决策等，以实现帧内预测算法的优化。

2 设计方法

2.1 结构分析

在 x265 的编码流程中，compressCTU 函数会调用 compressIntraCU 函数进行帧内编码 CTU 的编码。在 compressIntraCU 函数中，会对当前节点尝试进行不划分编码和四叉树划分编码，并选择其中具有最小率失真代价的编码方式。在对 CU 进行编码时，会对每一个 PU 先尝试 35 种帧内预测模式进行粗选，然后得到其中最好的几种模式进行无 TU 划分的细选，最后得到最好的一种模式进行有 TU 划分的帧内编码，从而决定了 CU 的帧内编码方式。

在以上过程中，对节点尝试进行不划分编码和四叉树划分编码会消耗大量的时间。因此我们应用了一种基于分析编码单元内容特性而快速判决划分的算法，在对节点进行编码尝试之前分析节点内容特性并预先作出划分判决，这样以避免进行大量的编码尝试，从而可以大大提高编码速度，同时由于判决的准确性也可以保持高的率失真性能。

2.2 算法设计

对于一个节点，首先需要计算亮度分量的均值 Ym 和亮度分量的复杂度信息：

$$Gh = \left| \sum_{i=0}^{N-1} \sum_{j=0}^{\frac{N}{2}-1} |Y(i,j) - Ym| - \sum_{i=0}^{N-1} \sum_{j=\frac{N}{2}}^{N-1} |Y(i,j) - Ym| \right|$$

$$Gv = \left| \sum_{i=0}^{\frac{N}{2}-1} \sum_{j=0}^{N-1} |Y(i,j) - Ym| - \sum_{i=\frac{N}{2}}^{N-1} \sum_{j=0}^{N-1} |Y(i,j) - Ym| \right|$$

$$G45 = \left| \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} |Y(i,j) - Ym| - \sum_{i=0}^{N-1} \sum_{j=0}^{i} |Y(i,j) - Ym| \right|$$

$$G135 = \left| \sum_{i=0}^{N-1} \sum_{j=0}^{N-1-i} |Y(i,j) - Ym| - \sum_{i=0}^{N-1} \sum_{j=N-1-i}^{N-1} |Y(i,j) - Ym| \right|$$

$$G'h(k) = \left| \sum_{i=\frac{kN}{4}}^{\frac{(k+1)N}{4}-1} \sum_{j=0}^{\frac{N}{2}-1} |Y(i,j) - Ym| - \sum_{i=\frac{kN}{4}}^{\frac{(k+1)N}{4}-1} \sum_{j=\frac{N}{2}}^{N-1} |Y(i,j) - Ym| \right|$$

$$G'v(k)=\left|\sum_{i=0}^{\frac{N}{2}-1}\sum_{j=\frac{kN}{4}}^{\frac{(k+1)N}{4}-1}|Y(i,j)-Ym|-\sum_{i=\frac{N}{2}}^{N-1}\sum_{j=\frac{kN}{4}}^{\frac{(k+1)N}{4}-1}|Y(i,j)-Ym|\right|$$

然后对亮度分量进行滤波处理并计算滤波后的复杂度信息:

$$fh\_Y(i,j)=|Y(i-1,j)-Y(i+1,j)|$$
$$fv\_Y(i,j)=|Y(i,j-1)-Y(i,j+1)|$$
$$f45\_Y(i,j)=|Y(i-1,j-1)-Y(i+1,j+1)|$$
$$f135\_Y(i,j)=|Y(i+1,j-1)-Y(i-1,j+1)|$$
$$Lh=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|fh\_Y(i,j)-fh\_Ym|$$
$$Lv=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|fv\_Y(i,j)-fv\_Ym|$$
$$L45=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|f45\_Y(i,j)-f45\_Ym|$$
$$L135=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|f135\_Y(i,j)-f135\_Ym|$$

对于四叉树划分得到的四个子节点，也分别计算亮度分量的复杂度信息 sGh(k)、sGv(k)、sG45(k)、sG135(k)和滤波后的复杂度信息 sLh(k)、sLv(k)、sL45(k)、sL135(k)。

