Data-Parallel Intra Decoding for Block-Based Image and Video Coding on Massively-Parallel Architectures

Bart Pieters, Charles-Frederik Hollemeersch, Jan De Cock, Peter Lambert, and Rik Van de Walle

Multimedia Lab, Department of Electronics and Information Systems (ELIS) Ghent University — IBBT, Gaston Crommenlaan 8 bus 201, 9050 Ledeberg Ghent, Belgium

Abstract

With the increasing number of processor cores available in modern computing architectures, task or data parallelism is required to maximally exploit the available hardware and achieve optimal processing speed. Current state-of-the-art data-parallel processing methods for decoding image and video bitstreams are limited in parallelism by dependencies introduced by the coding tools and the number of synchronization points introduced by these dependencies, only allowing task or coarse-grain data parallelism. In particular, entropy decoding and data prediction are bottleneck coding tools for parallel image and video decoding. We propose a new data-parallel processing scheme for block-based intra sample and coefficient prediction that allows fine-grain parallelism and is suitable for integration in current and future state-of-the-art image and video codecs. Our prediction scheme enables maximum concurrency, independent of slice or tile configuration, while minimizing synchronization points. This paper describes our data-parallel processing scheme for one- and two-dimensional prediction and investigates its application to block-based image and video codecs using JPEG XR and H.264/AVC Intra as a starting point. We show how our scheme enables faster decoding than the state-of-the-art wavefront method with speedup factors of up to 21.5 and 7.9 for JPEG XR and H.264/AVC intra coding tools respectively. Using the H.264/AVC intra coding tool, we discuss the requirements of the algorithm and the impact on decoded image quality when these requirements are not met, as well as the Finally, we discuss the impact on coding rate in order to allow for optimal parallel intra decoding.

Keywords:
GPU, H.264/AVC, intra prediction, JPEG XR, massively parallel, parallel processing

1. Introduction

Popular image and video compression systems such as JPEG XR and H.264/AVC use block-based intra prediction schemes. Each image and in case of a video sequence, each video picture is divided into blocks typically 4×4, 8×8, or 16×16 samples in size. To achieve a high compression ratio, these blocks are predicted using intra prediction methods either in the transform (JPEG XR) or sample domain (H.264/AVC) using surrounding coefficients and samples respectively. Because of this, a high number of dependencies are introduced in the decoding phase of these pictures as typically the standards agree on predicting blocks serially, in a raster-scan order, using previously-decoded coefficients or samples. Evidently, this processing order and these dependencies conflict with fine-grain parallel processing. Indeed, besides the entropy decoding, intra prediction is the main coding tool not suited for parallel processing[1, 2] in a block-based codec, and is therefore the parallel processing bottleneck according to Amdahl's law[3]. To enable limited data-parallelism, typically the concept of independently-coded block groups (often referred to as slices or tiles) is introduced. Yet, these do not allow fine-grain parallelism.

Massively-parallel architectures accommodate a high number of parallel processors, typically in the order of thousands. As some hardware achieves this level of parallelism by using an underlying SIMD architecture, support for task-level parallelism is limited and data-level parallelism is preferable. Furthermore, synchronization of all processors in a massively-parallel architecture is costly, as a significant overhead is introduced as thousands of processors are waiting for the last thread to end. Finally, as throughput is the main target of these systems, a single processor is typically slow and introduces a large latency in program outputs, which hinders even simple serial calculations[4, 5]. An example of such a massively-parallel architecture is the Graphics Processing Unit available in commodity computer systems these days. These GPUs provide between 480 (NVIDIA) and 3200 (ATI) Streaming Processors and use a mix of MIMD and SIMD hardware to achieve such high parallelism. With these GPUs, synchronization between processors is

costly as shown by Feng *et al.*[5] and fine-grain parallelism is especially required as the sequential speed of these processors is typically low (see Pieters *et al.*[4]).

This paper proposes a new data-parallel algorithm for parallel intra prediction decoding for block-based image and video coding standards, suited for massively-parallel architectures. Our algorithm allows for one- and two-dimensional prediction without significant changes to the prediction algorithm, thereby still retaining optimal serial processing capabilities for single-core systems and limiting the impact on rate-distortion behavior. Most importantly, our new prediction scheme minimizes synchronization points and allows for maximum parallelism, contrary to the current state-of-the-art, without limiting prediction dependencies. The proposed parallel prediction scheme is applied to the JPEG XR image and H.264/AVC video coding standard. The level of parallelism achieved by our algorithm is invariant to block group (slice or tile) configuration. The impact on decoding quality and the trade off between fast parallel processing and bit rate is investigated for the H.264/AVC standard. Finally, the performance of the algorithm in terms of speed is compared to that of the current state-of-the-art wavefront algorithm for both JPEG XR and H.264/AVC.

This paper is organized as follows. Section 2 describes the data-dependencies introduced by intra prediction and their impact on coding rate, and discusses previous work. Section 3 proposes our new data-parallel processing algorithm for one-dimensional and two-dimensional intra prediction. Section 4 discusses the application of our parallel method to the JPEG XR and H.264/AVC standards, as well as the implementation and mapping of the algorithm on the GPU. In Section 5, performance results are presented. Finally, Section 6 concludes the paper.

2. Intra Prediction Dependencies Overview and Related Work

Figure 1 shows intra prediction schemes typically used in image and video coding standards. To remove redundancies from the bitstream, samples or coefficients are predicted using previously-decoded neighboring samples or coefficients to the left, top, top-left, and top-right. The prediction mode is defined in the coding standard and is described or derived from the bitstream in the decoding process. For example, the JPEG XR image coding standard uses a prediction mode with only one dependency as shown in Figure 1(a),

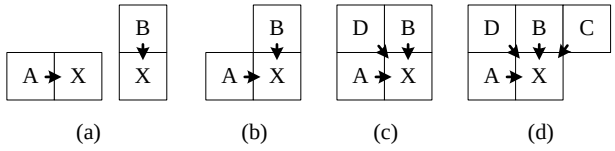


Figure 1: Sample or coefficient prediction for the current block X can use one (a), two (b), three (c), or four (d) previously-decoded input sample or coefficient blocks.

while the H.264/AVC video coding standard uses up to four input blocks, as illustrated in Figure 1(d).

As each sample becomes dependent on the previously-decoded blocks, a dependency chain is introduced, impeding efficient parallel processing. To limit dependencies, block-based image and video coding standards introduced the concept of block groups. Block groups containing a number of coded blocks can be decoded independently thus eliminating dependencies over block groups boundaries and increasing parallel decoding potential. In H.264/AVC block groups are called slices. For JPEG XR, these are called tiles. The literature proposes a combination of introducing coding block groups[6, 7, 8, 9] and altering the block processing orders to allow limited parallel processing[10, 11, 12, 13]. As mentioned before, task level parallelism [9, 10, 14, 15] does not allow sufficient fine-grain parallelism to enable optimal processing on massively-parallel architectures. To enable fine-grain parallelism for these architectures, a high number of block groups need to be introduced, decreasing compression efficiency as our measurements in Figure 2 show. For example, to enable 240 concurrent processing jobs in a 1080p video picture, one block group will hold 32 macroblocks. The figure shows how the compression efficiency decreases by 14.7% for H.264/AVC and by 23.5% for JPEG XR, compared to using only one block group for the entire picture. With the increasing number of parallel processing units of current and future processors, parallel processing is required to be independent of block group configuration. It is clear that there is a need for a parallel processing algorithm that is scalable for future hardware while minimizing the impact on coding efficiency.

Another method to introduce parallel processing for data prediction is the use of a processing order which has equivalent effects as the mandatory raster-scan order, but introduces parallelization opportunities. At the time of writing, state-of-the-art algorithms use a wavefront method[11, 12, 14, 13] for

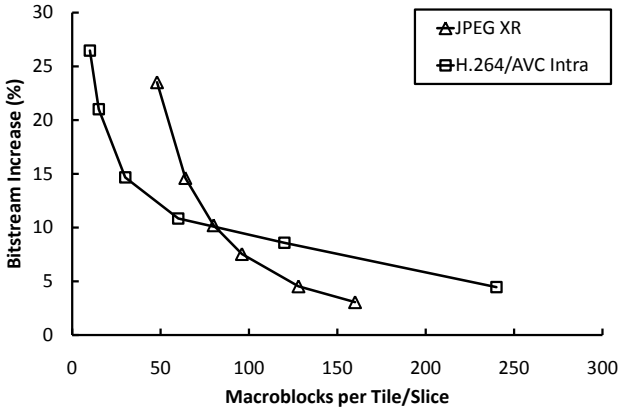


Figure 2: Bit rate increase when encoding using multiple tiles or slices with JPEG XR and H.264/AVC respectively for 1080p.

parallel processing as depicted in Figure 3(a-c). Such an algorithm divides the image in a number of waves, processed serially. Each block in a wave can be predicted in parallel as all required information was decoded in the previous wave. Three wavefront configurations are typically used[13]. First, waves travel from left to right (or top to bottom) (Figure 3(a)), which corresponds to a prediction scheme as depicted in Figure 1(a). Second, waves follow a path of 45 degrees from the top-left to the bottom-right (Figure 3(b)), corresponding to the prediction scheme illustrated in Figure 1(b-c). Third, waves travel following a path of 33 degrees from top-left to bottom-right, corresponding to the prediction scheme illustrated in Figure 1(d). Note that a wavefront algorithm is useless for one-dimensional prediction where the first element of each line is predicted by the last element of the previous line (e.g., Quantization Parameter prediction in H.264/AVC). Indeed, in this case, there would be as many waves as elements, making the wavefront method equivalent to serial processing.

Even though the wavefront introduces parallelism in each wave, parallel processing is still limited and the method introduces a large number of synchronization points, as processing of a wave can only start after the previous wave was finished. For example, for a video picture predicted using the H.264/AVC standard, a wave can contain a maximum number of blocks determined by its height. In case of a resolution of 1080p, only 270 blocks can be processed in parallel for a limited number of waves. For a massively-parallel architecture, this number of parallel processing tasks is too small as

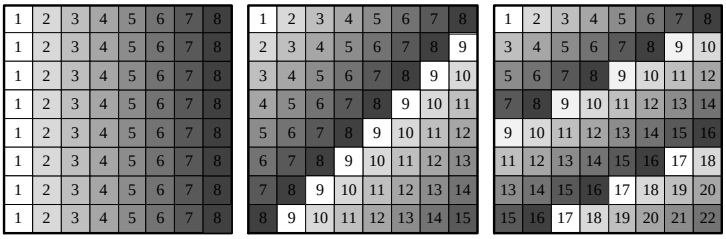


Figure 3: Prediction of samples/coefficients in a slice/tile using west (a), north-west (b) or a north-north-west wavefront (c).

at the time of writing, current state-of-the-art GPUs use up to 480 processing cores and are designed to process thousands of jobs in parallel. The high number of synchronization points introduced is dependent on the prediction mode. For example, for a wavefront using a 33 degree angle such as required by the prediction scheme used in H.264/AVC, the number of synchronization points is defined by the following equation where w and h are the width and height of the picture in blocks respectively:

$$SyncPointCount = w + 2h - 2. \tag{1}$$

For a 1080p H.264/AVC-coded video picture, this results in 1,018 waves or synchronization points. These synchronizations can compromise processing speed on certain massively-parallel processing architectures, such as the GPU. Feng *et al.*[5] shows how even low-computationally-complex serial operations can slow a modern GPU to a crawl. It is clear that there is a need for a parallel processing algorithm that maximizes parallel processing while minimizing synchronization points.

