Intra Line Copy for HEVC Screen Content Intra-Picture Prediction

Chun-Chi Chen and Wen-Hsiao Peng, *Senior Member, IEEE*

***Abstract*— This paper presents an intra line copy (ILC) technique for HEVC screen content coding. It shares the same origin with two other prominent techniques, intra string copy (ISC) and intra block copy (IBC), in applying the notion of string matching to intra-frame coding. This work combines their merits in one scheme with both good compression performance and high regularity. Specifically, it forms a prediction of a coding block by decomposing it into horizontal or vertical lines of pixels and performing line-based predictions based on previously coded data from the current frame. To address the massive amounts of search operations, our fast search algorithm first searches in the horizontal and vertical directions, then checks line vector candidates from spatial and temporal neighbors, and finally references lines having an identical hash value as the prediction line. The resulting line vectors are further predicted adaptively to minimize their coding overhead. Extensive experiments based on SCM-4.0, which includes IBC as an integral component, show that ILC can provide an additional 4%–7% BD-rate savings when the search area extends to the entire frame and 3%–4% improvements with a restricted local search. Compared with ISC, it achieves comparable performance without all its complications from sequential string parsing.**

***Index Terms*— Intra line copy (ILC), intra string copy (ISC), nonsquare intra block copy (IBC), screen content coding (SCC).**

I. Introduction

SCREEN content video is an emerging video type arising from the need to transmit screen visuals between devices in the form of video for applications such as wireless display, screen sharing/collaboration, and cloud/Web gaming. Such content, usually composed of text, graphics, and nature scene images, exhibits characteristics very different from those of camera-captured content. For example, repeating patterns, one-pixel-wide lines, and sharp edges are among its unique features. These types of signals, which are uncommon in camera-captured content, make conventional video coding methods extremely inefficient for coding screen content [23]. In some cases, a simple lossless file/data compression scheme can even achieve a bit-rate comparable to that of state-of-the-art video codecs operating in lossy mode.

In a joint effort to enhance the capability of High Efficiency Video Coding (HEVC) [20] in coding screen content, the ISO/IEC Moving Picture Experts Group (MPEG) and the ITU-T Video Coding Experts Group (VCEG) standardization organizations have been working together to develop screen content coding (SCC) extensions [1] since April 2014. During the development of SCC, an intra-coding technique known as pseudo 2D matching (P2M) [14], [15] was found particularly effective in addressing the repeating patterns, and laid the foundation for several other tools. Basically, it extends the Sliding Window Lempel-Ziv (LZ) algorithm [4] for file/data compression to image coding. The principle of the LZ algorithm is string matching. To encode a sequence of source symbols, it sequentially parses them into strings of variable length and encodes each string with a pointer-length pair indicating where an identical string can be found in the search window and how long it is. In P2M, the symbol is an image pixel and the sequence of symbols is formed by visiting consecutive 2D blocks of pixels through horizontal or vertical scanning. P2M owes its name to this 2D-to-1D conversion. Although straightforward, it can already capture most of the redundancy inherent in the repeating nature of screen content, justifying its rather good performance.

There have been many improvements and variations around this string matching notion. For example, the reconstruction-based string matching [10], [11] turns the P2M into real 2D matching by allowing a 2D string to be freely searched in a previously coded area within the same frame. By doing so, the need for storing the image in 1D format is avoided. Based on the same matching scheme, the intra string copy (ISC) [16], [24] further relaxes the matching criterion to allow nonperfect matching for lossy compression. Both schemes require a 2D vector to indicate the position of a matching string, in a way very much like identifying a reference block for motion-compensated prediction. This motivates the intra block copy (IBC) [3], [7], [17], which turns string matching into a regular block-based operation similar to block-based motion estimation except for the use of the current frame as reference. As such, IBC is more parallel friendly and compatible with the mainstream codecs. It, however, lacks the flexibility of ISC in determining the length of a string.

To strike a balance between flexibility and regularity, we propose in this paper an intra line copy (ILC) technique. Specifically, we look at a prediction block as being composed of several (horizontal or vertical) lines of pixels, as depicted in Fig. 1. From this viewpoint, it is seen that a variable-length coding string in ISC can start and end at any position in a line. The string can thus be as short as being only a portion of a line, or as long as spanning across multiple lines (see ISC in Fig. 1). All these lines, integral or fractional, share the same displacement vector. By contrast, a prediction block in IBC can be thought of as a fixed-length string with explicit end points that groups together all its composing lines as a whole

Manuscript received October 18, 2015; revised January 2, 2016; accepted January 28, 2016. Date of publication March 16, 2016; date of current version June 30, 2017. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant 104-2221-E-009-065-MY3, and in part by the NOVATEK Fellowship. This paper was recommended by Associate Editor Y. Wang.

The authors are with the Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: cheerchen.cs98g@g2.nctu.edu.tw; wpeng@cs.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2016.2543098

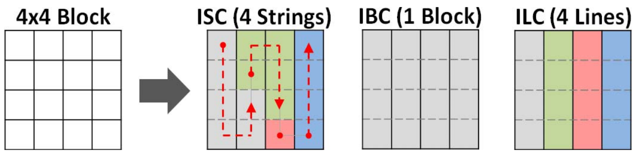

Fig. 1. Partition structures of ILC, ISC, and IBC.

for matching (see IBC in Fig. 1). As an approach in between these two extremes, our ILC tries to keep both the flexibility of ISC for better compression performance and the regularity of IBC for lower complexity. To this end, we consider each line of pixels in a prediction block a basic matching unit. Due to their uniform length and explicit end points, the matching for these lines can run independently and in parallel. Furthermore, by controlling their displacement vectors, ILC can produce a prediction effect similar to that of ISC.

To address the dramatic increase in both search operations and the number of vectors due to the line-based prediction, we propose a fast line vector search algorithm and an adaptive line vector prediction scheme. The former involves three major search strategies: 1) searching horizontally and vertically in a 1D manner; 2) checking vectors from spatially or temporally neighboring blocks/lines; and 3) constructing hash tables to quickly identify repeating patterns. The latter predicts a line vector adaptively from those for the spatially neighboring blocks/lines.

Extensive experiments show that on top of SCM-4.0 [22], ILC can provide an additional 4%–7% BD-rate savings when the search area extends to the entire frame and 3%–4% improvements with local 4-CTU (coding tree unit) search. Interestingly, ILC can largely compensate for the loss of IBC from performing local search, making the combination of ILC and IBC with local search an attractive alternative to full-frame IBC. It is generally believed that full-frame search may be prohibitive in practical applications. Compared with ISC, ILC shows similar performance and runtime characteristics. It is, however, able to reuse many existing components in HEVC and has proved to be as effective as ISC without all the complications from sequential parsing.

The rest of this paper is organized as follows. Section II reviews ISC and IBC. Section III analyzes the block distribution of IBC, which motivates the design of ILC. Section IV presents the line vector search algorithm and the coding scheme. Section V evaluates the compression performance and the complexity of ILC and provides comparisons with ISC. Section VI concludes this paper with a summary of our observations.

II. Related Work

A. Intra String Copy

ISC [16], [24] is an intra-coding technique for SCC. It drew much attention during the development of SCC [6] because of its promising compression performance. Inspired by the LZ coding, it forms an intra prediction of a coding unit (CU)[1] through string matching. Specifically, the process includes the following:

[1]CU is the basic coding unit in HEVC [20], which is similar to the notion of macroblock in Advanced Video Coding, but is of variable size. For encoding, each frame is divided uniformly into CTUs, the size of which is typically 64 × 64. Each CTU may be further divided into CUs by quadtree splitting.

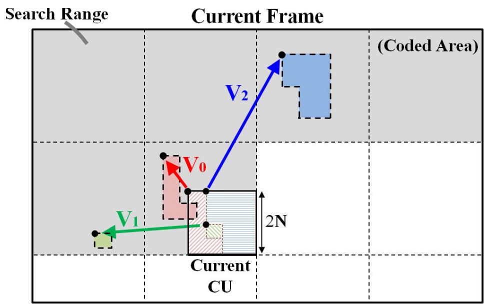

Fig. 2. Illustration of ISC.

1) sequentially parsing the pixels in a CU in traversal order into disjoint strings, each being the longest string of pixels that can find a match in a search area within the current frame;
2) signaling each string with a displacement vector indicating the beginning of the match and a length value representing the length of the match. In particular, ISC adopts lossy matching; that is, any 2 pixels with their difference smaller than a quantization parameters (QP)-dependent threshold value are considered identical.

Another feature to be noted is that it requires no transform and residual coding. Thus, it supports the so-called In-CU reference [24], which allows a string to be copied from another string that may overlap partially or fully the current CU. One such string is indicated by the vector V_0 in Fig. 2.