最后将计算出的所有复杂度信息与两个阈值 Thg 和 Thl 进行比较得出划分判决。如果在某一方向上的复杂度信息均不大于给定的阈值则当前节点判决为不划分，如果在所有方向上的复杂度信息均大于给定的阈值则当前节点判决为划分，否则当前节点判决为需要继续尝试。具体代码如表 1 所示。

表 1 所设计算法伪代码

```
if ((pGh <= Thg && pLh <= Thl && maxSGh <= Thg / 4 && maxPGhh <= Thg / 4 && maxSLh
<= Thl / 4)
|| (pGv <= Thg && pLv <= Thl && maxSGv <= Thg / 4 && maxPGvv <= Thg / 4 && maxSLv
<= Thl / 4)
|| (pG45 <= Thg && pL45 <= Thl && maxSG45 <= Thg / 4 && maxSL45 <= Thl / 4)
|| (pG135 <= Thg && pL135 <= Thl && maxSG135 <= Thg / 4 && maxSL135 <= Thl / 4)
) {
    flagSplit   = false;
    flagUnsplit = true;
}
else if ((pGh > Thg && pGv > Thg && pG45 > Thg && pG135 > Thg)
|| (pLh > Thl && pLv > Thl && pL45 > Thl && pL135 > Thl)
) {
    flagSplit   = true;
    flagUnsplit = false;
}
else {
    flagSplit   = true;
    flagUnsplit = true;
}
```

2.3 代码实现

在软件编码器中，算法的代码实现也会很大地影响到运行速度和性能。由算法描述可以知道，在分析节点内容特性时需要进行大量的滤波处理和复杂度计算。因此，在保持代码具有良好可读性、可维护性、可扩展性的同时提高代码的执行效率也是一个重要的考虑因素。

首先是旁路不必要的处理。因此只需要对同时具有不划分和划分两种可能性的中间节点进行以上的分析和判决，而对于强制划分或强制不划分的节点则直接跳过而按照正常流程进行处理即可。此外还有对于子节点不需要计算 sG'h(k)(l)和 sG'v(k)(l)，因此在函数中设置一个标记位来决定是否需要跳过这一部分复杂度的计算。

其次是简化计算和循环操作。例如使用定点化来移除浮点数运算，使用移位操作来替代乘法操作和除法操作，预先计算和存储大量使用的公共项来避免重复计算，合并代码来避免使用大量和复杂的循环等等。

另外是利用运行时的时空冗余。由于在当前节点和子节点的计算中都需要对亮度分量进行滤波处理，而子节点用到的是当前节点的一部分内容，因此在当前节点滤波完成后不会立即释放，这样子节点可以简单地通过指针索引来获取子节点所需要的内容，从而不需要再对子节点再进行一次滤波处理。通过这样父子块的复用也可以减少处理过程。

3 实验结果

使用 HEVC 通用测试序列，在 All Intra 配置下对前 100 帧进行了测试。图 1 显示了在 veryslow、slow、medium、fast、superfast 五个档次下原始编码器（蓝色）的率失真性能-复杂度曲线和应用了算法优化后（橙色）的率失真性能-复杂度曲线。数据按照原始编码器 veryslow 档次的测试结果作为了基准（BD Rate 0%，复杂度 100%）。可以看出，应用了算法优化后，在相同复杂度下具有更好的率失真性能（BD Rate 增加更少），在相同率失真性能下具有更低的复杂度，与原始编码器相比提升大概在 10%~20%左右。