3. Proposed One- and Two-Dimensional Parallel Intra Prediction Algorithm

In the next two sub sections, we propose our novel data-parallel prediction scheme. For this prediction scheme, the following requirements were set:

- The algorithm should allow a data-parallel prediction which maximizes parallelism, independent of block cluster configuration. That is, the number of parallel threads working on the prediction should be proportional to the number of samples or coefficients in the picture.

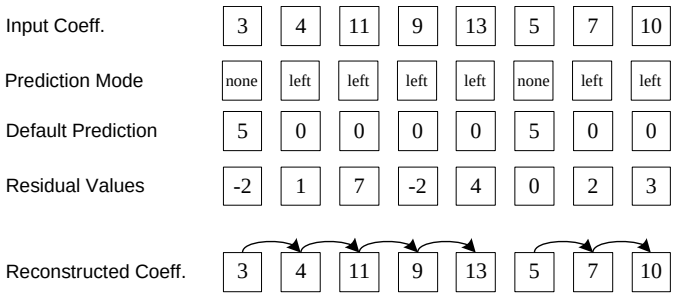


Figure 4: Example of eight samples being predicted from previously-encountered samples. Note how the sixth sample is not predicted from the left, but is instead initialized at zero.

- The algorithm should require a low number of synchronization points to ensure fast execution on massively-parallel architectures.
- Alteration to the bitstream in order to allow fast parallel prediction should be limited such that fast serial prediction (e.g., for single core solutions) is still possible while the compression rate is not compromised.
- The algorithm should mimic existing block-based prediction behavior as closely as possible as these coding tools have proven their efficiency and this will help fast adaptation.

With these goals in mind, our data-parallel prediction scheme for one-dimensional prediction is introduced next.

3.1. Proposed One-Dimensional Parallel Intra Prediction Algorithm

We start by showing an example of a row of samples predicted in Figure 4. One-dimensional prediction predicts each sample or coefficient in a row or column using previously-decoded samples or coefficients. For the remainder of the paper, we will refer to predicted values as coefficients while either meaning transformation coefficients or samples. In the figure, eight coefficients are predicted from left to right. From the figure it becomes clear that each reconstructed coefficient is the sum of the previous coefficient and the residual value. For example, value 13 is predicted as 9 plus 4. Except for the sixth element, which is predicted using the default prediction value, in

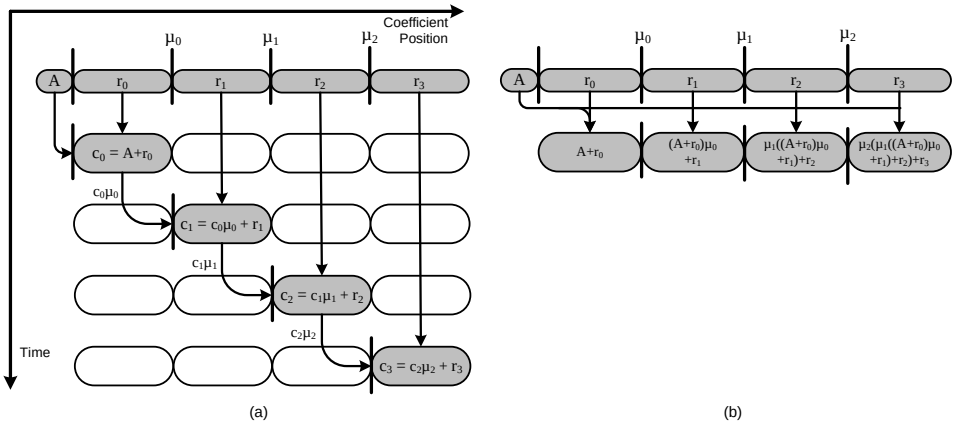


Figure 5: Prediction of coefficients in a one-dimensional array. (a): serial prediction, (b): parallel prediction using recursive doubling. Here, μ_i denotes the prediction factor (e.g. 1.0 for left prediction), r_i the residual coefficient, c_i the reconstructed coefficients at position i, and value A, the default prediction. Note that in this figure, only one default prediction A is considered, the others are considered to be zero.

this case five, just as the first value was predicted using this default prediction value. This type of prediction is illustrated in Figure 5(a) and is defined by the following equation for a n-dimensional array:

$$c_0 = A_0 + r_0, \tag{2}$$

$$c_i = A_i + \mu_{i-1} c_{i-1} + r_i, i < n. \tag{3}$$

Here, r_i is the residual value on position i and μ_i defines the proportion of the previous coefficient used for prediction. This factor μ_i typically lies in the interval [0-1] and is either a fixed value or is signaled in or derived from the bitstream. For the well-known delta-coding for example, μ_i is always 1. Prediction starts using the default prediction A, which typically holds a value of 2^{b-1} at the start position, with b the bit depth of the samples, e.g. a value of 128 for 8-bit components. At further positions, A is typically zero.

Now we can interpret this type of prediction as a transformation on the one-dimensional row of residual data where every coefficient is a weighted sum of all residual values, offset by a value A. For each coefficient, typically only previously-decoded residual values from the current row or column are used in the transformation. Returning to the example given in Figure 4,

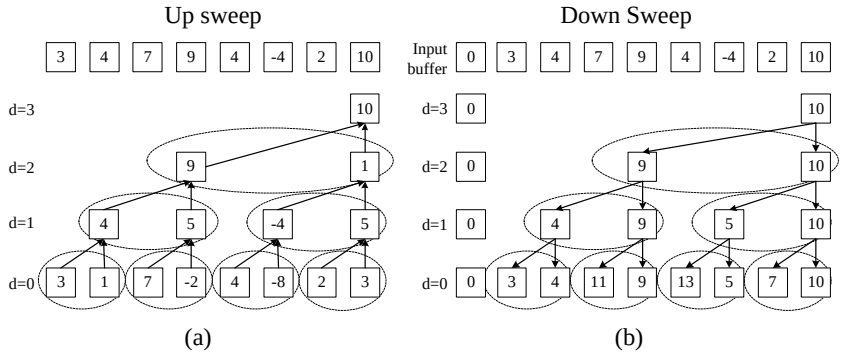


Figure 6: Parallel-prefix-sum scan algorithm using two sweeps: up sweep (a), down sweep (b).

we see in this figure how each coefficient can be calculated as the sum of all residual values in the array until a prediction mode is encountered that breaks the dependency on the left. For example, the third coefficient, valued 11, is the sum of residual values -2, 1, and 7, and default prediction 5. The seventh sample is the sum of 5, 0 and 2. We can write Equation (2)-(3) in a non-recursive form by using recursive doubling as follows:

$$c_i = r_i + \sum_{k=0}^{k<i} v_{k,i}(r_k + A_k), \tag{4}$$

$$v_{k,i} = \prod_{j=k}^{j<i} \mu_j. \tag{5}$$

This is illustrated in Figure 5(b). Intuitively, this shows how every reconstructed coefficient is a linear combination of all previously-encountered residual values, specified by the factor $v_{k,i}$, which defines the proportion of each residual value in the prediction of coefficient c_i. For example, in Figure 5 (a), the contribution of c_0 in c_2 is defined by the factor $\mu_0\mu_1$, while the contribution of c_1 in c_2 is defined by μ_1. Note that the values A_k can be incorporated into the residual data. Typically, A will be zero for all residual values except the first. This generalization will be used throughout the paper.

A trivial parallelization would be to calculate each c_i in parallel by evaluating Equation (4) as illustrated in Figure 5(b). It is clear however, that this introduces a computational complexity of $O(n^3)$ for each element, compared to $O(n)$ for sequential processing, and that parallelism in calculating each

element is limited. We will now show how this complexity can be reduced and parallelism can be increased. Returning to the example given in Figure 4 it becomes clear that one could calculate the fourth coefficient valued nine in two steps, by first summing up both coefficients one and two, and coefficients three and four, resulting in intermediate values seven and five. Next, the fourth coefficient can be calculated by summing these two intermediate values. This technique, proposed by Blelloch [16] and illustrated in Figure 6, is called a parallel prefix sum and allows us to calculate a summation as shown in Equation (4) in parallel. The parallel prefix sum or scan operation takes the binary associative operator $\oplus$ with identity I and an array of n elements and calculates the following:

$$[I, r_0, (r_0 \oplus r_1), .., (r_0 \oplus r_1 \oplus .. \oplus r_{n-2})]$$

By using a sum as operator $\oplus$, the scan algorithm can calculate for each element in a row the sum of its predecessors. Figure 6 shows how the scan algorithm works in two phases or sweeps for the example previously given. In the up sweep or reduce phase, partial sums are calculated and propagated over a number of steps we call passes to the top of the sum tree. For every pass, a number of summations can be done in parallel, namely $2^{-d-1}n$, where d is the depth in the tree. The number of passes and therefore, synchronization points in the up sweep is $log_2 n$. All operations occur in place on the input array. After all passes, each tree node contains the sum of its children leafs with the root of the tree containing the sum of all items of the array. Formally, the up sweep is defined as follows:

$$\begin{aligned} i &= n - j2^d - 1, 0 \leq j < n2^{-d}, \\ c_i^0 &= r_i, \\ c_i^d &= c_{i-2^{d-1}}^{d-1} + c_i^{d-1}, d > 0, \end{aligned}$$

with c_i^d the coefficient with index i on depth d. Next, the down sweep distributes all propagated local sums back into the list in $log_2 n$ passes. The tree is traversed from root to leaf. Here, we introduce a slightly-altered version of the original algorithm proposed by Blelloch:

$$\begin{aligned} i &= n - j2^{d+1} - 1, 0 \leq j < n2^{-d-1}, \\ c_i^d &= c_i^{d+1}, \\ c_{i-2^d}^d &= c_{i-2^d}^{d+1} + c_{i-2^{d+1}}^{d+1}. \end{aligned}$$

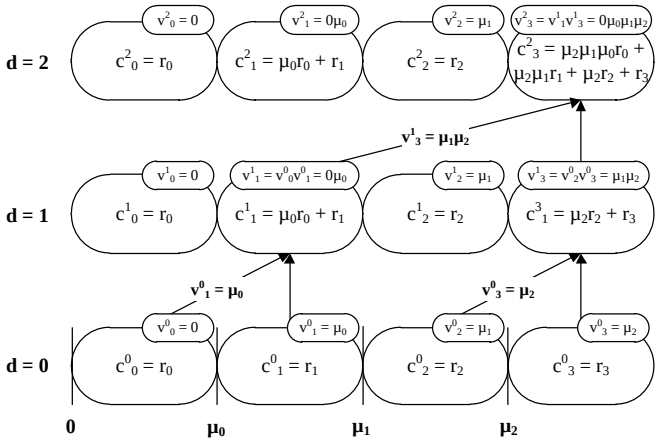


Figure 7: Up sweep of the proposed algorithm for one-dimensional prediction. Here, r_i denotes the residual coefficient at index i, c_i^d the temporary sum at position i and depth d, and μ_i the prediction factor at position i.