B. Intra Block Copy

IBC (Fig. 3) is conceptually a simplified version of ISC. Instead of sequentially parsing pixels in a CU into strings of variable length, it performs block-based string matching, requiring that each string have the same shape and size of a prediction unit (PU).[2] Essentially, this makes the operation of IBC almost identical to motion-compensated prediction, except for using the current frame as reference. Since the chance of having an exact match between two fix-sized blocks is usually small, it signals the prediction residual. In particular, the same residual coding as for inter prediction is used. Owing to the block-based transform, for the prediction loop to be closed only the reconstructed pixels outside of the current CU can be referenced, and thus In-CU reference is prohibited.

Given its high compatibility with the existing state-of-the-art coding architectures, IBC has been an integral part of the SCC standard [6] since the 17th Joint Collaborative Team on Video Coding (JCT-VC) meeting and is one of the most powerful coding tools in this standard [21]. Currently, the search for IBC can cover the coded area within the current frame partially due to the wavefront parallel processing constraint [9], [19]. Therefore, IBC cannot access pixels in the CTUs (i.e., outside of the area marked using thick dashed line in Fig. 3) next to those lying along the up-right diagonal direction relative to the current one in each CTU row above.

III. Lines and Nonsquare Partitions

There are some interesting observations about IBC. Fig. 4 provides a breakdown analysis of the partition types of IBC

[2]PU is the basic prediction unit in HEVC, which can have a size of $2N \times 2N$, $2N \times N$, $N \times 2N$, or $N \times N$.

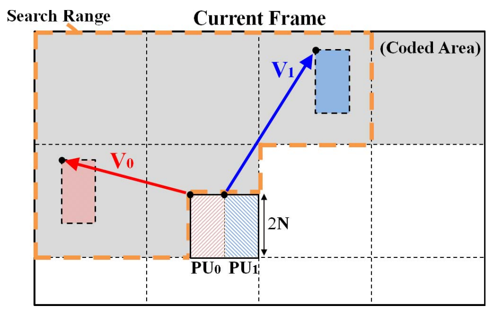

Fig. 3. Illustration of IBC.

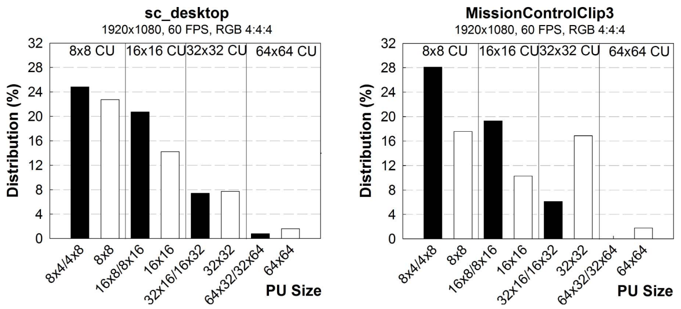

Fig. 4. Distributions of IBC block sizes with QP = 37.

blocks in terms of their spatial coverage, namely, the percentage of pixels coded in individual types. The results are based on the 1-second encoding of two standard sequences, sc_desktop and MissionControlClip3,[3] using SCM-4.0 [22] under the all-intra (AI) configuration. In particular, we turn off all fast algorithms[4] for IBC, so that the results are not biased.

From Fig. 4, IBC tends to work more efficiently with smaller partitions and nonsquare structures. These phenomena are, however, not seen in coding camera-captured content. The reason may be explained by the unique signal characteristics of screen content, which is usually full of sharp edges and thin lines. For these types of signals, a slight mismatch in phase between a prediction block and its reference could easily incur considerable prediction errors. Choosing a smaller block size seems a logical step toward improving prediction accuracy. The fact that these sharp edges and thin lines are often horizontally or vertically positioned partly justifies the more frequent use of nonsquare partitions. These observations motivate our radical attempt of introducing lines as basic prediction units. Although the overhead for signaling vectors for these extremely small partitions may seem to outweigh their prediction benefit, our analysis in Section V-A surprisingly indicates the opposite.

IV. Intra Line Copy

In this section, we present the notion of ILC, a fast algorithm for line vector search, and a predictive line vector coding scheme.

[3]The sc_desktop sequence is a purely screen content sequence, while MissionControlClip3 is a mixture of screen content and camera-captured content.

[4]Exhaustive and full-frame search is applied to all PU sizes and the 1D search decision at CU level is disabled; both pixel gradient and R-D cost-based algorithms for early termination are disabled; an early termination algorithm for CU splitting based on IBC's root cbf flag is also disabled.

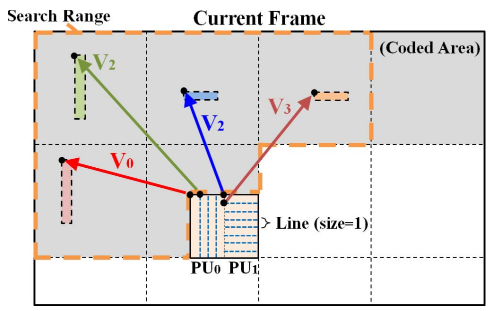

Fig. 5. Illustration of ILC.

A. Concept of Operations

ILC extends the notion of IBC by allowing a PU to be further divided uniformly into 1-pixel-thin horizontal or vertical lines for prediction. According to the dividing orientation, these lines have the same width or height as their parent PU, and each has its own line vector indicating where in the current frame it is predicted from. For example, as depicted in Fig. 5, an $N \times 2N$ PU can be divided into lines of $N \times 1$ when split rowwise or of $1 \times 2N$ when split columnwise, and these lines can be separately predicted from the reconstructed region in the current frame (the gray area). As with IBC, the residuals of ILC are transformed, quantized, and entropy coded, so exact match is not a necessary requirement for lines. Also, because of the block-based transform, none of the lines within the current CU can reference pixels in the same CU (i.e., prohibiting In-CU reference).

To determine the best mode for each PU, the sum of absolution prediction distortion and the bit overhead for the splitting direction and line vectors are considered in the computation of the PU-level R-D cost. The one with the lowest PU-level R-D cost among vertical ILC, horizontal ILC, and IBC is selected. Once each PU mode is determined and the associated CU is finished encoding, the resulting CU-level R-D cost is used for comparison with the other prediction modes.

B. Fast Algorithm for Line Vector Search

This section introduces a fast line vector search algorithm for ILC. This algorithm comprises three major parts:

1) 1D search, which searches horizontally and vertically within a given search area;
2) candidate-based search, which tests candidate vectors derived from spatially or temporally neighboring blocks/lines, and two most recently coded vectors;
3) hash-based search, which aims to test through a specific set of reference lines whose source signals match that of the prediction line.

1) 1D Search: Screen objects, such as tables, boxes, and text, are usually aligned with respect to a grid-like structure when displayed. For instance, letters in the text are often aligned horizontally, and some other objects may be aligned vertically. This salient feature of screen content suggests a 1D search along the horizontal and vertical axes.

To justify the effectiveness of this search pattern, Fig. 6(a) shows how ILC's line vectors are distributed in the vector field. The results are obtained by conducting an exhaustive

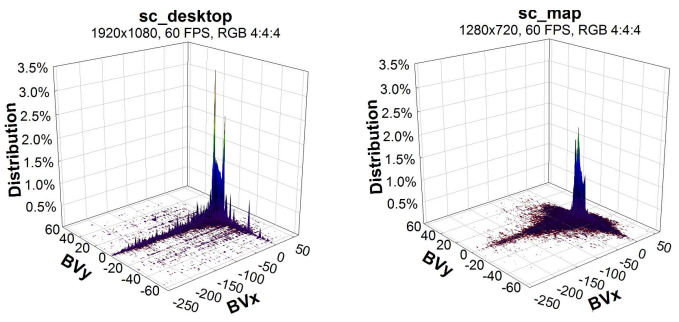

Fig. 6. Line vector distributions of (a) sc_desktop and (b) sc_map.

TABLE I
PERCENTAGE OF 1D LINE VECTORS

All Intra	TGM		MC	
	QP22	QP37	QP22	QP37
Min.	21%	43%	45%	51%
Max.	61%	61%	56%	59%
Avg.	50%	54%	52%	55%

TABLE II
PERCENTAGE OF LINE VECTORS MATCHING THE EXHAUSTIVE SEARCH RESULTS

All Intra	TGM		MC	
	QP22	QP37	QP22	QP37
Min.	44%	72%	69%	76%
Max.	80%	81%	76%	78%
Avg.	72%	78%	73%	77%

search on Class TGM and MC test sequences.[5] As expected, many line vectors have either a nonzero x-component or a nonzero y-component, but not both. This indicates that they are pointing in the horizontal or vertical direction. Table I further analyzes such 1D line vectors in percentage terms for TGM and MC sequences. As shown, about 50%–55% of line vectors in these sequences are 1D. There are, however, exceptions. One example is the sc_map sequence in Class TGM, for which the line vector distribution as shown in Fig. 6(b) is less concentrated along the horizontal and vertical axes. Thus, more candidates must be checked.