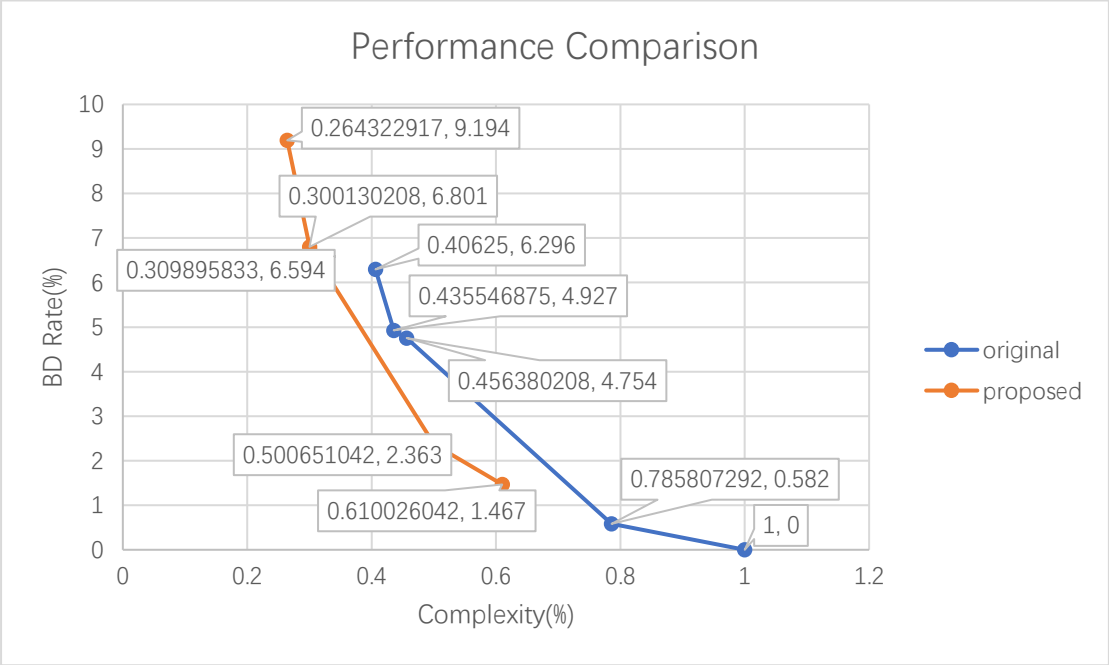

图 1 RD-复杂度曲线表示

表 2 显示了在 veryslow、slow、medium、fast、superfast 五个档次下原始编码器和应用了算法优化后的编码时间相对减少量和 BD Rate 相对增加量。可以看出，应用了算法优化

后，编码时间能相对减少 31%~39%，而 BD Rate 也会相应相对增加 1.47~2.73%。其中在 superfast 档次下的 BD Rate 相对增加量最多，可能的原因是因为在 superfast 下的编码工具很少，因此再应用快速算法之后会产生较大的 BD Rate 性能损失。

表 2 不同档次下复杂度降低和压缩效率变换表现

档次	编码时间相对减少量	BD Rate 相对增加量
superfast	34.90%	2.73%
fast	31.10%	1.79%
medium	32.10%	1.76%
slow	36.30%	1.77%
veryslow	39.00%	1.47%

最后，考虑到实现难度，还可以对算法进行如下的修改：例如，（1）考虑整个 CTU 中的父子块的复用，只对最大的节点进行一次滤波处理，其他节点都通过指针索引来获取所需要的内容。但是这样需要对最大的节点的滤波处理结果进行存储，并且将其传递到不同深度和不同节点的 compressIntraCU 中的调用。（2）更多地利用运行时的时空统计信息，使用相邻块的深度来辅助当前节点的划分判决。

4 展望

未来的工作可以从以下几个方面入手：（1）在对不同节点的内容特性分析的过程中存在着较多的重复计算。当前算法中考虑了一个节点中父子块的滤波的复用。还可以考虑整个 CTU 中的父子块的复用，例如只对最大的节点进行一次滤波处理，其他节点都通过指针索引来获取所需要的内容。以及可以将已计算节点的复杂度信息存储然后用于后续直接使用。但是这样需要对滤波处理结果和复杂度信息计算结果进行存储，并且将其传递到不同深度和不同节点的 compressIntraCU 中的调用。（2）对判决算法作出修改，更多地利用运行时的时空统计信息，使用相邻块的深度来辅助当前节点的划分判决。（3）在对不同节点的内容特性分析的过程中存在较多数据密集但相对简单的计算，因此可以针对不同的架构使用 SIMD 指令进行实现，以加速处理过程。