For each parent node, the sum stored in this parent node is copied to the right child node and the left node gets the sum of itself and its left sibling. Again, in each pass, $2^{-d-1}n$ parallel threads can be active to execute the sums. As with the serial algorithm, the scan algorithm has a complexity of $O(n)$.

Blelloch also shows how recurrence equations can be parallelized using the scan algorithm, where the factor v is a constant. However, in Equation (4), v is a variable factor based on all preceding μ factors. It appears the complexity of Equation (4) is $O(n^2)$ as for every factor v, a product of a maximum of n factors μ need to be calculated. In the next paragraphs, we will show how to use this scan algorithm for the prediction with a variable factor v, and how the complexity can be reduced to $O(n)$.

As a first step, the up-sweep of our proposed one-dimensional parallel prediction algorithm works according to the following equations and is illus-

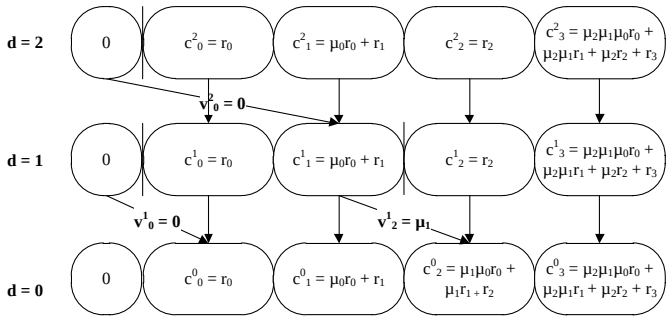


Figure 8: Down sweep of the proposed algorithm for one-dimensional prediction. Here, r_i denotes the residual coefficient at index i, c_i^d the temporary sum at position i and depth d, and μ_i the prediction factor at position i.

trated by Figure 7:

$$
\begin{aligned}
i &= n - j2^d - 1, j < n2^{-d}, \\
v_i^0 &= \mu_{i-1}, && (6)\\
v_i^{d+1} &= v_i^d v_{i-2^d}^d, d >= 0, && (7)\\
c_i^0 &= r_i, \\
c_i^{d+1} &= c_i^d + v_i^d c_{i-2^d}^d, d > 0. && (8)
\end{aligned}
$$

In the figure, the algorithm starts with the leafs of the tree by taking residual data r_i and r_{i+1} of two neighboring coefficients and calculates the temporary local sums c_1^1 and c_3^1 by using Equation (8). For every temporary local sum, the v factor is calculated using v factors from the depth below by simply multiplying the left and right branch v factors. In fact, Equation (7) calculates the amount of energy transferred from the left branch to the right-most node of the right branch. Notice how at the last pass, all local sums are complete and the last element contains the sum of the entire row. These sums are always written in place in the output buffer. The same goes for the factors v of each pass. Hence, extra memory required for the algorithm is limited. Indeed, for a buffer of one megabyte holding 32-bit coefficients, two megabyte of additional memory is required to store the 32-bit floating point factors and 32-bit coefficient values.

The down sweep is less complex than the up sweep and works in a similar manner as the algorithm by Blelloch previously explained. Here, sums are distributed down from the parent node to the right child node. The left

child node receives the local sum of its left sibling. As the local sum of the left sibling traverses the edge between itself and its sibling, the factor v of that node is taken into account as Equation (10) shows. Indeed, this factor defines how much the left sum is used in predicting its successive coefficient. The entire tree is eventually calculated as follows:

$$\begin{aligned} i &= n - j2^d - 1, j < n2^{-d}, \\ c_{-1}^d &= 0, \\ c_i^{log_2(n)} &= x_i, \\ c_i^d &= c_i^{d+1}, d > 0, \\ c_{i-2^d}^d &= c_i^{d-1} + v_{i-2^d}^{d+1} c_{i-2^{d+1}}^{d+1}. \end{aligned} \tag{9, 10}$$

The result of the down sweep is the predicted and reconstructed value for each coefficient. This way, prediction occurs in $O(n)$ passes as with serial processing. The extra calculations introduced are additional multiplications of the μ factors. As with the all-prefix-sum scan, parallelism with our proposed algorithm is maximized and synchronization points minimized.

As intermediate sums are generated over $log_2 n$ passes, storage with sufficient precision is required to hold these temporary coefficients. With a normal sum-scan, the required precision for the nodes in the tree is defined by:

$$P_c = log_2(n) + P_r, \tag{11}$$

where P_r is the number of bits required for storing one residual value and P_c the number of bits required for a coefficient in the tree. The precision of the residual values is defined by $2P_r$ where P_r is the precision of the sample or coefficient values. For 8-bit sample prediction in a 1080p image, Equation (11) calculates to 19 bits. Hence, the use of 32-bit values to hold coefficients is recommended. When the μ factor holds a fractional value, sufficient precision is required to store these fractional values, namely:

$$P_c = 2log_2(n) + P_r, \tag{12}$$

which calculates to 30 bits for a 1080p image using 8-bit sample prediction. The variance of the residual values however is typically small and values are distributed around zero as a good prediction, i.e., minimizing r_i, is the aim of the encoder. Therefore, the required precision will approach Equation (11) rather than Equation (12).

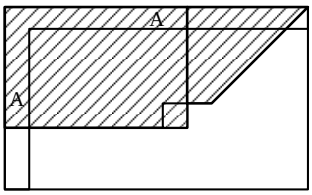


Figure 9: Visualization of possible non-zero parts of the matrix $\mathbf{V}_p$ for a coefficient at position p(x,y). Here, A denotes the initialization value for edge predictions.

One important aspect of the parallel prediction scheme is that over successive passes, available information is not in a final state. For example, in the up sweep, temporal coefficient values are in fact sparsely distributed local sums in an indefinite state. As such, inferring can only be done when the algorithm has finished. This implies that prediction schemes cannot rely on decoded values for decisions about prediction modes for example, and thus these modes should be coded in the bitstream.

As a result of the up and down sweep, all coefficients in the array are predicted in $2log(n)$ passes in parallel as they would have using a serial algorithm. Application of our proposed parallel one-dimensional prediction scheme to the JPEG XR image coding standards is discussed in Section 4. Next, we extend our algorithm for two-dimensional prediction.

3.2. Proposed Two-Dimensional Parallel Intra Prediction Algorithm

We now propose an extension to the one-dimensional prediction method into two dimensions where the prediction uses a combination of left, upper, upper-left, and potentially, upper-right blocks, as Figure 1(b-d) illustrates. As the prediction for a one-dimensional array is a transformation of the residual data of that array (typically using only residual data from coefficients to the left), so is the prediction of a two-dimensional array a transformation of the 2D residual data. Here, the calculation of each sample uses all residual data above, to the left, and partially to the right of the current coefficient as stated in the following equation and illustrated in Figure 9:

$$c_{p_{(x,y)}} = r_{p(x,y)} + \sum_{i=0,j=0}^{(i<=x+y-j),j<=y,(i\neq x)\wedge(j\neq y)} \mathbf{V}_{p(x,y),i,j}(r_{p(i,j)} + A_{p(i,j)}). \tag{13}$$

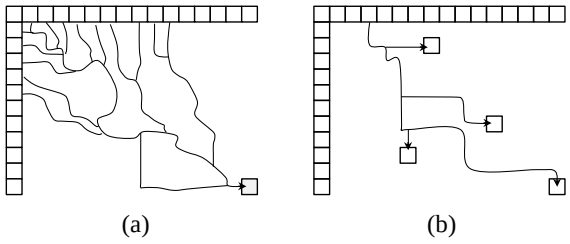


Figure 10: Path of prediction in a picture. (a): a coefficient is any linear combination of coefficients above and to the left of it, (b): a coefficient is predicted by exactly one or zero coefficients.

Here, $p(x, y)$ indexes elements at coordinates (x, y). Again, $r_{p(x,y)}$ denotes the residual data coefficient at position $p(x, y)$ and $A_{p(x,y)}$ the default prediction values. Throughout the remainder of the paper, A is considered to be incorporated into the residual values. The $\mathbf{V}_{p(x,y)}$ matrix for a coefficient $c_{p(x,y)}$ containing all v factors is built by using the $\mu_{p(i,j)}$ factors defined by the prediction mode of the coefficient. Each element $\mathbf{V}_{p(x,y),i,j}$ of $\mathbf{V}_{p(x,y)}$ determines the proportion of a residual coefficient $r_{p(i,j)}$ in the current coefficient $c_{p(x,y)}$. To preserve detail in intra prediction, typically only a limited number of surrounding blocks are used at once in Figure 1(b-d). As a result, $\mathbf{V}_{p(x,y)}$ will be sparse. Figure 10(a) illustrates how $\mathbf{V}_{p(x,y)}$ defines a prediction path when all non-zero values of the matrix are highlighted. The more inputs are used to predict a coefficient, the more the prediction path branches when traveling the path backwards. Figure 10 (b) shows what happens if prediction of a coefficient is limited to using only one input block, e.g., block A in Figure 1(b). Here, each prediction of a coefficient is defined by exactly one path. We will start by formulating a parallel algorithm for the situation where a coefficient is predicted by many previously-decoded coefficients (Figure 10(a)) and then apply this theory to the case where prediction is limited (Figure 10(b)). In the next paragraphs, we show how these prediction paths can be calculated in parallel analogously to the proposed one-dimensional scan algorithm in a limited number of passes.

As typically intra prediction is performed in raster scan, predicted values are carried to successive macroblocks. At the first pass of the algorithm, each output coefficient, c_p^1 on position $p(x, y)$ and depth 1, is a linear combination of the block's input coefficients, i_p^0 and μ parameters, according to the

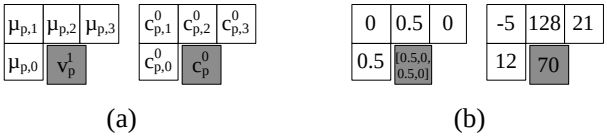


Figure 11: The prediction mode parameters μ and input coefficients for coefficient c_p^0 (a), and example μ parameters and coefficients.

following equations and illustrated by Figure 11(a):

$$
\begin{aligned}
n &= 2(d+1)+2, \\
c_p^0 &= r_p, && (14)\\
\vec{\imath}_p^{\,1} &= \begin{bmatrix} c_{p,0}^0 & \dots & c_{p,n-1}^0 \end{bmatrix}, d > 0, \\
\vec{v}_p^{\,1} &= \begin{bmatrix} \mu_{p,0} & \dots & \mu_{p,n-1} \end{bmatrix}, && (15)\\
c_p^1 &= \vec{\imath}_p^{\,1} \cdot \vec{v}_p^{\,1} + r_p. && (16)
\end{aligned}
$$

The vector $\vec{\imath}_p^{\,d}$ contains all input coefficients for the coefficient at position p and depth d, according to the specified prediction scheme and as illustrated in Figure 11(a) for the prediction scheme of the H.264/AVC standard in case of depth equal to one. Evidently, the prediction is corrected by the coefficient's residual data r_p. Here, the dependency parameter for each coefficient is a vector $\vec{v}$ consisting of the μ prediction mode parameters. Its dot product with the input coefficients $\vec{\imath}$ produces the temporary predictions for the current position at the current depth. Figure 11(b) illustrates an example where a coefficient is predicted using two input coefficients immediately to the left and to the top, as defined by the coefficient's dependency vector $\vec{v}_p^{\,0}$.