2) Candidate-Based Search: Usually, spatial or temporal neighboring lines/blocks have identical line/block vectors. An effective yet simple strategy to speed up the search process is to include their vectors as possible candidates. In some cases where their vectors are not available for reference, we refer to the two most recently coded block/line vectors. Similar ideas have been adopted by SCM-4.0 [22] to accelerate the search process for IBC [8].

Table II indicates that 70%–80% of the line vectors obtained by these simple strategies along with the 1D search match the results of the optimal exhaustive search. Comparing Tables I and II, we see that the candidate-based scheme helps identify another ∼20% of line vectors that are not detectable by the 1D search pattern. In the next section, we shall proceed with a more complicated search method to approach the remaining line vectors.

[5]In [22], 26 video sequences are specified for the development of the SCC standard [6]. They are grouped into four classes according to their content types, with Class TGM comprising 14 sequences of pure screen content and Class MC six sequences with a mixture of screen and camera-captured content.

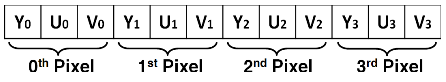

Fig. 7. Input format to the CRC computation.

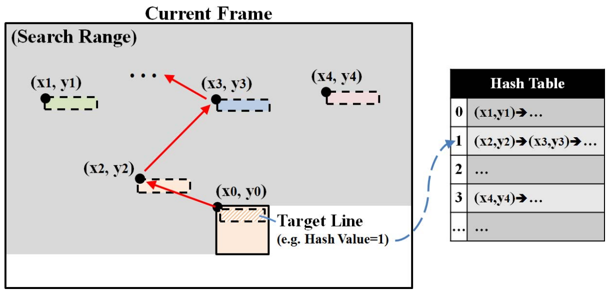

Fig. 8. Illustration of the hash-based search for ILC.

3) Hash-Based Search: Performing an exhaustive search for block, string, or line matching can be time consuming, particularly when the search range is extended to full frame. To explore the potential of full-frame IBC, SCM-4.0 adopts a hash-based search scheme [12], which trades hash tables (memory space) for computational time to approach the performance of the optimal exhaustive search. The same strategy was also applied to inter prediction and ISC [16] during the development of SCC for a fair experimental setup. By the same token, this section presents our hash technique for ILC.

The principle of hash-based search for ILC is to confine the search only to those candidate lines that have the same hash value as the prediction line. The process relies on computing a hash value for every line starting at every permissible search location. This is achieved by computing the cyclic redundancy check (CRC) [25] code. As shown in Fig. 7, we take as input the cascaded binary representations of a line of pixels, and compute its CRC code by returning the remainder after this input number is divided by a predefined divisor. The operation is applied in the source signal domain (instead of the reconstruction signal domain used for 1D and candidate-based search methods). After the hash code for every line is collected, a hash table is used to store the locations of lines sharing identical hash values, as delineated in Fig. 8. For fast line matching, only those locations where the reference lines have the same hash value as the prediction line will be checked. In particular, when a hash value is computed with respect to every line at every pixel position, this yields an effect similar to performing full-frame ILC.

In computing the CRC code, the choice of the divisor and the form of the input critically affect the search runtime and quality. For example, the larger the divisor, the more diversified values the remainder can take on. From Fig. 8, this demands a hash table with more entries and on average fewer candidate lines in each. The smaller candidate set implies a reduced search runtime. By contrast, the input representation can significantly influence the search quality. It is noticed that the CRC code can easily detect any two lines that do not match perfectly in the source domain. While this property is desirable

TABLE III

CODING PERFORMANCE AND RUNTIME COMPARISONS OF VARIOUS SEARCH METHODS

All Intra		**1-D**	**Cand.**	**Hash**	**1-D, Cand.**	**All**	**1-D**	**Cand.**	**Hash**	**1-D, Cand.**	**All**	**Exhaust.**
Search Range		**Full-frame**					**2-CTU**					
Lossy	**TGM$_{RGB}$**	-2.7	-2.1	-3.1	-4.5	-6.7	-2.2	-1.6	-1.9	-2.8	-3.8	-4.1
	MC$_{RGB}$	-2.6	-2.1	-4.2	-4.6	-7.6	-2.0	-1.2	-1.1	-2.5	-2.9	-3.5
	TGM$_{YUV}$	-2.8	-2.0	-3.2	-4.6	-7.0	-2.3	-1.7	-1.9	-2.9	-3.8	-4.3
	MC$_{YUV}$	-2.6	-2.1	-4.1	-4.3	-7.4	-2.2	-1.1	-1.1	-2.6	-2.7	-3.4
Lossless	**TGM$_{RGB}$**	-2.6	-2.3	-3.0	-3.9	-6.1	-2.3	-1.0	-1.6	-2.6	-3.8	-3.9
	MC$_{RGB}$	-1.6	-1.6	-2.2	-2.4	-3.6	-1.3	-0.5	-0.7	-1.4	-1.6	-1.8
	TGM$_{YUV}$	-2.4	-2.4	-3.3	-3.8	-6.4	-2.1	-1.0	-1.8	-2.4	-3.9	-4.0
	MC$_{YUV}$	-1.7	-1.8	-2.6	-2.6	-4.1	-1.4	-0.6	-0.8	-1.5	-1.8	-2.0
Enc. [%]		114	101	106	115	121	110	101	101	111	113	636
Dec. [%]		100	101	100	99	98	100	101	100	100	99	100

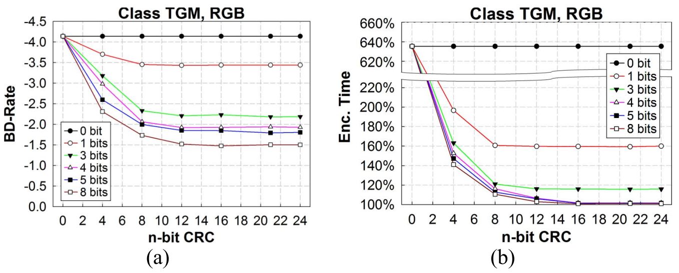

Fig. 9. Effects of the divisor size and the remaining MSB bits on (a) compression performance and (b) encoding runtime.

for string matching schemes that hope to identify perfect matching strings of variable length, it becomes superfluous and detrimental to ILC, in which the approximate matching of fixed-length lines is sought instead. Very often, the overly stringent criterion leads to fewer candidates to be checked. To address this issue, we change the line representation in Fig. 7, rather than turning to other hash functions. Specifically, we involve only the most significant bits (MSB) bits of pixel samples in the CRC calculation, allowing lines with the same MSB bits in their samples to receive an identical CRC code. Here, a uniform MSB truncation scheme is applied to every sample.

To see their combined effects, the encoding time ratios and compression performance relative to SCM-4.0 [22] are shown in Fig. 9(a) and (b), respectively, for different choices of divisor size ($n = 0, 4, \ldots, 24$ b) and the retained MSB bits ($m = 0, \ldots, 8$). In Fig. 9, each curve is plotted with a fixed m over various choices of n. The two special cases with $m = 0$ or $n = 0$ correspond to the exhaustive search.[6] From Fig. 9(a), two observations can be made.

1) For a fixed m, the BD-rate saving drops when the divisor size n increases.
2) For a fixed n, the same trend continues when more MSB bits m are retained.

Both can be explained by our earlier predictions that fewer candidates would be found when the divisor size increases or when the line matching is made more exact. The same line of reasoning can be followed to justify the much reduced search runtime in either case, as illustrated in Fig. 9(b). In this paper, we choose the combination of $n = 16$ and $m = 4$ for its good compression performance and negligible impact on the encoding runtime.

The much reduced runtime of the hash-based search is achieved at the expense of extra memory requirements for storing hash tables. From Fig. 8, the size of such a table for full-frame search can be estimated to be of the same order of magnitude as for storing a reference frame. Essentially, we need to keep the memory address of every line according to their hash values. With the full-frame search, a line can start at any pixel position. Therefore, the number of addresses to be kept is the same as the frame resolution. The caveat is that a separate table is needed for each type of line, e.g., 1×4, 4×1, 1×8, 8×1.

4) Performance and Combination Tests: Table III compares different search methods in terms of their compression performance on SCM-4.0 and provides results for several combination tests. With the full-frame search range, it is no surprise to see that the hash-based scheme achieves the highest gains among all the three methods. However, the more practical approach which combines the 1D and the candidate-based search can reach even higher gains without the costly hash tables. Interestingly, with all the three methods combined, ILC can bring 4%–7% improvements over SCM-4.0 at a 20% increase in encoding time, which gives a rough indication of its full potential. When the search range is limited to 2-CTU, the preferred setting for ILC, the 1D search dominates the gain and encoding time among the others. We also see that the full combination scheme approaches the performance of the optimal exhaustive search with a much reduced encoding time. Note that the exhaustive search results are not provided for the full-frame case since the simulation cannot be completed. With the above observations, we shall adopt the full combination scheme for the rest of the experiments.