二、X265 的高性能算法设计

1 引言

视频的采集和传输会引入噪声，而这些噪声缺乏相关性，很难有效地编码。为了提升视频的压缩率，可以在编码前对视频进行前处理，滤除噪声。而 X265 中没有视频编码前处理的工具，因此我们引入了 MCTF（Motion Compensated Temporal Filter，基于运动补偿的时域滤波）[2-6]工具，以优化压缩效率。

MCTF 属于时域滤波，其核心思想是帧间平均，也就是将当前帧与时间上相邻的多个帧（以下称为“参考帧”）进行加权平均，从而平滑噪声。MCTF 中包含运动估计和运动补偿的过程，使得当前帧中的块可以与各个参考帧中的最佳匹配块进行加权平均，从而避免引入运动的“拖尾”现象。

MCTF 的使用与 GOP（Group of Pictures）结构息息相关。GOP 结构反映了帧与帧之间的参考关系，这个参考关系也可以用时间层来描述，处于时间层越底层的帧，在帧间预测中可能被越多的帧参考。而 MCTF 仅需作用于处于较低时间层的帧，这是因为处于较高时间层的 B 帧，往往会分配更大的 QP（Quantization Parameter）值，因而在视频编码中的量化环节，已经能够去除相当一部分的高频噪声；并且 B 帧所采用的双向预测已包含帧间平均的过程，也有利于噪声的去除。而处于较低时间层的帧，可能无法借由编码过程去除噪声，因此需要在编码之前，通过 MCTF 进行前处理，完成去噪。

目前，MCTF 已经应用于 HM，能够很好地适配 HM 中规整的金字塔型 GOP 结构。而 X265 尚未支持 MCTF，并且由于 X265 中的 GOP 结构非常灵活多变，因此在支持 MCTF 工具时，有必要进行重新设计。

2 设计方法

2.1 整体结构

MCTF 仅需对视频中的一些关键帧进行前处理。对 X265 这样的具有比较灵活的 GOP 结构的视频编码器，首先需要确定哪些帧需要进行滤波。而对于 X265 的任意 GOP 结构，我们也进行了 MCTF 的重新设计，通过前向动态的参考帧滤波和后向静态的参考帧滤波方法，来适应 X265 中灵活的 mini-GOP 帧结构。

2.2 确定待滤波帧

X265 中，帧类型可以细分为 I/i/P/B/b 中的一种。其中，I 和 i 分别表示关键和非关键 I 帧，B 和 b 分别表示被参考和不被参考的 B 帧。从时间层的角度来看，X265 中的时间层是有限的，即 I/i/P 帧位于最底层，B 帧位于其上的一层，而所有的 b 帧位于顶层时间层。

考虑到 X265 中的 GOP 结构具有短小灵活、时间层数少的特点，我们仅对处于最底层时间层的 I/i/P 帧进行 MCTF。

2.3 确定参考帧

在 X265 中，编码的 GOP 结构并不是固定的，而是具有很强的自适应性。根据 X265 的流程，编码之前首先会按照播放顺序，向输入队列中读入多帧，然后根据配置项的约束，决策输入队列中各个帧的类型（I/i/P/B/b）。当完成类型决策的帧能够构成一个 mini-GOP 时，则将该 mini-GOP 输出到待编码队列。而后将从待编码队列每次取出一帧进行编码。