As with the 1D scan algorithm, an up and down sweep is required. In the up sweep, local 2D sums and the matrix $\mathbf{V}$ are calculated and propagated up into the tree. The down sweep distributes these sums and calculates the local correct predictions. As illustrated in Figure 12, in each pass, four local sum blocks are combined. In each pass in the up sweep, the local prediction is calculated according to Equation (16). Analogously, the dependency vector $\vec{v}$ needs to be calculated for each edge coefficient. Contrary to the one-dimensional algorithm, a coefficient can depend on more than one input coefficient. Therefore, each temporary predicted coefficient c_p^d has its own vector $\vec{v}_p^{\,d}$.

With the two-dimensional up sweep, four leafs are combined into one tree

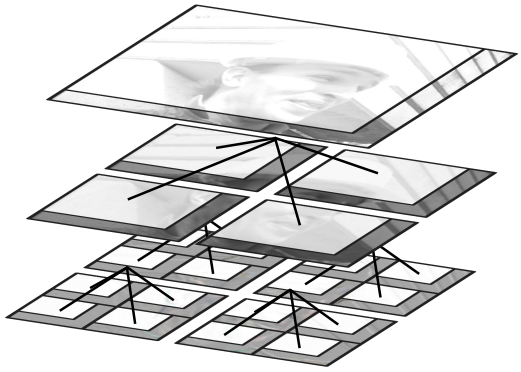


Figure 12: In each pass in the up sweep, edge coefficients are predicted using edge coefficients and dependency vectors of four quadrants of the pass below.

node, i.e., four blocks of the lower depth are combined into one coefficient block as illustrated by Figure 12. In each pass, temporary local coefficients are calculated for the tree node, as well as the vectors $\vec{v}$ for each edge coefficient defining the node's dependence on its input values. Figure 13 shows the four steps for combining four 1×1 blocks to one 2×2 block. Calculations start for the top-left quadrant (this is because of the typical raster-scan order used by most block-based image and video codecs). Starting with the first quadrant, the dependency vector $\vec{v}_p^d$ of each edge coefficient c_p^d needs to be remapped to a new vector $\vec{v}_p^{d+1}$, which defines the dependency of the coefficient on the inputs of the 2×2 block. Remapping this first quadrant is trivial as in fact, $\vec{v}_p^d$ is already a subset of $\vec{v}_p^{d+1}$ as illustrated in Figure 13. For the second quadrant, part of $\vec{v}_p^{d+1}$ is already defined by $\vec{v}_p^d$, this is:

$$\begin{aligned} \vec{\nu} &= \vec{v}_p^d, \\ \vec{\gamma}_p^d &= \begin{bmatrix} 0 & \dots & 0 & \nu_{n-n/2-1} & \dots & \nu_{n-1} \end{bmatrix}. \end{aligned} \tag{17}$$

The remaining factors ($v_{0,0}^1$ in Figure 11(b)) can now be expressed using the already remapped factors of the first quadrant, namely, $\vec{v}^2_{p(v_{0,0}^1)}$. Note that $p(v_{p,0}^1)$ here is used to denote the corresponding position of $v_{p,0}^1$, the first dependency factor for $c_{p,0}^1$ in the picture. Remapping this vector can be done

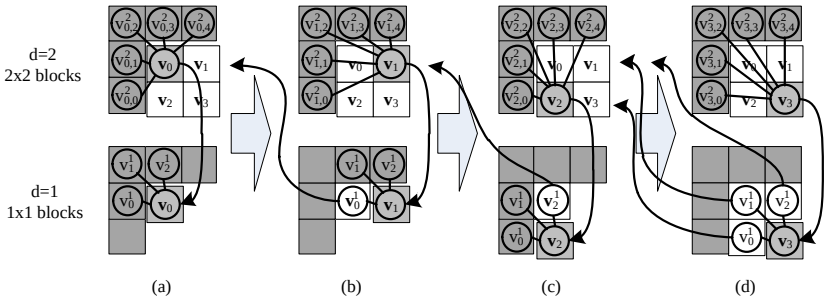


Figure 13: Combining four quadrants using the dependency vectors (v_p^d at depth d on position p) of the lower depth starting from the upper-left quadrant (a) to the lower-right quadrant (d).

using the following equations:

$$\vec{v}_p^{d+1} = \begin{bmatrix} v_{p(\nu_0),0}^{d+1} & v_{p(\nu_1),0}^{d+1} & \cdots & v_{p(\nu_k),0}^{d+1} \\ v_{p(\nu_0),1}^{d+1} & v_{p(\nu_1),1}^{d+1} & \cdots & v_{p(\nu_k),1}^{d+1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{p(\nu_0),j}^{d+1} & v_{p(\nu_1),j}^{d+1} & \cdots & v_{p(\nu_k),j}^{d+1} \end{bmatrix} \begin{bmatrix} \nu_0 \\ \nu_1 \\ \vdots \\ \nu_k \end{bmatrix} + \gamma_p^d, \tag{18}$$

$$k < n - n/2 - 1, j < n - 1. \tag{19}$$

Here, the coefficients that need remapping (ν_0 to ν_k, or v_0^1 in the case illustrated by Figure 13(b)) are multiplied with the input vectors corresponding with the coefficient at the position at which the dependency factor is applicable. At the same time, the local edge coefficient sum for each edge coefficient can be calculated using the dependency vector $\vec{v}_p^d$ for each edge coefficient as follows:

$$c_p^{d+1} = \begin{bmatrix} c_{p_{\nu_0}}^d & c_{p_{\nu_1}}^d & \cdots & c_{p_{\nu_{n-1}}}^d \end{bmatrix} \cdot \vec{v}_p^{d+1}. \tag{20}$$

The process of dependency parameter and local edge coefficient calculation is repeated for the third and fourth quadrant (Figure 11(c-d)), in a matter similar to the second quadrant. The following passes for the next depths function in the same manner, remapping edge coefficient dependency vectors and calculating local sums of edge coefficients.

All dependency vectors $\vec{v}$ for each pass need to be stored, as these are required in the next step to perform the down sweep. In the final pass, $\vec{v}$ does not need to be calculated as it defines the dependency on the prediction value zero. Once all these factors are available, the down sweep can start.

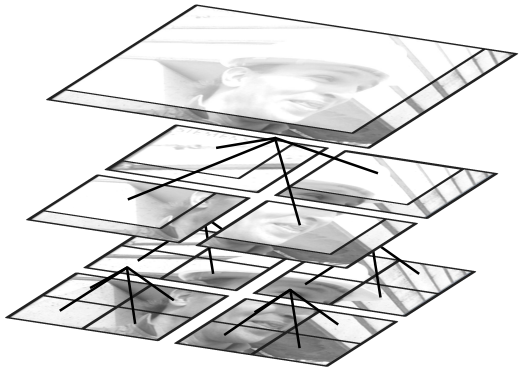


Figure 14: In each pass in the down sweep, the coefficient sums are distributed to four quadrants according to the dependency vectors of the edge coefficients of those quadrants. The final result is the predicted picture.

In the down sweep, localized sums are distributed over all areas of the video picture as illustrated by Figure 14. In the first pass in the figure, four quadrants are defined by their input coefficients and their dependency vector build in the up sweep on these coefficients. Using Equation (20), the inner coefficients for each quadrant can be calculated, effectively distributing the local total sums down to the lowest level.

Although at first sight the proposed method resembles techniques used in hierarchical image codecs such as wavelets with its distinguishable tree-based structure, our proposed technique significantly differs. Firstly, it is a prediction technique instead of a transformation. We remain in the same domain as the input coefficients, e.g., sample domain. Secondly, in the proposed two-dimensional method, each local sum contains the sum of all preceding residual data coefficients, i.e., the current predicted value, and not the sum of all sample values, i.e., the average of the image. As such, eliminating the final pass of the down sweep for example, does not introduce a down-sampled, average version of the image, but rather shows a coarsely-sampled version of the image where three quarters of the samples were disregarded.

3.3. Complexity Analysis

For each pass, the number of dimensions of $\vec{v}$ for each coefficient increases. However, as stated, the prediction path matrix is typically sparse. Indeed, in order to achieve a more detailed prediction, prediction modes preserving detail are preferred by the encoder. Furthermore, samples cannot be predicted

from samples to the right and bottom. Therefore, remapping of zero-valued components of $\vec{v}$ can be omitted. As a result, a typical implementation will use a list approach to store only the necessary components of $\vec{v}$. In fact, when the number of prediction modes is limited to using only one input block (see Figure 1(a)), the dimensionality of $\vec{v}$ is one, hence the number of dependency factors per coefficient does not grow over passes. Note that raster-scan order ensures us that a coefficient can only be dependent of residual data according to Equation (13) as illustrated in Figure 9. Therefore, for each coefficient, a significant well-defined number of components of its vector $\vec{v}$ will be zero.

Although remapping the vector $\vec{v}$ for each coefficient in a quadrant has a complexity of $O(n^3)$, in each pass, every vector $\vec{v}_p^d$ and coefficient c_p^d can be calculated in parallel. As a result, parallel complexity is $O(n)$. Because of the sparse nature of the vectors $\vec{v}$, the complexity is further reduced once a list approach is used. Finally, the locality of the algorithm allows the easy use of multiple processing units (e.g., four GPUs) to process each quadrant of the picture in parallel.

As only $2log_2$ passes are required, synchronization is limited. For example, predicting a 1080p picture requires 28 synchronization points compared to 1027 for the wavefront algorithm. Section 4.4 discusses the number of synchronization points required by the algorithm in more detail.

The precision required to hold coefficients c_p^d is defined by:

$$P_c = log_2(w) + log_2(h) + P, \tag{21}$$

where again P_c is the precision of the coefficient in the 2D tree and P the precision of the residual values. For a 1080p image, this calculates to 30 bits. Again, as remarked for the one-dimensional prediction scheme, the task of the encoder is to minimize residual values r_p in value. Hence, the actual required precision is far less than Equation (21) defines. Our measurements show a precision of 16-bit to be sufficient for for example, the H.264/AVC standard, addressed in the next section.

4. Application to image and video codecs

In this section we evaluate the proposed parallel intra algorithm by investigating how the parallel intra coding tool would fit in current and future state-of-the-art image and video codecs. The JPEG XR and H.264/AVC Intra coding tools were chosen to represent the one-and two-dimensional algorithm respectively.

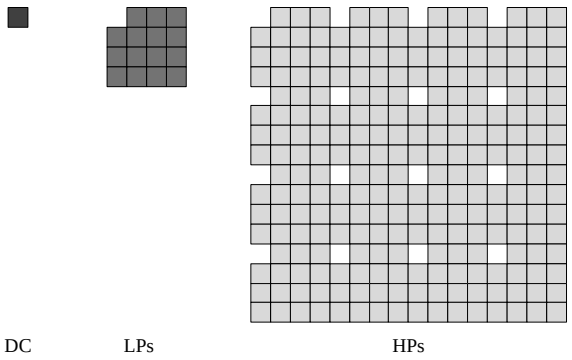


Figure 15: Frequency hierarchy of JPEG XR for one macroblock (left: luma and chroma 4:4:4, middle: chroma with 4:2:2, right: chroma with 4:2:0).