C. Predictive Line Vector Coding

Sending displacement vectors at the granularity of a 1-pixel-thin line can be costly. In the worst case, where all the PUs in a frame are of size 8×4 (respectively, 4×8) and are further split into 1×4 (respectively, 4×1) lines for ILC, the number of vectors required for signaling will be increased eightfold compared with that of IBC. Apparently, how to encode these line vectors is vital to the compression performance of ILC.

In this paper, we adopt predictive line vector coding. The line vector difference is coded in the same way as the block vector difference. Particularly, we evaluate three different

[6]Every n-b divisor actually has $n+1$ b in which the leading bit is 1. Setting m equal to 0 means all the MSB bits of pixel samples are omitted from computing the CRC code. In this special case, we simply set the CRC code to zero.

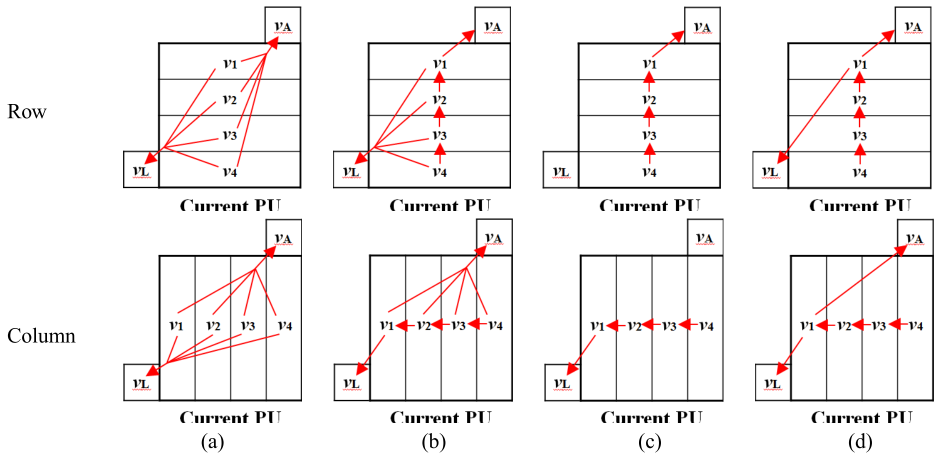

Fig. 10. Illustration of various strategies for the line vector prediction. (a) v_A, v_L. (b) #1. (c) #2. (d) #3.

TABLE IV

COMPARISON OF LINE VECTOR PREDICTION SCHEMES IN TERMS OF THE AVERAGE NUMBER OF CODING BITS PER VECTOR

		sc_desktop	MissionControlClip3
QP22	v_A, v_L	12.4	13.0
	#1	8.0	8.6
	#2	7.8	8.4
	#3	7.4	8.0
QP37	v_A, v_L	12.1	12.3
	#1	7.6	7.8
	#2	7.4	7.5
	#3	6.9	7.1

TABLE V

COMPRESSION PERFORMANCE COMPARISON OF LINE VECTOR PREDICTION SCHEMES

All Intra		v_A, v_L	#1	#2	#3
Lossy	**TGM**$_{RGB}$	-2.6	-3.9	-3.9	-4.1
	MC$_{RGB}$	-2.1	-3.2	-3.3	-3.5
	TGM$_{YUV}$	-2.5	-3.9	-4.0	-4.3
	MC$_{YUV}$	-2.0	-3.0	-3.2	-3.4
Lossless	**TGM**$_{RGB}$	-2.9	-3.8	-3.7	-3.9
	MC$_{RGB}$	-1.4	-1.8	-1.8	-1.8
	TGM$_{YUV}$	-3.1	-3.9	-3.8	-4.0
	MC$_{YUV}$	-1.5	-2.0	-2.0	-2.0

Results shown are based on 1-second encoding

prediction schemes, which differ in their choice of line vector predictors. The baseline for comparison is IBC's block vector prediction scheme, where as shown in Fig. 10(a), it predicts every line vector in a PU from either v_A or v_L—the block/line vector for the spatially neighboring PU, which is to the top right or to the bottom left of the current PU. The choice between the two predictors is made by rate-distortion optimization and is signaled explicitly with a flag. Obviously, this straightforward extension may not be optimized for the line vector prediction. For instance, in predicting v_2, the previously coded v_1 could be a better prediction reference than v_A in the rowwise splitting or v_L in the columnwise splitting, as v_1 is spatially closer to v_2 and may thus correlate more strongly with v_2. As such, in our first test (Method #1), we replace v_A or v_L with the previously coded line vector v_{i-1} in forming the predictor set for all line vectors v_i, except v_1 for which its predictor set remains the same as that for IBC. Our second test (Method #2) further saves the overhead for signaling predictors by keeping for each line only one predictor, as illustrated in Fig. 10(c). The last test (Method #3) is a hybrid of Methods #1 and #2, in which the first line in coding order retains the same predictor set as IBC while the others simply refer to the previously coded line vector.

Table IV presents the coding results associated with these prediction schemes. We re-encode the line vectors generated based on the baseline scheme (v_A and v_L) using Methods #1–#3. Each number in Table IV indicates the average number of bits consumed on coding a line vector. As shown, all the three methods perform better than the baseline, the IBC's block vector prediction scheme. Moreover, it is seen that Method #2 consistently outperforms #1, which can be attributed to the frequent occurrence of choosing previously coded line vector v_{i-1} for prediction. The cost of having to signal the choice does not seem to be necessary for each line vector. In fact, such overhead can justify its compression benefit only when it comes to the coding of v_1, as can be observed by comparing the results of Methods #2 and #3. When integrated into SCM-4.0, all the three methods show about 1% BD-rate savings relative to the baseline method (see Table V) under the AI condition. As expected, Method #3 achieves the highest gain. We thus adopt Method #3 in all the experiments that follow.

V. EXPERIMENTAL RESULTS

This section characterizes the compression performance and the complexity of ILC, and compares its performance/complexity characteristics with ISC. We begin with providing performance results to justify the choice of lines as additional nonsquare prediction units, which is a continuation of our earlier discussion in Section III. We then present performance comparisons between ILC and ISC with the search range configurations, including 2-CTU, 4-CTU and full frame.[7] After that, we analyze the impacts of ILC and ISC

[7]The N-CTU search range comprises the reconstructed region of one current CTU plus a $(N-1)$*64-pixel-wide region to the left, where N is a positive integer; the full-frame search range allows a line or a string to be copied from any position sitting in the reconstructed region of the current frame.

on the decoding complexity. Finally, we discuss IBC with a restricted search range and its combination with ILC.

For experiments, ILC is implemented with SCM-4.0 [22]. To switch it on and off adaptively, one flag is sent for each PU when the reference frame index indicates the current frame as reference. Once this flag is switched on, one additional flag is sent to indicate the splitting direction (i.e., vertical or horizontal) for lines. All the remaining syntax elements are aligned with those of IBC. For ISC, its software package is also built on top of SCM-4.0 and is provided in [16]. All the experiments are conducted following the common test conditions [22] for developing SCC extensions [6], in which each simulation batch involves the encoding and decoding of 26 test sequences in RGB and YUV formats, at five QP values (lossless, 22, 27, 32, and 37), and under three encoder settings [All Intra (AI), Random Access (RA) and Low Delay B (LB)]. These video sequences have picture resolutions ranging from 1280×720 to 2560×1440. According to their content types, they are categorized into four classes—Text and Graphics with Motion (TGM_{RGB} and TGM_{YUV}), Mixed Content (MC_{RGB} and MC_{YUV}), Animation Content (AC_{RGB} and AC_{YUV}), and Camera Content (CC_{RGB} and CC_{YUV}). Class TGM is pure screen content with computer-generated graphics and text only, whereas Class MC is a mixture of screen content and camera-captured content. Classes CC and AC are typical camera-captured content and computer animation, respectively. In particular, results for these two classes are omitted, because none of the methods tested offers any meaningful gain. For reporting compression performance improvements, we adopt the BD-rate [2] saving for lossy coding and the overall bit-rate reduction for lossless coding. In both metrics, negative numbers indicate rate savings/reductions.

A. Revisit of Lines and Nonsquare Partitions

In Section III, we have seen that IBC works most effectively with small block sizes and that nonsquare structures are particularly beneficial to IBC. Our discussion, however, is restricted to the partition structures and sizes supported by the current draft standard specification [6]. In this section, we remove such limitation to provide justification for ILC. Specifically, we allow IBC to go below the minimal allowed partition sizes in SCM-4.0, i.e., 8x4/4x8, and under which case, test both square and nonsquare partitions of various sizes to see which of them performs the best.