如果 mini-GOP 仅包含一个关键 I 帧，则此时不存在前向参考帧。而大多数情况下，mini-GOP 会包含一个 i/P 帧和若干可能参考该帧的 B/b 帧，则这些 B/b 帧应该尽可能地作为 MCTF 的前向参考帧。根据提案 JVET-V0056，MCTF 在单个方向的参考帧最多可以有 4 帧，在该限制下，我们使前向参考帧数量可以根据 mini-GOP 的决策情况进行动态调整，即：

$$\text{前向参考帧数} = \min\{4, \text{mini-GOP 中的 B/b 帧数量}\}$$

而对当前 mini-GOP 中的 I/i/P 帧进行 MCTF 时，输入队列中的后续帧还没有决定帧类型，即无法提前获知后续 mini-GOP 的结构，因此无法自适应地决定后向参考帧的数量。因此我们将后向参考帧数固定为提案 JVET-V0056 所规定的 4 帧，即：

$$\text{后向参考帧数} = 4$$

3 实验结果

我们使用 HEVC 的通用测试序列，在 X265 提供的预定义配置下对 MCTF 的 BD-rate 表现进行了测试。需要注意的是，计算 PSNR 需要使用未经 MCTF 滤波的真正的原始帧。

表 1 是 ultrafast 配置下，各序列的测试结果，BD-rate 平均可以下降 4.34%，其中在 BQTerrace 序列上可以达到惊人的 15.75%，这切实地证明了 MCTF 在提升压缩效率上的有效性。

表 1 各序列的 BD-rate 表现（ultrafast）

Class	Sequence	Y	U	V	Average
Class A	Traffic	-6.14%	-5.34%	-5.34%	-6.01%
	PeopleOnStreet	-1.32%	-1.76%	-1.97%	-1.40%
Class B	Kimono	-1.14%	-1.46%	-1.64%	-1.21%
	ParkScene	-4.02%	-3.90%	-5.28%	-4.11%
	Cactus	-11.11%	-16.61%	-9.18%	-11.26%
	BasketballDrive	-3.02%	-1.86%	-3.04%	-2.91%
	BQTerrace	**-15.80%**	**-14.91%**	**-16.40%**	**-15.75%**
Class C	BasketballDrill	-1.19%	-2.39%	-2.54%	-1.46%
	BQMall	-2.48%	-2.44%	-1.73%	-2.40%
	PartyScene	1.38%	-0.79%	-1.64%	0.95%
	RaceHorses	-2.57%	-2.20%	-3.28%	-2.60%
Class D	BasketballPass	-0.53%	-1.47%	-1.06%	-0.67%
	BQSquare	2.61%	-1.87%	-3.16%	1.92%
	BlowingBubbles	-1.73%	-2.29%	-2.92%	-1.89%
	RaceHorses	-0.85%	-0.72%	-0.88%	-0.83%
Class E	FourPeople	-8.93%	-8.98%	-8.12%	-8.86%
	Johnny	-10.68%	-11.57%	-13.95%	-11.12%
	KristenAndSara	-8.39%	-9.01%	-9.28%	-8.53%
Overall		**-4.22%**	**-4.98%**	**-5.08%**	**-4.34%**

另外，我们还选取了其它几个预设配置进行了测试，相关结果如表 2 所示。

表 2 各 Preset 的 BD-rate 表现

Preset	Y	U	V	Average
ultrafast	-4.22%	-4.98%	-5.08%	-4.34%
veryfast	-2.41%	-3.51%	-3.49%	-2.61%
fast	-2.28%	-3.57%	-3.50%	-2.51%
slow	-2.42%	-3.93%	-4.05%	-2.71%
slower	-2.08%	-3.69%	-3.91%	-2.40%

在 veryfast、fast、slow 和 slower 的配置下，BD-rate 可降低 2.6%左右，稍次于 ultrafast 配置下的表现。这是因为 ultrafast 配置的工具集比较少，压缩效率较低，因此 MCTF 的引入能够使压缩性能得到更显著的提升。而其余配置下，由于开启了更多的编码工具，本来就具有比较好的压缩性能，因此 MCTF 带来的压缩性能提升效果会稍有削弱。

为了更清晰地显示 MCTF 的效果，我们给出了 BQTerrace 和 Johnny 序列在 3 种配置下的 R-D（Rate-distortion）曲线，如图 1 所示。可见，在高码率配置下，MCTF 带来的编码效率提升更加明显。