4.1. JPEG XR and the one-dimensional algorithm

JPEG XR (formerly known as Microsoft HD Photo) is a relatively new still-image compression standard, originally developed by Microsoft Corporation and standardized in June 2009. It specifically targets digital photography and features amongst others state-of-the-art compression capability, high dynamic range support, lossless coding support, and full-format 4:4:4 color coding. The standard allows for low-complexity encoding and decoding implementations.

With JPEG XR, pictures are divides in tiles which can be decoded independently. Within each tile, transformation coefficients are stored and predicted in a frequency hierarchy, a three-tiered prediction scheme where each tier's coefficients are predicted using previously-decoded coefficients in that tier. The frequency hierarchy is shown in Figure 15. Coefficients of each tier are predicted using their neighboring tiers and/or coefficients of the tier below. High-Pass (HP) coefficients are predicted using surrounding Low-Pass (LP) coefficients while these LP coefficients are predicted using previously-decoded neighboring LP coefficients and DC coefficients. DC coefficients finally are predicted using previously-decoded neighboring DC coefficients. Next, we will discuss how the proposed data-parallel prediction method fits into this hierarchical prediction scheme.

4.1.1. Mapping of the one-dimensional algorithm

Each DC coefficient is predicted using previously-decoded DC values above and to the left of the current DC according to the prediction scheme

illustrated in Figure 1(b). The prediction mode is not described in the bitstream, but instead is derived from the direction of the similarity of the previously-decoded DC values. As no information can be inferred from temporary local predictions in a scan algorithm such as the proposed two-dimensional algorithm, parallel prediction is not possible. Fortunately, only a limited amount of DC coefficients need to be predicted thanks to the frequency hierarchy. Prediction of LP coefficients uses previously-decoded LP and DC coefficients. Figure 16 shows in detail how LP coefficients are predicted. In the figure, every LP coefficient (light gray) next to the DC coefficient (dark gray) is predicted from its left neighbor coefficients left of the DC (LP_a). Alternatively, the coefficient can be not predicted at all. LP coefficients below the DC coefficient are predicted using LP coefficients directly above (LP_b). LP_a and LP_b calculation share no dependencies and can thereby be predicted in parallel. Additionally, each column or row of LP_a or LP_b respectively is not dependent on calculations of neighboring columns or rows, thereby allowing all rows and columns of LP coefficients to be calculated in parallel. Within each row or column, the coefficients can be predicted using the proposed one-dimensional parallel prediction method.

The proposed algorithm starts by assigning the μ factors according to the LP dependencies using Equation (6). Where the LP coefficient is predicted using the previous LP, this factor holds the value of 1, otherwise this factor holds the value of 0. At tile borders, the μ factor can be set to zero, or tiles can be scheduled for calculation in parallel by the application. In the up sweep using Equations (6)-(8), the algorithm calculates local sums of LP coefficients within one column or row. Next, these local sums are distributed down in the down sweep using Equations (9)-(10). The result is a column or row with coefficients predicted as they would using a serial algorithm.

Parallelization of High-Pass (HP) coefficients is trivial as no dependencies over 4×4 blocks are introduced. Each HP is predicted using its surrounding LP and DC coefficients. For more details on JPEG XR, the reader is referred to [17, 18].

4.1.2. Quality Assessment

As previously stated, LP rows and columns are predicted as as they would using a serial algorithm. Indeed, this is because the prediction tool in JPEG XR only uses linear operations, hence the proposed algorithm introduces bit-for-bit correct results.

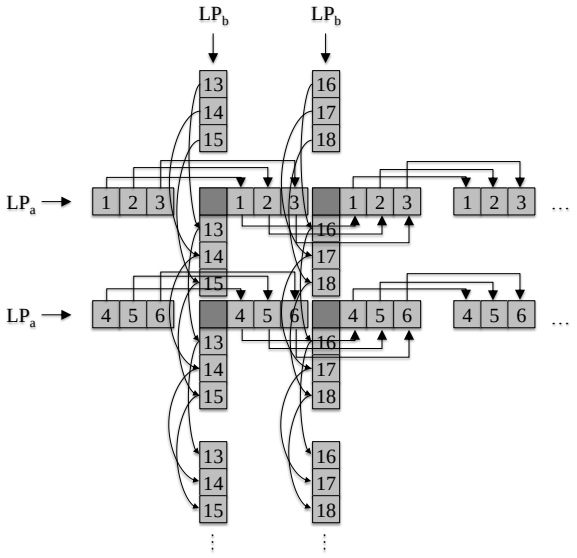


Figure 16: Prediction of LP coefficients in JPEG XR.

4.1.3. Implementation and Thread Mapping

By using the proposed one-dimensional data-parallel scheme for LP coefficient prediction combined with wavefront prediction for the DC coefficients, parallelization is maximized. Table 1 compares the number of synchronization points required for both the proposed algorithm and the wavefront algorithm. From the table it is clear that the number of synchronizations is significantly diminished with, for example, only 18 synchronization points required when processing a 1080p image. Hence we can state that the proposed algorithm successfully minimized the number of synchronization points. We implemented the algorithm using the NVIDIA CUDA Platform[19]. The thread mapping of the proposed one-dimensional scheme is straightforward. Here, we use an implementation similar to the parallel prefix sum scan algorithms available in the NVIDIA primitives library cudpp[20] where each CUDA thread is responsible for a single coefficient. For the first few passes in the up sweep, local block synchronization (using the *syncThreads*() instruction) is used as each CUDA block calculates the local sums over a small number of elements, e.g. 64. For the next passes, instead, kernel invocations are used as synchronization points. Because the proposed data-parallel algorithm minimizes synchronization points, the DC prediction becomes the main bottleneck. However, here, local block synchronizations can be used on

Table 1: Number of synchronization points required for the proposed one-dimensional method compared to the wavefront method.

Res.	Proposed one-dimensional			Wavefront
	Up Sweep	Down Sweep	Total	
CIF	7	7	14	88
480p	8	8	16	120
720p	9	9	18	320
1080p	9	9	18	480
2160p	10	10	20	960

massively-parallel systems like the GPU with their limited overhead. Therefore, we have successfully limited synchronization costs as will become clear from the results in the next section.

4.2. H.264/AVC Intra and the two-dimensional algorithm

In this sub section, we show how the proposed two-dimensional algorithm compares to the intra prediction tool in the H.264/AVC standard. Specifically, we show how non-linear operations in the H.264/AVC standard cause drift between encoder and parallel decoder output. Finally, we show in this sub section how an H.264/AVC encoder needs to be adjusted in order to synchronize results with the parallel decoding algorithm and what impact this has on coding efficiency using H.264/AVC Intra as a model.

4.2.1. Mapping of the two-dimensional algorithm

The H.264/AVC[21] video coding standard uses data prediction schemes that introducing calculation dependency chains in the coding tools frequently. Like JPEG XR, H.264/AVC offers coarse independent decoding by using block groups in a video picture, this time called slices. As previously mentioned, this introduces a loss in bit rate and provides parallelism insufficient for fine-grain parallel architectures. H.264/AVC Intra specifically uses data prediction for Quantization Parameter prediction and sample-based prediction for intra-coded video pictures. Firstly, delta prediction is used to signal the Quantization Parameter (QP) for each 4×4 block. For this, the proposed one-dimensional parallel prediction algorithm can be applied using a value of 1 for all μ factors but those corresponding to slice boundaries (see Equation (6)) . For these positions, the μ factor will hold zero. Next, the proposed one-dimensional algorithm will simply calculate the sum of all delta values in the up and down sweep resulting in a row of predicted QP values.

Secondly, the H.264/AVC Intra standard uses data prediction to predict samples in intra-coded video pictures. For this prediction tool we propose the use of the parallel two-dimensional algorithm. H.264/AVC Intra provides a number of intra prediction modes on both blocks of size 4×4 and 16×16. Note that only the right and bottom edges of a 4×4 or 16×16 block are used to predict neighboring blocks. As such, calculation of these edges suffice to later predict all inner coefficients of all blocks of the picture in parallel. For each prediction mode, μ, here a vector, defines the dependency of samples within a block on one or more samples in the neighboring blocks. This vector (using Equation (15)) will be used in the 2D up sweep to calculate the fractions of residual data of coefficients that are used to predict the current coefficient. Figure 17 shows some example mappings of prediction modes to μ vectors. For example, the DC prediction mode is represented by a eight-dimensional μ vector with all components holding value $\frac{1}{8}$. Hence, a single edge coefficient of a 4×4 block is dependent on each neighboring edge coefficient for $\frac{1}{8}$.

Blocks of size 16×16 can easily be handled as sixteen 4×4 blocks with μ vectors according to the 16×16 block prediction mode. The PCM prediction mode, where each sample is reconstructed without prediction at all can easily be handled by setting the μ vector for the coefficients of this block to a vector with a single zero component. Next the up sweep calculates the local sums and dependencies on neighboring coefficients according to Equations (14)-(20) as illustrated in Figure 12. Afterwards, the down sweep distributes the calculated local sums and resolves the dependencies of coefficient on neighboring blocks using the dependency vector v build in Equation (18). The final step involves predicting inner coefficients using edge coefficients of 4×4 blocks and rounding the final fractional reconstructed values to the nearest integer value.

Note that while similar, our method provides not exactly the same results as the H.264/AVC Intra standard as this standard specifies the use of non-linear operations such as clipping and rounding in the prediction and reconstruction phase. The next sub section will discuss the influence of these operations and their effects on the decoded image in detail.

4.2.2. Effects of Non-Linear Operations on the Decoding Quality

This sub section discusses how drift between encoder and parallel decoder can be introduced by using non-linear operations in the intra prediction tool. For this, the clipping and rounding operations of the H.264/AVC Intra coding tool are investigated.

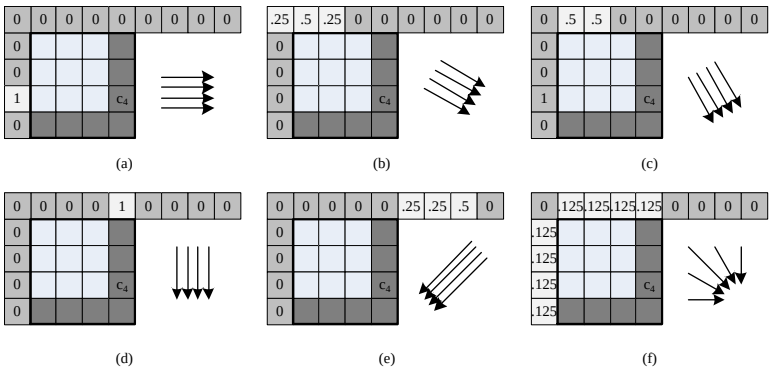


Figure 17: Example mappings of intra prediction modes in H.264/AVC to the μ vector for an edge coefficient c_4 at depth zero. (a): prediction from left, (b): prediction from top-left, (c): prediction from top (left), (d): prediction from top, (e): prediction from upper-right block, (f): DC prediction.