Fig. 11 depicts all the partitions tested. They are implemented in such a way that one additional flag is sent for each PU to indicate whether it should be further split. The resulting partitions are thus referred to as sub-PUs. Table VI summarizes the BD-rate savings relative to SCM-4.0 for every sub-PU type when they are enabled separately. The results are based on 1-second encodings with 2-CTU search range for all sub-PUs. Comparing 4x4 and 8x2/2x8 sub-PU types—both when enabled having the same number (3) of sub-PUs in a CU—we again observe that nonsquare partitions bring more compression benefit. Interestingly, the highest gain is achieved when the nonsquare partition degenerates into a line, justifying the use of ILC. A careful examination of Table VI reveals that the 4x1/1x4 line performs slightly better than 8x1/1x8 for

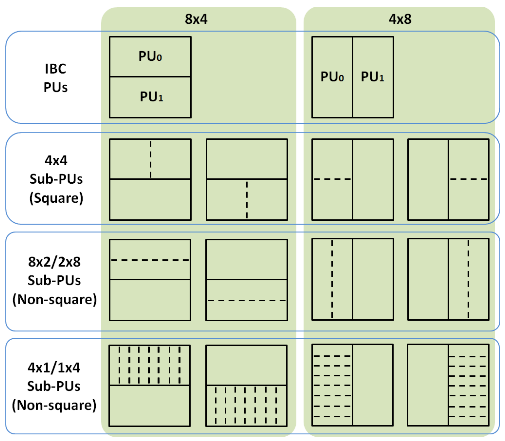

Fig. 11. Examples of square (4×4) and nonsquare ($8 \times 2/2 \times 8$ and $4 \times 1/1 \times 4$) partitions.

TABLE VI
COMPRESSION PERFORMANCE COMPARISON OF PARTITION STRUCTURES

All Intra		4x1	4x2	4x4	8x1	8x2
Lossy	**TGM_{RGB}**	-2.4	-2.2	-1.4	-2.6	-2.3
	MC_{RGB}	-2.0	-1.8	-0.9	-2.3	-1.5
	TGM_{YUV}	-2.2	-2.2	-1.7	-2.6	-2.3
	MC_{YUV}	-1.8	-1.6	-0.9	-2.2	-1.7
Lossless	**TGM_{RGB}**	-2.4	-2.1	-1.4	-2.2	-1.8
	MC_{RGB}	-1.3	-1.1	-0.7	-1.2	-0.9
	TGM_{YUV}	-2.7	-2.4	-1.5	-2.5	-2.0
	MC_{YUV}	-1.5	-1.3	-0.9	-1.5	-1.1

For simplicity, the notation 4x1 represents 4x1/4x1, and so do the others.

lossless coding (0%–0.4%) and worse (0.2%–0.4%) for lossy coding, which agrees with the general observation that smaller partitions usually provide a better R-D tradeoff at high rates.

In Table VII, we present the results for several combinations of nonsquare sub-PUs, aiming to identify a minimum set of sub-PUs that can achieve the highest gain. Starting from combining small sub-PUs (see Part I), it is seen that the combination of 4x1/8x1 achieves the best performance among the others, involving at least one sub-PU type that is not a line. On top of 4x/8x1, additionally applying line splitting to 16x8/8x PUs, which yields 16x1/1x16 lines, can bring up to 0.5% more gain. This approach is also seen to be performing better than adding 16x2/2x16 or 16x4/4x6 partitions (see Part II), although the differences between these three are rather minor. Last, additional sub-PU splitting at even larger PU sizes shows nearly no extra gain (see Part III). In view of these, we shall adopt only 4x1/1x4, 8x1/1x8, and 16x1/1x16 lines for the rest of our experiments.

B. Compression Performance

This section analyzes the performance of ILC by showing its rate savings over SCM-4.0 and providing comparisons with ISC [16]. The results are summarized in Table VIII.

Overall, they perform close to each other in the lossy cases. ILC shows slightly better results in MC sequences, while ISC is better performing in TGM sequences. They both benefit similarly from the increase in search range. Under the AI condition, enlarging the search range from 2-CTU to 4-CTU can provide, on average, $\sim$1% more rate savings across all

TABLE VII

COMPRESSION PERFORMANCE COMPARISON OF VARIOUS COMBINATIONS OF NONSQUARE PARTITIONS

All Intra		Part I				Part II			Part III		
		4x1/8x1	**4x1/8x2**	**4x2/8x1**	**4x2/8x2**	**16x1**	**16x2**	**16x4**	**32x1**	**32x2**	**32x4**
		-				w/ 4x1, 8x1			w/ 4x1, 8x1, 16x1		
Lossy	**TGM$_{RGB}$**	-4.0	-3.7	-4.0	-3.3	-4.1	-4.1	-4.1	-4.1	-4.1	-4.1
	MC$_{RGB}$	-3.3	-2.9	-3.3	-2.5	-3.5	-3.5	-3.5	-3.5	-3.5	-3.5
	TGM$_{YUV}$	-3.9	-3.7	-3.9	-3.6	-4.3	-4.2	-4.2	-4.2	-4.2	-4.2
	MC$_{YUV}$	-3.2	-2.8	-3.2	-2.5	-3.4	-3.3	-3.2	-3.4	-3.4	-3.3
Lossless	**TGM$_{RGB}$**	-3.4	-3.2	-3.1	-2.6	-3.9	-3.8	-3.8	-3.9	-3.9	-3.9
	MC$_{RGB}$	-1.7	-1.5	-1.6	-1.3	-1.8	-1.8	-1.8	-1.8	-1.8	-1.8
	TGM$_{YUV}$	-3.9	-3.7	-3.6	-2.9	-4.0	-4.0	-3.9	-4.0	-4.0	-4.0
	MC$_{YUV}$	-2.0	-1.9	-2.0	-1.6	-2.0	-2.0	-2.0	-2.1	-2.1	-2.1

For simplicity, the notation 4x1 represents 4x1/4x1, and so do the others.

TABLE VIII

COMPRESSION PERFORMANCE AND RUNTIME COMPARISONS OF ILC AND ISC

		ILC			ISC			ILC			ISC			ILC			ISC		
		AI	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**
Search Range		**2 CTUs**						**4 CTUs**						**Full Frame**					
Lossy	**TGM$_{RGB}$**	-3.7	-2.1	-1.1	-3.8	-2.5	-2.0	-4.6	-2.6	-1.4	-4.6	-3.1	-2.0	-6.8	-4.0	-2.1	-6.6	-4.5	-3.3
	MC$_{RGB}$	-2.3	-1.4	-0.5	-1.2	-0.7	0.1	-2.6	-1.7	-0.5	-1.4	-0.8	0.0	-4.2	-2.5	-1.2	-2.1	-1.2	-0.4
	TGM$_{YUV}$	-3.8	-2.0	-1.0	-4.2	-2.4	-1.6	-4.6	-2.5	-1.1	-4.9	-2.9	-1.9	-6.9	-3.9	-1.9	-6.8	-4.3	-2.7
	MC$_{YUV}$	-2.2	-1.4	-0.5	-1.5	-0.9	-0.3	-2.6	-1.7	-0.4	-1.8	-1.1	-0.3	-4.1	-2.8	-0.9	-2.5	-1.6	-0.5
	Avg.	**-3.0**	**-1.7**	**-0.8**	**-2.7**	**-1.6**	**-0.9**	**-3.6**	**-2.1**	**-0.9**	**-3.1**	**-2.0**	**-1.1**	**-5.5**	**-3.3**	**-1.5**	**-4.5**	**-2.9**	**-1.7**
Lossless	**TGM$_{RGB}$**	-3.7	-2.4	-1.9	-5.6	-3.5	-2.8	-4.3	-2.9	-2.3	-6.6	-4.3	-3.3	-6.1	-4.2	-3.3	-8.9	-6.1	-4.9
	MC$_{RGB}$	-1.2	-0.2	-0.1	-0.9	-0.1	-0.1	-1.3	-0.2	-0.1	-1.2	-0.2	-0.1	-2.1	-0.4	-0.2	-1.8	-0.3	-0.1
	TGM$_{YUV}$	-3.8	-2.4	-1.8	-6.0	-3.9	-3.1	-4.5	-2.9	-2.2	-7.1	-4.7	-3.7	-6.6	-4.3	-3.3	-9.5	-6.5	-5.3
	MC$_{YUV}$	-1.3	-0.3	-0.1	-1.0	-0.1	-0.1	-1.5	-0.3	-0.2	-1.2	-0.2	-0.1	-2.3	-0.4	-0.2	-2.0	-0.3	-0.1
	Avg.	**-2.5**	**-1.3**	**-1.0**	**-3.4**	**-1.9**	**-1.5**	**-2.9**	**-1.6**	**-1.2**	**-4.0**	**-2.3**	**-1.8**	**-4.3**	**-2.3**	**-1.8**	**-5.6**	**-3.3**	**-2.6**
Enc. [%]		114	99	99	113	102	103	114	100	98	117	105	103	124	102	100	126	110	105
Dec. [%]		102	102	103	101	103	103	101	102	103	101	101	103	102	102	102	101	103	103

sequences, and extending further from 4-CTU to full frame brings another 2% gain, yet at the cost of an almost 10% increase in encoding time. For the RA and LB conditions, the gains are relatively smaller since the inter-prediction tools can dominate. A side experiment indicates that increasing the search range not only allows more candidates to be searched but also increases the chance of longer lines/strings being selected, which agrees with the theoretical prediction [4].