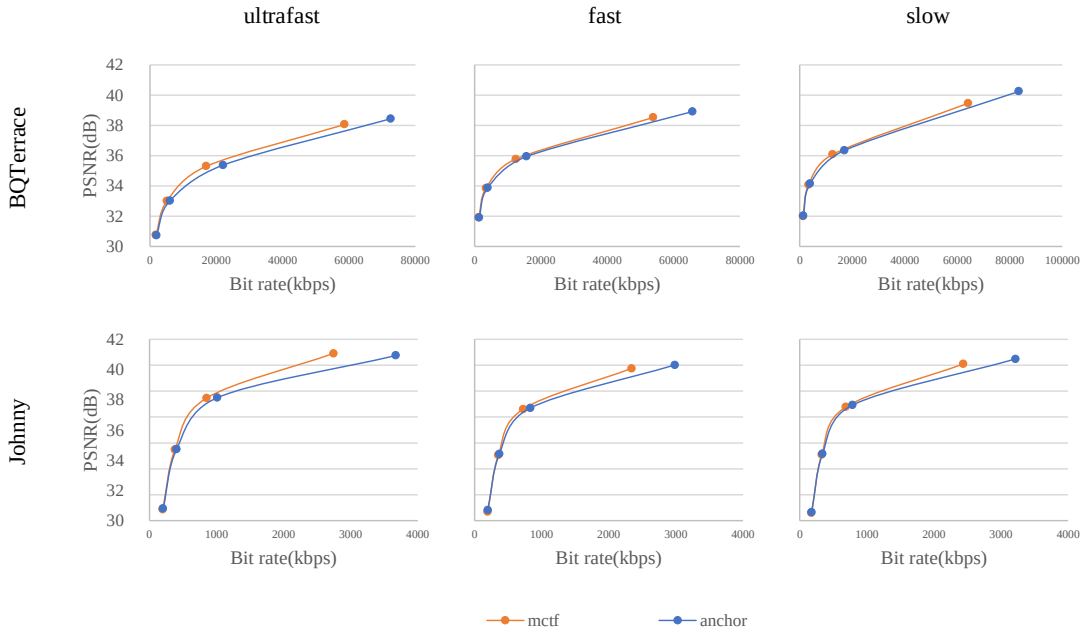

图 1 序列 BQTerrace 和 Johnny 的 R-D 曲线

4 展望

目前我们的 MCTF 仍然采用 JVET-V0056 提案所推荐的固定滤波权重。而对于 X265 这种具有灵活帧结构的软件，可以利用 mini-GOP 结构对滤波权重进行自适应调整，或许可以进一步提升压缩效率。

三、X265-3.5 版本测试结果

为了进一步验证所设计方法的有效性，我们在 X265-3.5 上也进行了测试。整体的实验表现和 X265-3.2 是类似的，这也说明了该算法具有很好的鲁棒性。具体结果如下：

1 快速算法设计

将算法应用到 x265 3.5 上进行测试。测试结果如图 1 和表 2 所示。可以看出，在 x265 3.5 上该算法具有同样的效果。与原始编码器相比提升大概在 10%~20%左右。在相同档次下编码时间能相对减少 31%~41%，而 BD Rate 也会相应相对增加 1.47~2.73%。