The H.264/AVC Intra standard states that after each sample prediction step, results are rounded upwards to the nearest integer and clipped to the range of the used bit depth, i.e., [0..255] for 8-bit values. However, as stated before, information on temporary predicted local values is limited in each prediction pass. Therefore, it is not possible to know whether to clip and round up or down a decimal value to the nearest integer before passing the coefficient to the successive pass. Unlike other block-based prediction methods such as the coefficient prediction in JPEG XR, clipping of prediction values is mandatory at the H.264/AVC decoder and not at the encoder. As a result, an H.264/AVC bitstream can cause prediction values outside the bit depth range (e.g., value 260).

Figure 18 shows the effect of omitting clipping of reconstructed values in the proposed parallel prediction method for a video sequence coded using JM 18[22]. Content containing a large number of dark and bright samples, for example the logo and helmet in the figure, show unclipped values in these areas to propagate to the lower right side and affect a large number of samples. Note that the video sequence was coded without the use of prediction modes capable of introducing rounding errors in the decoded output, the second non-linear operation in H.264/AVC intra prediction discussed in the next sub section. A number of sequences were encoded this way and the impact of decoding without clipping compared to the output of a standard-compliant decoder is shown in Table 2 and Table 3. Note that sequences

(a) (b)

Figure 18: Impact of eliminating the clipping of reconstructed values on the decoded output for an intra-coded video sequence (Foreman, CIF) encoded with JM 18 using one slice. (a): decoded using a standard-compliant decoder, (b): decoded without clipping using the proposed restricted two-dimensional algorithm; notice how the error starting above the helmet propagates to the bottom right.

Table 2: PSNR quality results (in dB) of decoding a number of video sequences using the proposed parallel decoding algorithm. The sequences were intra encoded with the DC prediction mode disabled.

Test Sequences	Bitrate (mbps)	Loop Filtering	PSNR Quality of Parallel Decoded Image (dB)								
			Y			Cb			Cr		
			min	avg	max	min	avg	max	min	avg	max
Foreman CIF	1.2	on	30.2	39.3	46.4	54.3	68.0	78.6	58.8	71.3	81.0
		off	30.4	39.9	46.4	54.3	67.9	78.2	58.8	71.4	81.4
	4.0	on	38.9	49.9	64.6	51.6	59.7	69.5	53.7	61.6	70.8
		off	38.9	50.0	64.8	51.6	60.1	71.1	53.8	61.9	71.5
Whale Show 480p	9.0	on	30.8	56.5	66.2	50.4	56.3	63.0	48.5	54.8	61.0
		off	30.8	56.5	65.6	50.4	56.8	63.5	48.5	54.8	61.6
	28.0	on	56.8	$+\infty$	$+\infty$	46.1	51.5	58.0	46.2	52.0	60.9
		off	56.8	$+\infty$	$+\infty$	46.1	51.6	58.3	46.3	51.7	58.7
Park Run 720p	36.0	on	38.9	51.0	63.2	54.7	62.6	72.2	56.3	64.4	75.3
		off	38.9	51.6	63.5	54.7	62.6	72.3	56.3	64.5	75.4
	90.8	on	51.8	73.0	87.1	50.9	55.0	60.4	51.8	55.0	59.3
		off	51.8	72.7	88.7	50.9	55.0	57.9	52.2	55.1	59.3
Tractor 1080p	21.6	on	67.3	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	63.7	$+\infty$	$+\infty$
		off	66.2	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	63.6	$+\infty$	$+\infty$
	69.2	on	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	74.8	$+\infty$	$+\infty$
		off	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	74.8	$+\infty$	$+\infty$

Table 3: Absolute distortion quality results (in absolute values) of decoding a number of video sequences using the proposed parallel decoding algorithm.

Test Sequences	Bitrate (mbps)	Loop Filtering	Absolute Distortion of Parallel Decoded Image								
			Y			Cb			Cr		
			min	avg	max	min	avg	max	min	avg	max
Foreman CIF	1.2	on	0.0	3.6	53.0	0.0	0.0	2.0	0.0	0.0	9.0
		off	0.0	3.3	50.0	0.0	0.0	2.0	0.0	0.0	2.0
	4.0	on	0.0	1.0	18.0	0.0	0.0	5.0	0.0	0.0	5.0
		off	0.0	1.0	9.0	0.0	0.0	3.0	0.0	0.0	3.0
Whale Show 480p	9.0	on	0.0	1.5	30.0	0.0	0.0	14.0	0.0	0.0	16.0
		off	0.0	1.5	22.0	0.0	0.0	3.0	0.0	0.0	3.0
	28.0	on	0.0	0.0	5.0	0.0	0.3	7.0	0.0	0.2	7.0
		off	0.0	0.0	5.0	0.0	0.3	6.0	0.0	0.2	4.0
Park Run 720p	36.0	on	0.0	0.9	37.0	0.0	0.0	6.0	0.0	0.0	2.0
		off	0.0	0.9	22.0	0.0	0.0	3.0	0.0	0.0	2.0
	90.8	on	0.0	0.0	8.0	0.0	0.0	7.0	0.0	0.0	6.0
		off	0.0	0.0	6.0	0.0	0.0	4.0	0.0	0.0	4.0
Tractor 1080p	21.6	on	0.0	0.0	7.0	0.0	0.0	0.0	0.0	0.0	3.0
		off	0.0	0.0	8.0	0.0	0.0	0.0	0.0	0.0	2.0
	69.2	on	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0
		off	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0

not showing bright or dark areas such as Blue Sky, show no clipping errors. The tables show the PSNR and absolute distortion metrics respectively in order to study both objective quality and maximum absolute differences in sample values. The absolute difference values are the absolute difference for each pixel calculated between the original standard-compliant and the parallel decoded pixel values. The tables show how the error introduced causes a drop in image quality, resulting in 37 dB for luma, and 70 dB for both chroma channels. The absolute distortion values show how the absolute average (equal to the rooted MSE of the picture) value is 5. The results also show that the error increases the lower the quality. This is to be expected, as with coarse quantization, the encoder will exploit the use of the clipping operations more to achieve values 0 and 255. Visual inspection of a number of sequences confirms that the decoding errors, although limited, hinder a good viewing experience. Hence, one restriction the algorithm imposes on video codecs targeting parallel decoding using the proposed algorithm is the use of clipping at the encoder by adjusting residual coefficients. The impact of such requirement is discussed in the next sub section.

The second non-linear operation in the H.264/AVC intra prediction tool is rounding of predicted values to the nearest integer value before passing the values on to predict the next block. As previously stated, the information on

Figure 19: Impact of eliminating rounding of reconstructed values on the decoded output for an intra-coded video sequence (Foreman, CIF) encoded with JM 18. (a): decoded without rounding using the proposed restricted two-dimensional algorithm, to the authors visually indistinguishable from the correctly decoded image, (b): view of error introduced by neglecting the rounding operations (black and white areas show correctly and faulty decoded samples respectively).

temporary predicted local values is limited in each prediction pass. Hence, it is not known yet whether to round up or down before passing the coefficient to the successive pass. With H.264/AVC, these rounding operations occur when a prediction mode other than vertical or horizontal prediction is used. Prediction modes such as the DC mode take a fraction of the surrounding coefficients, add up these fractions and round the result to the nearest integer value. This behavior is reflected in the proposed algorithm by using a μ vector with component values representing these fractions. For example, Figure 17(f) shows this for the DC prediction mode where the μ vector holds eight components with values equal to $\frac{1}{8}$. The rounding operation however can not be executed in our proposed algorithm. Figure 19 illustrates how this influences the decoding quality when disregarding rounding in H.264/AVC. Note that in this video picture, clipping was introduced at the encoder in order to focus on the impact of rounding errors. The figure shows how rounding errors (white area), introduced by, for example, the DC prediction mode, propagate to the lower-right corner because of the raster scan order of the standard. When erroneous values are used in subsequent DC predicted blocks, the error is reduced in size and can dissipate. The next DC block

Table 4: PSNR quality results (in dB) of decoding a number of video sequences using the proposed parallel decoding algorithm. The sequences were intra encoded with clipping of reconstructed values at the encoder.

Test Sequences	Bitrate (mbps)	Loop Filtering	PSNR Quality of Parallel Decoded Image (dB)								
			Y			Cb			Cr		
			min	avg	max	min	avg	max	min	avg	max
Foreman CIF	1.2	on	51.1	56.8	66.3	54.3	68.0	78.6	58.8	71.3	81.0
		off	48.1	57.3	64.2	54.3	67.9	78.2	58.8	71.4	81.4
	4.0	on	49.2	56.8	72.9	51.6	59.7	69.5	53.7	61.6	70.8
		off	49.9	56.4	73.9	51.6	60.1	71.1	53.8	61.9	71.5
Whale Show 480p	9.0	on	51.2	60.3	68.4	50.4	56.3	63.0	48.5	54.8	61.0
		off	51.3	60.3	69.3	50.4	56.8	63.5	48.5	54.8	61.6
	28.0	on	67.9	78.6	86.6	46.1	51.5	58.0	46.2	52.0	60.9
		off	68.1	80.8	103.5	46.1	51.6	58.3	46.3	51.7	58.7
Park Run 720p	36.0	on	46.3	52.3	58.3	58.5	61.1	64.2	57.9	65.6	71.7
		off	46.3	52.9	57.4	58.4	61.1	64.2	57.9	65.8	72.0
	90.8	on	65.9	72.9	77.1	53.9	54.9	55.9	53.4	54.9	57.1
		off	65.9	72.9	77.3	54.0	55.0	56.0	53.4	54.9	57.1
Tractor 1080p	21.3	on	48.5	51.5	54.5	50.4	52.3	54.1	45.8	49.5	51.0
		off	47.2	50.8	54.7	50.4	52.3	54.1	45.8	49.6	51.0
	69.0	on	56.0	58.1	59.4	47.2	50.0	53.7	47.6	48.9	50.6
		off	57.9	58.8	59.5	47.3	50.0	53.8	47.6	49.0	50.7

can then introduce a new rounding error, which can propagate further. We compare in Table 4 and Table 5 the output of the proposed algorithm to that of a standard-compliant decoder, using the PSNR and absolute distortion metrics respectively. The tables show how the error introduced causes a drop in image quality, resulting in 54 dB for luma, and 68 dB for both chroma channels. The absolute distortion values average at a rooted MSE of 0.1 absolute values. The tables show the error again to increase when bit rate drops. This is to be expected as for example, the DC prediction mode is chosen much more often with low bit rates. Errors introduced in the chroma components are limited compared to the luminance component because of the lack of detail in these components. Visually, no errors could be distinguished over all tested video sequences and resolutions.

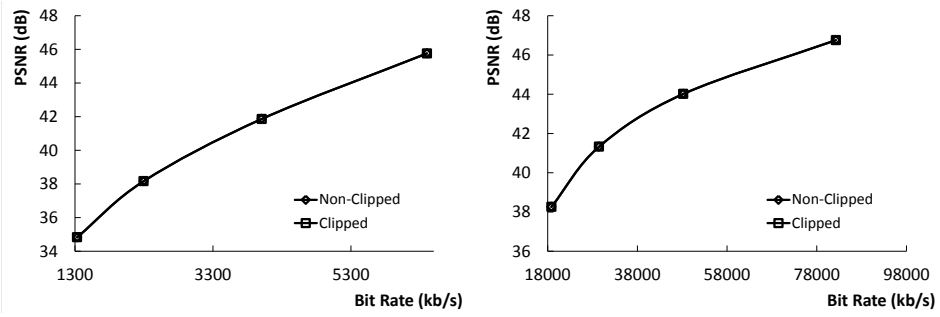
4.3. Required Parallel Intra Coding Tool Adaptations

In this sub section we show the impact of eliminating the non-linear operations from the decoding prediction tools on bit rate and quality. Again, we use the H.264/AVC standard as a model for such an image or video codec.