In the lossy cases, it is interesting to note that ISC outperforms ILC in TGM sequences by a significant margin. This has to do with two of its unique features, namely, the flexibility of choosing short strings and the support of In-CU reference, as was pointed out in [13]. It was found that the percentage of short strings (of length 4 pixels, the minimum line size in ILC) increases dramatically when ISC is operated at higher rates or in lossless mode. Intuitively, short strings (small partitions) have an edge over long strings (large partitions) in providing better prediction at high rates. Moreover, when the string becomes shorter, it is more likely to find a match nearby—that is, there is no need to look further back in the buffer. Its ability to support In-CU reference happens to provide the required function, allowing a string to be copied immediately from those that have just been coded, even when they are all within the same CU. It is, however, noticed that both short strings and In-CU reference have negative impacts on complexity. This will become more apparent in Section V-C. The results in Table IX show that turning off either of them or both will cause the performance of ISC to drop considerably, particularly in TGM sequences and in lossless conditions, where ISC used to deliver a much better performance.

C. Memory Access Bandwidth

One major source of complexity for implementing tools like motion-compensated prediction and IBC is the need to access reference pixels in the previously decoded pictures or the current picture, which is usually stored in the external memory due to the considerable costs needed for buffering them on a chip or with cache. The bandwidth consumed for the required memory access can thus serve as an indication of complexity, which reflects the practicality of a tool in terms of its memory access characteristics. Here, we follow the practices of the JCT-VC committee to characterize the runtime [18] and per-pixel memory access bandwidth [5] for ILC and ISC.

1) Runtime Behavior Analysis: This section compares the runtime memory access bandwidths of ISC and ILC at the decoder to provide a rough indication of their complexity characteristics. The software [18] that was used by the JCT-VC committee to measure the runtime memory bandwidth is integrated into ILC and ISC. As shown in Fig. 12, a memory model, which is assumed in the software, divides a reference frame into equal-sized nonoverlapping memory blocks of size $m \times n$ (e.g., 4×2), and then these blocks are stored in raster order into a physical memory at consecutive addresses. Each of such blocks serves as the basic unit for data access, with each read from or write to the memory further restricted by the memory alignment and burst size. The former specifies the minimum addressable unit while the latter refers to the data size per memory access. Apparently, the memory configuration has a critical impact on the efficiency of data access. For example, in Fig. 12, to retrieve the reference block of size $M \times N$ (e.g., 2×4), a total of 12 memory blocks have to be

TABLE IX
COMPRESSION PERFORMANCE COMPARISON OF ILC AND ISC WITH RESTRICTIONS ON STRING LENGTH AND/OR IN-CU REFERENCE

		ILC			ISC			ISC w/o L<4			ISC w/o In-CU			ISC w/o both		
		AI	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**	**AI**	**RA**	**LB**
Search Range		**2 CTUs**														
Lossy	**TGM$_{RGB}$**	-3.7	-2.1	-1.1	-3.8	-2.5	-2.0	-3.3	-2.2	-1.6	-1.8	-1.3	-0.8	-1.6	-1.1	-0.6
	MC$_{RGB}$	-2.3	-1.4	-0.5	-1.2	-0.7	0.1	-1.0	-0.6	-0.2	-0.5	-0.3	0.1	-0.4	-0.4	0.0
	TGM$_{YUV}$	-3.8	-2.0	-1.0	-4.2	-2.4	-1.6	-3.7	-2.2	-1.5	-1.9	-1.2	-0.7	-1.7	-1.1	-0.6
	MC$_{YUV}$	-2.2	-1.4	-0.5	-1.5	-0.9	-0.3	-1.3	-0.8	-0.2	-0.7	-0.6	-0.3	-0.7	-0.6	-0.3
	Avg.	**-3.0**	**-1.7**	**-0.8**	**-2.7**	**-1.6**	**-0.9**	**-2.3**	**-1.5**	**-0.8**	**-1.2**	**-0.9**	**-0.4**	**-1.1**	**-0.8**	**-0.4**
Lossless	**TGM$_{RGB}$**	-3.7	-2.4	-1.9	-5.6	-3.5	-2.8	-3.9	-2.4	-1.8	-3.6	-2.4	-2.0	-2.2	-1.5	-1.2
	MC$_{RGB}$	-1.2	-0.2	-0.1	-0.9	-0.1	-0.1	-0.6	-0.1	0.0	-0.7	-0.1	-0.1	-0.4	-0.1	0.0
	TGM$_{YUV}$	-3.8	-2.4	-1.8	-6.0	-3.9	-3.1	-4.3	-2.8	-2.1	-3.8	-2.6	-2.1	-2.4	-1.7	-1.3
	MC$_{YUV}$	-1.3	-0.3	-0.1	-1.0	-0.1	-0.1	-0.6	-0.1	0.0	-0.8	-0.1	-0.1	-0.5	-0.1	0.0
	Avg.	**-2.5**	**-1.3**	**-1.0**	**-3.4**	**-1.9**	**-1.5**	**-2.3**	**-1.3**	**-1.0**	**-2.2**	**-1.3**	**-1.1**	**-1.4**	**-0.8**	**-0.6**

TABLE X
RUNTIME MEMORY ACCESS BANDWIDTH OF ILC AND ISC

		ILC									
		AI					**RA & LB**				
Memory Model		**4x2**	**4x4**	**8x2**	**8x4**	**cache**	**4x2**	**4x4**	**8x2**	**8x4**	**cache**
Lossy	**Min.**	2.1	1.7	2.1	2.3	0.8	0.0	0.0	0.0	0.0	-0.3
	Max.	9.9	9.1	10.2	11.8	5.2	1.5	1.1	1.6	1.5	0.1
	Avg.	4.7	5.3	5.9	7.2	2.3	0.3	0.2	0.3	0.3	0.0
Lossless	**Min.**	4.0	4.1	4.3	5.2	2.2	0.0	0.0	0.0	0.0	0.0
	Max.	16.3	13.4	17.3	17.7	7.9	6.2	5.1	6.4	6.9	2.0
	Avg.	8.4	7.5	8.7	10.0	4.1	0.9	0.7	0.9	1.0	0.2
		ISC									
Memory Model		**4x2**	**4x4**	**8x2**	**8x4**	**cache**	**4x2**	**4x4**	**8x2**	**8x4**	**cache**
Lossy	**Min.**	-2.4	0.1	-1.9	0.3	0.6	-0.1	-0.1	-0.1	-0.1	-0.1
	Max.	6.5	8.4	7.0	9.9	13.7	0.3	0.4	0.3	0.6	1.7
	Avg.	1.9	2.9	2.1	3.7	6.5	0.0	0.1	0.1	0.1	0.2
Lossless	**Min.**	-0.3	1.6	0.4	2.3	3.1	-0.1	0.0	-0.1	0.0	0.0
	Max.	38.3	36.2	40.4	49.7	40.6	5.6	6.4	5.6	8.5	10.9
	Avg.	8.9	10.3	9.4	13.9	15.4	0.7	0.8	0.7	1.1	1.2

Results shown are with full-frame search range.

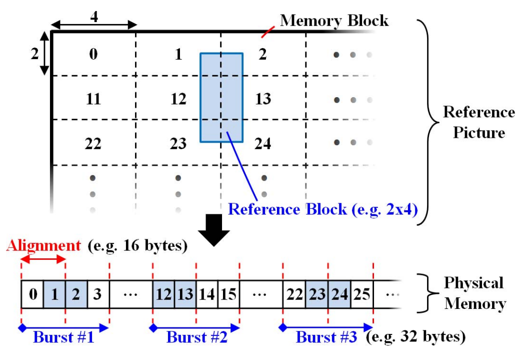

Fig. 12. Illustration of fetching a reference block from a physical memory.

fetched; that is, more data than needed have to be retrieved due to the many constraints imposed by the underlying memory configuration. It is also worth pointing out that the amount of overhead incurred depends on the location of the reference block. As such, interpretation of the results must take into account the dynamics of these factors.