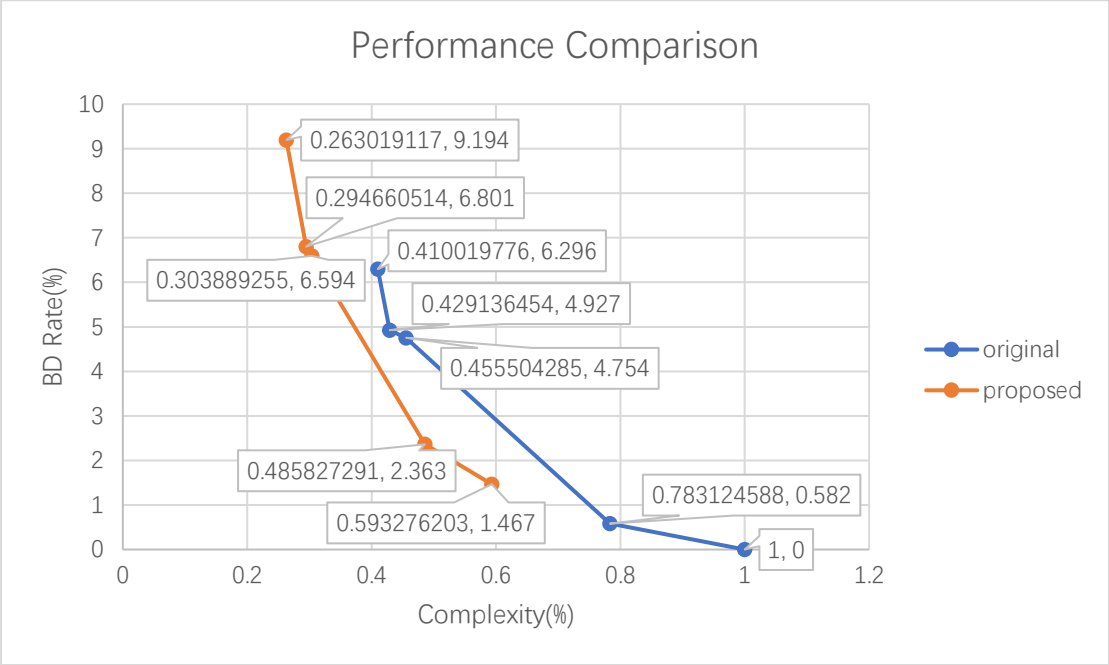

图 1 RD-复杂度曲线表示

表 2 不同档次下复杂度降低和压缩效率变换表现

档次	编码时间相对减少量	BD Rate 相对增加量
superfast	35.90%	2.73%
fast	31.30%	1.79%
medium	33.30%	1.76%
slow	38.00%	1.77%
veryslow	40.70%	1.47%

2 高性能算法设计

将算法应用到 x265 3.5 上进行测试，取得了和 X265-3.2 版本上一致的压缩性能表现。

Preset	**Y**	**U**	**V**	**Average**
ultrafast	-4.22%	-4.98%	-5.08%	-4.34%
veryfast	-2.41%	-3.51%	-3.49%	-2.61%
fast	-2.28%	-3.57%	-3.50%	-2.51%
slow	-2.42%	-3.93%	-4.05%	-2.71%
slower	-2.08%	-3.69%	-3.91%	-2.40%

四、附录

1 代码

版本库：https://github.com/chaoliu18/X265

V3.2 版本

X265-Anchor 代码：master 分支

X265-快速算法版本：optimizedIntra/h 分支

X265-高性能版本：lsr/mctf 分支

V3.5 版本

X265-Anchor 代码：master3.5 分支

X265-快速算法版本：3.5/optimizedIntra/h 分支

X265-高性能版本：3.5/lsr/mctf 分支

运行方法：

切换到目标分支：$ git checkout target_branch

回归测试：$ make run

2 参考文献

[1]. Min B, Cheung R C C. A fast CU size decision algorithm for the HEVC intra encoder[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2014, 25(5): 892-896.

[2]. JVET-O0549, “Encoder-only GOP-based temporal filter”.

[3] JVET-P0328, “Performance of the GOP-based temporal filter in VTM-6.1”.

[4] JVET-V0056, “GOP-based temporal filter improvements”.

[5] Algorithm description for Versatile Video Coding and Test Model 16 (VTM16).

[6] J. Enhorn, R. Sjöberg and P. Wennersten, "A Temporal Pre-Filter For Video Coding Based On Bilateral Filtering," 2020 IEEE International Conference on Image Processing (ICIP), 2020, pp. 1161-1165, doi: 10.1109/ICIP40778.2020.9191359.