As previously stated, non-linear operations such as clipping and rounding introduce artifacts at the decoder. Clipping artifacts can easily be undone

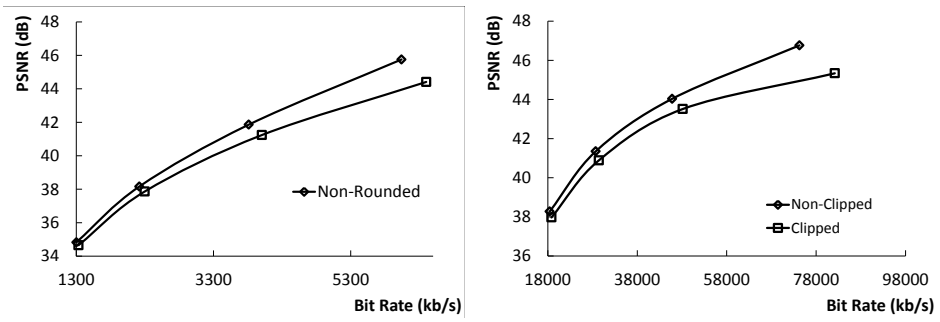
Table 5: Absolute distortion quality results (in absolute values) of decoding a number of video sequences using the proposed parallel decoding algorithm. The sequences were intra encoded with clipping of reconstructed values at the encoder.

Test Sequences	Bitrate (mbps)	Loop Filtering	Absolute Distortion Quality of Parallel Decoded Image								
			Y			Cb			Cr		
			min	avg	max	min	avg	max	min	avg	max
Foreman CIF	1.2	on	0.0	0.0	18.0	0.0	0.0	2.0	0.0	0.0	9.0
		off	0.0	0.2	15.0	0.0	0.0	2.0	0.0	0.0	2.0
	4.0	on	0.0	0.0	11.0	0.0	0.0	5.0	0.0	0.0	5.0
		off	0.0	0.0	3.0	0.0	0.0	3.0	0.0	0.0	3.0
Whale Show 480p	9.0	on	0.0	0.0	19.0	0.0	0.0	14.0	0.0	0.0	16.0
		off	0.0	0.0	16.0	0.0	0.0	3.0	0.0	0.0	3.0
	28.0	on	0.0	0.0	5.0	0.0	0.3	7.0	0.0	0.2	7.0
		off	0.0	0.0	1.0	0.0	0.3	6.0	0.0	0.2	4.0
Park Run 720p	36.0	on	0.0	0.6	19.0	0.0	0.0	5.0	0.0	0.0	2.0
		off	0.0	0.6	22.0	0.0	0.0	2.0	0.0	0.0	2.0
	90.8	on	0.0	0.0	6.0	0.0	0.0	6.0	0.0	0.0	4.0
		off	0.0	0.0	6.0	0.0	0.0	4.0	0.0	0.0	3.0
Tractor 1080p	21.3	on	0.0	0.0	15.0	0.0	0.0	10.0	0.0	0.4	16.0
		off	0.0	0.4	8.0	0.0	0.0	3.0	0.0	0.4	7.0
	69.0	on	0.0	0.0	7.0	0.0	0.6	7.0	0.0	0.6	6.0
		off	0.0	0.0	4.0	0.0	0.6	5.0	0.0	0.6	8.0



(a) (b)

Figure 20: Rate-distortion curves showing the impact of explicit clipping at the encoder for the (a), CIF Foreman, and (b), Blue Sky sequences.



(a) (b)

Figure 21: Rate-distortion curves showing the impact of predicting samples using fractional values for the (a), CIF Foreman, and (b), Blue Sky sequences.

by clipping at the encoder by incorporating the clipping operation in the residual values. One method is to predict samples using unclipped values and clip these values only as a post process after the reconstruction of an entire picture. An encoder will typically choose a residual coefficient causing reconstructed values outside of the valid range if this choice provides superior rate-distortion results. As a result, clipping at the encoder will increase the required bit rate as residual data will contain more energy to compensate for the unclipped predictions. Figure 20 shows the impact on rate-distortion by introducing explicit clipping at the JM 18 encoder in a number of rate-distortion curves for sequences Foreman (CIF), and Blue Sky (1080p). These sequences were chosen for their bright and dark areas which require clipping operations. As clearly visible from the graph, the impact of explicit clipping introduces little additional bit rate as non-clipped coefficients only occur occasional even with large bright and dark areas. Hence any codec targeting parallel decoding using the proposed method should have no clipping to allow parallel decoding.

The second non-linear operation is the rounding operation. By predicting sample values by using floating point arithmetic, encoder and decoder prediction can be synchronized and drift can be avoided. Note that as a final step in the proposed prediction algorithm, fractional sums are rounded to the nearest integer. At the encoder, residual values are calculated using these rounded values, but fractional versions are kept to predict the next blocks. Compared to the original prediction method in H.264/AVC, the fractional version can potentially underestimate the prediction values, and

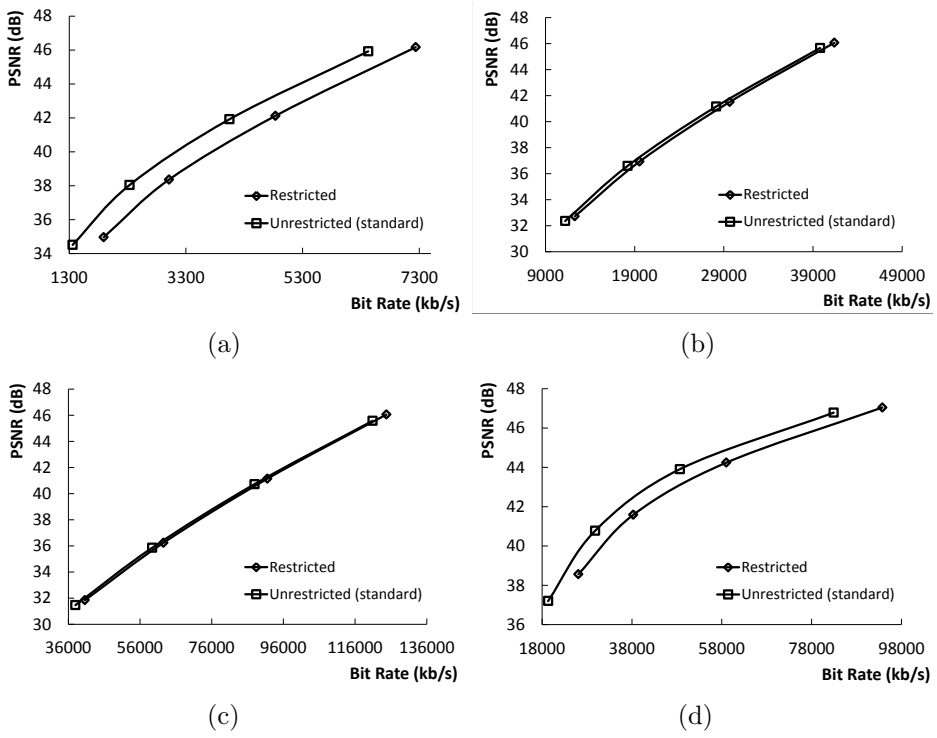


Figure 22: Rate-distortion curves showing the impact of predicting samples using only horizontal and vertical intra prediction modes for the (a), CIF Foreman, and (b), Whale Show, (c), Park Run, (d), Blue Sky sequence.

as such, the residual data is responsible to correct this underestimation. Figure 21 shows the impact of using fractional prediction on coding rate for two sequences. The results show that fractional prediction accomplishes better bit rates as prediction occurs with higher precision. We confirmed this behavior on other sequences as well, such as Whale Show, Shields, Park Run, Riverbed, and Rush Hour. Downside is that the encoding phase requires floating-point operations which typically take longer to process. Note however that on the GPU, floating point operations are as fast as any typical integer operations[19]. Here we can conclude that when using fractional prediction in order to target parallel intra decoding, no sacrifice in coding rate has to be made.

Lastly, we show the impact of restricting the complexity of the two-dimensional algorithm by limiting the μ vectors to a single component vector. As a result, for the H.264/AVC intra prediction tool, only two prediction

modes remain, namely left and horizontal. Figure 22 shows the impact on rate-distortion when disabling all but these prediction modes. As the graphs shows, this clearly influences the rate-distortion characteristics and introduces an maximum loss of 2dB in quality depending on the content. Here the trade off between complexity and coding efficiency is the most profound. Indeed, as will become clear in the next section when performance results are discussed, reducing the number of input coefficients lowers the complexity of the algorithm and hence results in faster performance.

4.4. Implementation and CUDA Thread Mappings for the two-dimensional algorithm

This section investigates the implementation and CUDA thread mapping of the proposed two dimensional processing scheme. The two-dimensional scheme uses multiple thread mappings, each tuned for the amount of communication and coefficients that need processing. The limited intra prediction scheme is discussed first. Here, the first up sweeps use a sum unit-to-coefficient mapping, where each thread is responsible for all coefficients in a sum unit, i.e., a block of size $d \times d$ at depth d. This thread iterates all coefficients and resolves the dependencies to the coefficients of the other inner quadrants as shown in Figure 13. For example, the up pass 3, which groups four 4×4 blocks into one 8×8 block, uses one thread to first resolve the seven coefficients of the second quadrant, next uses this thread to resolve all seven coefficients from quadrant 3, and finally uses this thread to resolve all seven coefficients from quadrant 4 (using the previously-resolved coefficients from quadrants 2 and 3). For an image of 2048×2048, this mapping provides 65536 threads, mapped on CUDA blocks of 256 threads and a CUDA grid of 256 CUDA blocks. This mapping is used until there are 32 or more coefficients per sum unit. Indeed, from a 64×64 sum unit in pass 7, each CUDA block calculates one sum unit and each thread of such CUDA block is responsible for resolving one coefficient. This way, for example for pass 9 targeting 256×256 sum units, a 2048×2048 image is processed using a CUDA block configuration of 128 threads and a CUDA grid configuration of 64 blocks, providing enough parallelism to occupy 8192 threads. In order to guarantee raster scan order of the inner quadrants, CUDA block synchronization is used to synchronize all threads of the CUDA block before proceeding to process the next quadrant. This synchronization method is quite efficient[5] compared to global synchronization using kernel invocations. Finally, when the number of coefficients within a CUDA block becomes too large and the num-

ber of sum units within an image becomes too low, more CUDA blocks are scheduled for each sum unit. Each CUDA block now targets a specific set of coefficients of the inner quadrants of the sum unit. Here, global synchronization is required to guarantee the raster-scan order processing of quadrants. Hence, for each pass, three kernel invocations are scheduled.

The unrestricted two-dimensional scheme follows almost the same thread mapping as the restricted version, but CUDA blocks are used sooner to resolve a single coefficient. For example, when the dimensionality of the vector v becomes larger than 32, a CUDA warp, i.e., 32 threads, is now responsible to resolve the 32 dependency coefficients. This way, a CUDA block is responsible resolving up to 32 coefficients. As a result, synchronization between resolving the four quadrants of each sum unit is done by invoking three CUDA kernels.

Table 6 shows for a number of image resolutions the total number of kernel invocations and therefore synchronizations points for the up and down sweep as well as for the wavefront method. It can be clearly seen in the table that for block sizes of for example 4×4, the number of synchronizations points required by the proposed algorithm is significantly less than for the wavefront algorithms. The table also shows how the number of synchronizations scales logarithmically with our algorithm instead of linearly.

Images are padded to fit a power of two memory buffer. However, only the number of threads corresponding with the actual width and height are started to process the image. For the one-dimensional and restricted two-dimensional algorithm, the amount of extra memory is limited. Passes write information in place in a single memory buffer which contains two floating point numbers and an integer. The unrestricted two-dimensional algorithm however requires a multitude of memory of the unrestricted algorithm as each coefficients now needs to store a multiple of input coefficients. Here, we allocate a new memory buffer on the GPU for each pass able to contain the maximum amount of input coefficients. This maximum is typically defined by the image height and width as follows: $(width + height)^2$. Compared to a serial implementation, the proposed algorithm is memory intensive, as each thread will load its coefficient from GPU memory, load its dependencies, and write the resulting coefficient back to GPU memory. However, the GPU architecture is prepared for these kinds of applications at it provides a large memory bandwidth and hardware mechanisms to specifically hide memory latency[19].

Table 6: Number of kernel invocations and synchronizations required for the proposed two-dimensional method compared to the wavefront method.

Res.	Proposed two-dimensional			NW Wavefront		NNW Wavefront	
	Up Sweep	Down Sweep	Total	1×1 blocks	4×4 blocks	1×1 blocks	4×4 blocks
CIF	9	9	18	639	159	926	230
480p	12	12	24	1199	299	1678	418
720p	15	15	30	1999	499	2718	678
1080p	15	15	30	2999	749	4078	1018
2160p	18	18	36	5999	1499	8158	2038

5. Performance Results

In this section, results are presented in terms of processing speed by comparing the execution time of our data-parallel prediction scheme to that of the state-of-the-art wavefront method. We evaluated the proposed one-dimensional algorithm using implementations of the JPEG XR LP and H.264/AVC QP prediction. For these prediction tools, the proposed scheme enables standard-compliant decoding. The two-dimensional algorithm is evaluated using an implementation of the H.264/AVC Intra prediction tool. As previously discussed in sub section 4.2.2, the proposed scheme allows near-lossless decoding for this standard. Specifically, we compared the execution time of the two-dimensional algorithm using the H.264/AVC Intra unrestricted prediction (e.g., DC mode enabled), and H.264/AVC Intra restricted mode prediction. Prediction block sizes were limited to 4×4. As a computing platform, the massively-parallel architecture of the GPU was chosen, more specifically, an NVIDIA GeForce GTX 480 card with 480 streaming processors using the NVIDIA CUDA Platform[19]. The test system was installed with an Intel i7 950 processor, Windows 7, and the NVIDIA CUDA Toolkit 3.2.

Table. 7 shows the increase of decoding speed of the proposed data-parallel prediction algorithm over the wavefront method for the previously-discussed prediction schemes, using a number of picture resolutions. Note that all but the unrestricted H.264/AVC parallel schemes are content unaware in term of processing speed. For the H.264/AVC unrestricted method, we show two values for two scenarios. First, a typical scenario with up to 25% blocks predicted using DC prediction, averaged over three video sequences, and second a worst-case content-independent approach with all blocks coded using DC prediction. Also, the amount of parallelism in our prediction scheme is independent of the used block group configuration (slices

Table 7: Experimental average results for the proposed parallel algorithm compared to the wavefront method.

Resolution	JPEG XR LP	H.264/AVC QP	H.264/AVC Intra		
			Unrestricted		Restricted
			Wost-case	Typical	
480p	7.7x	14.6x	0.7x	1.1x	4.0x
720p	9.5x	16.1x	0.8x	1.4x	4.9x
1080p	14.0x	17.9x	1.0x	2.1x	6.7x
2160p	21.5x	18.3x	1.3x	2.2x	7.9x

or tiles), but is dictated by the picture resolution. In our tests, we used one intra-slice-coded picture for H.264/AVC Intra and a JPEG XR image coded with one tile, to simulate a worst-case scenario.

The results shows our algorithm to outperform the wavefront method for both video and image coding standards except for the worst-case unrestricted H.264/AVC scenario. For the one-dimensional prediction method (JPEG XR, H.264/AVC QP prediction) the graph shows our method to outperform the wavefront method with a factor of 18.3 to 21.5. As stated, this is mainly because of the maximization of parallelism. Indeed, it can be seen how processing speed increases when resolution increases as the processing speed of the wavefront algorithm does not scale linearly with the number of coefficients in the video picture. Our two-dimensional prediction method outperforms the wavefront method by a factor of 2.2x for the typical H.264/AVC intra prediction scenario for 2160p video pictures. This is again because of the maximization of parallelism, but most importantly because the proposed algorithm limits synchronization, a bottleneck for wavefront calculations on the GPU. The table also clearly shows how processing speed increases for H.264/AVC decoding from a factor of 2.2 to 7.9 times faster than wavefront when restricting the number of prediction modes to only using horizontal and vertical prediction. This was to be expected as with this approach, each coefficient only needs to hold one input dependency. This significantly reduces calculations as the remapping of dependency coefficients is limited to a single multiplication. It is to be expected that the more processing cores come available on the GPU, the more the performance of the unrestricted mode will approach that of the restricted, as more dependency vectors can be calculated in parallel and our algorithm scales with the number of processors.

6. Conclusion

In this paper we proposed a novel data-parallel processing scheme for sample or coefficient prediction for block-based image and video codecs. Our parallel processing scheme maximizes parallelism independent of slice or tile configuration, while minimizing synchronization points opposed to the state-of-the-art wavefront method, making it suited for massively-parallel architectures such as the GPU. We used the intra coding tools of the JPEG XR and H.264/AVC Intra standards to show how our proposed algorithm can speed up prediction for block-based image and video codecs. We investigated the requirements of the parallel prediction method and analyzed the decoding errors introduced if non-linear prediction operations are not avoided in the prediction tools. We also showed the effects of eliminating these non-linear operations on the coding bitrate. Finally, our results show a speedup factor compared to the wavefront method of up to 21.5 and 7.9 for respectively JPEG XR and H.264/AVC Intra when executed on the massively-parallel architecture of the GPU.

[1] N.-M. Cheung, X. Fan, O. C. Au, M.-C. Kung, Video Coding on Multi-Core Graphics Processors, IEEE Signal Processing Mag. 27 (2) (2010) 79–89.

[2] G. Shen, G. ping Gao, S. Li, H. yeung Shum, Y. qin Zhang, Accelerate Video Decoding with Generic GPU, IEEE Trans. Circuits Syst. Video Technol. 15 (2005) 685–693.

[3] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference, ACM, New York, NY, USA, 1967, pp. 483–485. doi:http://doi.acm.org/10.1145/1465482.1465560.

[4] B. Pieters, C.-F. Hollemeersch, J. De Cock, P. Lambert, W. De Neve, R. Van de Walle, Parallel deblocking filtering in mpeg-4 avc/h.264 on massively parallel architectures, Circuits and Systems for Video Technology, IEEE Transactions on 21 (1) (2011) 96 –100.

[5] W. Feng, S. Xiao, To GPU Synchronize or Not GPU Synchronize?, in: IEEE International Symposium on Circuits and Systems (ISCAS), Paris, France, 2010.

[6] B. Jung, B. Jeon, Adaptive slice-level parallelism for h.264/avc encoding using pre macroblock mode selection, J. Vis. Comun. Image Represent. 19 (8) (2008) 558–572. doi:http://dx.doi.org/10.1016/j.jvcir.2008.09.004.

[7] M. Roitzsch, Slice-balancing h.264 video encoding for improved scalability of multicore decoding, in: EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software, ACM, New York, NY, USA, 2007, pp. 269–278. doi:http://doi.acm.org/10.1145/1289927.1289969.

[8] T. Jacobs, V. Chouliaras, D. Mulvaney, Thread-parallel MPEG-2, MPEG-4 and H.264 video encoder for SoC multi-processor architectures, IEEE Trans. Consumer Electron. 52 (1) (2006) 269–275.

[9] E. B. van der Tol, E. G. Jaspers, R. H. Gelderblom, Mapping of h.264 decoding on a multiprocessor architecture, Vol. 5022, SPIE, 2003, pp. 707–718.

[10] M. A. Baker, P. Dalale, K. S. Chatha, S. B. Vrudhula, A scalable parallel h.264 decoder on the cell broadband engine architecture, in: CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis, ACM, New York, NY, USA, 2009, pp. 353–362. doi:http://doi.acm.org/10.1145/1629435.1629484.

[11] J. C. A. Baeza, W. Chen, E. Christoffersen, D. Dinu, B. Friemel, Real-Time High Definition H.264 Video Decode Using the Xbox 360 GPU, in: Proc. of SPIE: Applications of Digital Image Processing XXX, no. 1, 2007.

[12] Z. Zhao, P. Liang, Data Partition for Wavefront Parallelization of H.264 Video Encoder, in: IEEE International Symposium on Circuits and Systems, 2006.

[13] F. H. Seitner, M. Bleyer, M. Gelautz, R. M. Beuschel, Evaluation of data-parallel H.264 decoding approaches for strongly resource-restricted architectures, Multimedia Tools and Applications (2009) 1380–7501.

[14] J. Chong, N. Satish, B. Catanzaro, K. Ravindran, K. Keutzer, Efficient Parallelization of H.264 Decoding with Macro Block Level Scheduling,

in: IEEE International Conference on Multimedia and Expo, 2007, pp. 1874–1877.

[15] G. Amit, A. Pinhas, Real-Time H.264 Encoding by Thread-Level Parallelism: Gains and Pitfalls., in: IASTED PDCS, 2005, pp. 254–259.

[16] G. E. Blelloch, Scans as Primitive Parallel Operations, IEEE Trans. Comput. 11 (38) (1989) 1526–1538.

[17] S. Srinivasan, C. Tu, S. L. Regunathan, G. J. Sullivan, HD Photo: A New Image Coding Technology for Digital Photography, in: Proc. of SPIE: Applications of Digital Image Processing XXX, Vol. 6696, 2007.

[18] ITU-T Rec. T.832 — ISO/IEC 29199-2, Information technology JPEG XR image coding system Image coding specification.
URL `http://www.itu.int/rec/T-REC-T.832`

[19] NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture: Programming Guide Version 3.2 (October 2010).

[20] NVIDIA Corporation, NVIDIA CUDA Performance Primitives, CUDA SDK 3.2 (October 2010).

[21] Joint Video Team (JVT) of ITU-T and ISO/IEC JTC 1, Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), Version 1, May 2003; Version 2, January 2004; Version 3 (with FRExt), September 2004; Version 4, July 2005; Version 5, July 2007 (with Intra Profiles and SVC).

[22] Jm 17.2 test mode codec, ISO/IECMPEG and ITU-T VCEG Joint Video Team.
URL `http://iphome.hhi.de/suehring/tml`