Table X compares ILC and ISC in terms of the increase (measured in percentage) in the runtime memory access bandwidth at the decoder when they are integrated into the SCM-4.0 software as an additional coding mode. The results are provided for four different memory block sizes, 4×2, 4×4, 8×2, and 8×4. For the 8×4 case, additional results are provided by assuming the presence of a 48-KB four-way set associative cache, which is shared by IBC, ILC/ISC, and other inter modes that involve motion-compensated prediction. As shown, the incorporation of ILC or ISC leads to an increase in the memory access bandwidth in most test cases. The increase is more significant under the AI condition than under the RA and LB conditions, in which inter modes are observed to dominate the bandwidth consumption. Further inspection of TABLE X reveals that ILC incurs slightly higher bandwidth penalties than ISC in the lossy cases, whereas ISC consumes more than ILC in the lossless counterparts. This can be attributed to the more frequent use of shorter strings by ISC at higher rates. In general, the smaller the data access granularity, the higher the incurred overhead. The same argument also explains the relatively higher bandwidth consumption of both ILC and ISC in the lossless conditions, where the mode decision process tends to choose shorter lines or strings to give a more accurate prediction. In particular, ISC exhibits a larger degree of variability in the bandwidth consumption. This may arise from its flexibility of being able to choose from a wide range of string lengths. Sometimes extremely long strings can occur, which could lead to reductions in bandwidth (as evidenced by the negative values in Table X) since longer strings usually have lower access overhead and thus make it more efficient to access the memory. Another point to be noted is that a larger memory block size has a detrimental effect on the access of lines or strings. Essentially, more unnecessary data would have to be accessed together with the requested line

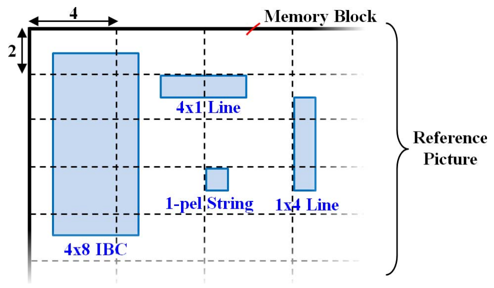

Fig. 13. Illustration of the worst case memory access for IBC, ISC, and ILC.

or string of pixels due to the mismatch between the line/string and block structures and the requirement to access the data in the basic unit of a memory block. The last observation is that when the cache memory is present, ISC incurs higher bandwidth penalties than ILC, particularly in the lossy cases.

2) Worst Case Analysis: Although the change of runtime memory access bandwidth helps characterize the impact of ILC and ISC on the decoder's complexity, the results can be misleading as they depend highly on the encoding algorithm. For example, a fine-tuning on how often they are enabled, a different restriction on how long a line/string can be, or a slight change in where lines/strings are copied from could easily alter the conclusion. For encoding-independent measurement, we provide results based on another complexity measure—per-pixel memory access bandwidth, termed P-number, which is computed by

$$P = \frac{\left\lceil \frac{m-1+M-1}{m} \right\rceil \times \left\lceil \frac{n-1+N-1}{n} \right\rceil \times m \times n}{M \times N} \quad (1)$$

where $M \times N$ and $m \times n$ represent the target block size and the memory block size, respectively [5].

This P-number metric, also introduced by the JCT-VC committee for their core experiments, reports the ratio of the data size associated with the memory blocks that need to be read from the memory in order to obtain the requested line/string/block to the data size of the line/string/block itself. That is, it serves as an indication of how much read overhead will be incurred for a given memory block pattern and size.[8] In particular, such ratio is computed for the worst case scenario, in which its value attains the maximum. Usually, the worst case occurs when the requested line/string/block is of its minimum size and located at some position where it spans across the maximum number of memory blocks. Fig. 13 illustrates the worst case scenarios for IBC, ISC, and ILC, respectively, where it is assumed that IBC has a minimum block size of 4×8, ISC a minimum string length of 1, and ILC a minimum line size of 4 (which can be a 4×1 horizontal or a 1×4 vertical line).

Table XI compares their P-numbers against that of Inter 8×8 bi-prediction—the worst case of all inter modes in terms of memory access characteristics—for 4×2, 4×4, 8×2, and 8×4 memory blocks, respectively. It is generally desirable to have a P-number lower than that of Inter 8×8 bi-prediction,

[8]Writing data to memory is usually less of a problem than reading them from memory as the former does not have the complication of possibly being random in data access pattern as is often the case with the latter.

TABLE XI
P-NUMBERS OF INTER PREDICTION, IBC, ISC, AND ILC

	Target Block Size $M \times N$	Memory Block Size $m \times n$			
		4×2	4×4	8×2	8×4
Inter	8×8	8	9.5	11	13
IBC	4×8	2.5	3	5	6
ISC	1×1	8	16	16	32
ILC (column)	1×4	6	8	12	16
ILC (row)	4×1	4	8	8	16

especially when the existing decoder designs are to be reused with minimum changes. Note that the P-number calculation for Inter 8×8 bi-prediction is slightly different from (1), mainly to accommodate the overhead from subpel prediction.[9] From Table XI, we see that IBC has consistently lower P-numbers than Inter 8×8 despite its smaller block size, i.e., 4×8. This is because IBC is essentially a uniprediction technique and it does not support subpel prediction. It is also observed that ISC has significantly larger P-numbers than the others due to the extremely small 1-pixel-long string in the worst case. By contrast, the P-numbers of ILC are somewhere in between those of IBC and ISC, with the 1×4 vertical line being worse than the 4×1 horizontal line. When compared with Inter 8×8 s, the P-number of ILC, vertical or horizontal, is either smaller or slightly larger depending on the memory block size. This, however, does not imply that the increase in memory access burden after incorporating ILC into the existing decoders would be moderate. After all, the above analysis does not include factors like memory alignment and burst, which would further complicate the identification of the worst case.

D. IBC With Restricted Search Range

So far, full-frame search for IBC has been taken for granted and has been implemented in SCM-4.0 as the baseline for comparison. There, however, has been much debate on whether full-frame IBC is really practical. Major concerns include the extra buffer for storing an unfiltered picture, the increase in memory access bandwidth for writing out this picture, the usually stringent low-delay requirements, and the limited computing power of the encoder. This section provides information on how another design alternative, which combines IBC and ILC with 4-CTU local search, compares against full-frame IBC and sees how ILC may compensate for the loss of IBC due to local search. It is generally believed that four CTUs may be small enough to be kept on-chip, so that the data access for IBC and ILC need not go off-chip.

For this experiment, we take SCM-4.0 with IBC disabled as the anchor. Table XII summarizes the rate savings over this stripped anchor when we separately enable full-frame IBC, 4-CTU IBC, and the combination of 4-CTU IBC and 4-CTU ILC. We see that full-frame IBC does bring considerable compression benefit, showing 20%–30% rate savings. However, it is also observed that IBC with 4-CTU local search already captures one-half or two-thirds of the gains. On top

[9]$P = \frac{\left\lceil \frac{m-1+M-1}{m} \right\rceil \times \left\lceil \frac{n-1+N-1}{n} \right\rceil \times m \times n}{M \times N}$, where L denotes the length of the interpolation filter.

TABLE XII
COMPRESSION PERFORMANCE COMPARISON OF FULL-FRAME IBC, 4-CTU IBC, AND 4-CTU IBC PLUS ILC

		IBC			IBC			IBC & ILC		
		AI	RA	LB	AI	RA	LB	AI	RA	LB
Search Range		Full-frame			4-CTU			4-CTU		
Lossy	$\mathbf{TGM}_{RGB}$	-32.3	-20.9	-12.8	-20.4	-12.8	-7.6	-25.9	-16.2	-9.2
	$\mathbf{MC}_{RGB}$	-28.7	-18.3	-8.4	-17.6	-11.1	-4.4	-20.9	-13.3	-5.4
	$\mathbf{TGM}_{YUV}$	-34.0	-21.7	-12.3	-21.9	-13.6	-7.4	-27.3	-16.9	-8.8
	$\mathbf{MC}_{YUV}$	-30.2	-20.1	-9.4	-19.6	-12.7	-5.2	-22.9	-14.9	-6.1
	Avg.	**-31.3**	**-20.3**	**-10.7**	**-19.9**	**-12.6**	**-6.1**	**-24.2**	**-15.3**	**-7.4**
Lossless	$\mathbf{TGM}_{RGB}$	-20.7	-14.3	-11.3	-12.3	-8.3	-6.8	-17.1	-11.5	-9.2
	$\mathbf{MC}_{RGB}$	-16.5	-3.7	-2.2	-7.6	-1.6	-1.0	-9.2	-1.9	-1.2
	$\mathbf{TGM}_{YUV}$	-21.1	-14.5	-11.1	-12.6	-8.4	-6.7	-17.6	-11.6	-9.0
	$\mathbf{MC}_{YUV}$	-18.5	-4.3	-2.5	-8.7	-1.9	-1.1	-10.5	-2.3	-1.3
	Avg.	**-19.2**	**-9.2**	**-6.8**	**-10.3**	**-5.1**	**-3.9**	**-13.6**	**-6.8**	**-5.2**

Results shown are measured relative to the stripped SCM-4.0 anchor.

of that, applying ILC achieves 3%–4% more savings, with the performance gap between full-frame and 4-CTU local search reduced to 3%–7% in lossy cases and 1%–6% in lossless cases. The gap would become even smaller in real applications, since it may not be possible to approach the full potential of full-frame IBC by adopting the costly hash-based search algorithm in SCM-4.0. Therefore, the combination of IBC and ILC with local search may be worth considering, particularly for low-delay and real-time applications.

VI. CONCLUSION

This paper proposed an ILC technique that adopts 1-pixel-thin lines as a basic matching unit for prediction. Essentially, it extends the notion of IBC by allowing a PU to be divided uniformly into lines. We first examined the partition types of IBC blocks in terms of their spatial coverage and showed that IBC tends to be enabled more with smaller partitions and nonsquare structures. This observation was then elaborated by empirical analyses, which reveal the superiority of lines over other nonsquare partitions, resulting in a line-based prediction. With such a uniform structure, lines can run independently and in parallel. Due to the increased amounts of search operation and line vectors, a fast line vector search algorithm and an adaptive line vector prediction scheme were proposed, which achieved a much reduced encoding time and a comparable compression performance to that of the optimal exhaustive search. Experimental results justify the effectiveness of the proposed prediction scheme. These results also reveal that ILC not only achieves comparable coding performance to ISC but also presents more compatible with IBC because it can reuse both the syntax design and decoding process of IBC for pixel copying. Thus, it would be more natural to combine ILC with IBC. When taking the compression performance and the memory access bandwidth tradeoffs into account, such a combination with local search may be considered more attractive than full-frame IBC, where full-frame search is generally considered less practical.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers and Associate Editor Y. Wang for their invaluable comments that helped us to improve the quality of this paper, and the Ministry of Science and NOVATEK, Taiwan, for their support of funding the work.

REFERENCES

[1] "Press release of the 108th meeting in Valencia, Spain," document MPEG-W14311, ISO/IEC JTC1/SC29/WG11, Apr. 2014.
[2] G. Bjontegaard, "Improvements of the BD-PSNR model," document VCEG-AI11, ITU-T SG16 Q6, Jul. 2008.
[3] T.-S. Chang *et al.*, "RCE3: Results of subtest D.2 on Nx2N/2NxN/NxN intra block copy," document JCTVC-P0180, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jan. 2014.
[4] T. M. Cover and J. A. Thomas, "Elements of Information Theory," 2nd ed. New York, NY, USA: Wiley, 2006.
[5] E. Francois, A. Tabatabai, and E. Alshina, "BoG report: Methodoly for evaluating complexity of combined and residual prediction methods in SHVC," document JCTVC-L0440, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jan. 2013.
[6] R. Joshi, S. Liu, J. Xu, and Y. Ye, "HEVC screen content coding draft text 3," document JCTVC-T1005, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Feb. 2015.
[7] D.-K. Kwon and M. Budagavi, "Fast intra block copy (IntraBC) search for HEVC screen content coding," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2014, pp. 9–12.
[8] G. Laroche, T. Poirier, C. Gisquet, and P. Onno, "Non-CE2: IBC encoder improvements for SCM2.0," document JCTVC-S0065, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Oct. 2014.
[9] B. Li and J. Xu, "On WPP with palette mode and intra BC mode," document JCTVC-S0088, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Oct. 2014.
[10] B. Li and J. Xu, "SCCE4: Results of test 3.1," document JCTVC-R0098, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2014.
[11] B. Li, J. Xu, and F. Wu, "1-D dictionary mode for screen content coding," in *Proc. IEEE Int. Conf. Vis. Commun. Image Process.*, Dec. 2014, pp. 189–192.
[12] B. Li, J. Xu, F. Wu, X. Guo, and G. J. Sullivan, "Description of screen content ccoding technology proposal by Microsoft," document JCTVC-Q0035, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Apr. 2014.
[13] R.-L. Liao, C.-C. Chen, and W.-H. Peng, "On comparison of intra line copy and intra string copy for HEVC screen content coding," in *Proc. IEEE Int. Conf. Vis. Commun. Image Process.*, Dec. 2015.
[14] T. Lin, X. Chen, and S. Wang, "Pseudo-2D-matching based dual-coder architecture for screen contents coding," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops*, Jul. 2013, pp. 1–4.
[15] T. Lin, P. Zhang, S. Wang, K. Zhou, and X. Chen, "Mixed chroma sampling-rate High Efficiency Video Coding for full-chroma screen content," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 173–185, Jan. 2013.
[16] T. Lin, K. Zhou, L. Zhao, and X. Chen, "Non-CE1: Enhancement to palette coding by palette with pixel copy (PPC) coding," document JCTVC-U0116, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jun. 2015.
[17] C. Pang, J. Sole, L. Guo, M. Karczewicz, and R. Joshi, "Non-RCE3: Intra motion compensation with 2-D MVs," document JCTVC-N0256, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2013.
[18] C. Pang, J. Sole, and M. Karczewicz, "SCCE1: Test 1.1—Intra block copy with different search areas," document JCTVC-R0184, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2014.

[19] K. Rapaka, V. Seregin, C. Pang, and M. Karczewicz, "On parallel processing capability of intra block copy," document JCTVC-S0220, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Oct. 2014.

[20] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[21] J. Xu, R. Joshi, and R. A. Cohen, "Overview of the emerging HEVC screen content coding extension," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 50–62, Jan. 2016.

[22] H. Yu, R. Cohen, K. Rapaka, and J. Xu, "Common test conditions for screen content coding," document JCTVC-T1015, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Feb. 2015.

[23] H. Yu, X. Wang, and J. Ye, "AHG8: More investigation on screen content coding," document JCTVC-M0320, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Apr. 2013.

[24] L. Zhao, K. Zhou, S. Wang, and T. Lin, "Non-CE3: Improvement on intra string copy," document JCTVC-T0139, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Feb. 2015.

[25] F. Zou, Y. Chen, M. Karczewicz, and V. Seregin, "Hash based intra string copy for HEVC based screen content coding," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops*, Jun./Jul. 2015, pp. 1–4.

Chun-Chi Chen received the B.S. degree in computer science from National Central University, Taoyuan, Taiwan, in 2007, and the M.S. degree in multimedia engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2009, where he is currently working toward the Ph.D. degree with the Institute of Computer Science and Engineering.

He has actively participated in the ISO/IEC MPEG and ITU-T VCEG standardization organization since 2011. He has been an active contributor to the High Efficiency Video Coding standard and its Screen Content Coding extension. In these standardization bodies, he has coordinated several core experiments on intra line copy and string copy techniques. He is an Intern with InterDigital Communications LLC, San Diego, CA, USA. He has authored or co-authored over 50 technical papers and video standard contributions. His research interests include high-efficiency image/video compression and screen content coding.

Wen-Hsiao Peng (SM'13) received the B.S., M.S., and Ph.D. degrees from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1997, 1999, and 2005, respectively, all in electronics engineering.

He was with the Intel Microprocessor Research Laboratory, Santa Clara, CA, USA, from 2000 to 2001, where he was involved in the International Organization for Standardization (ISO) Moving Picture Expert Group (MPEG)-4 fine granularity scalability and demonstrated its application in 3-D peer-to-peer video conferencing. Since 2003, he has actively participated in the ISO MPEG digital video coding standardization process and contributed to the development of the High Efficiency Video Coding (HEVC) standard and MPEG-4 Part 10 Advanced Video Coding Amd.3 Scalable Video Coding standard. His research group at NCTU is one of the few university teams around the world that participated in the Call-for-Proposals on HEVC and its Screen Content Coding extension. He is currently an Associate Professor with the Computer Science Department, NCTU. He is a Visiting Scholar with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA. He has authored over 60 technical papers in the field of video/image processing and communications and over 50 standard contributions. His research interests include HEVC, screen content coding, visual search and information retrieval, and machine learning.

Dr. Peng is a Technical Committee Member of the *Visual Signal Processing and Communications* and *Multimedia Systems and Application* tracks of the IEEE Circuits and Systems Society. He organized several special sessions on HEVC and related topics in prestigious conferences and was a Technical Program Co-Chair for the Conference on Visual Communications and Image Processing in 2011. More recently, he served as a Guest Editor for a Special Issue on Screen Content Video Coding and Applications in IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS.