ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.7, 1–7

Data dependency reduction for parallelism enhancement of HEVC decoder

Song Hyun Jo[a)] **and Yong Ho Song**

Department of Electronics and Computer Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul 133–791, Korea

a) *shjo@enc.hanyang.ac.kr*

Abstract: HEVC is a video codec which yields higher coding efficiency compared to its predecessors. This efficiency improvement has been realized by adopting many advanced algorithms to its coding tools which often require a huge amount of computation. Beside, HEVC is designed to be applicable mainly to high resolution videos that necessitate the enormous amount of computation. One common solution to this problem is to execute the algorithms in a parallelized way. However, the data dependency between neighboring coding tree units, the basic decoding unit in HEVC, limits the level of parallelization in Intra Prediction. This paper proposes a dependency reduction technique which identifies virtual dependencies among coding tree units via runtime analysis and eliminates them to enhance potential parallelism. The experimental results show that the performance of Intra Prediction can be significantly improved by lifting such virtual dependencies

Keywords: HEVC, parallelism, intra prediction

Classification: Electron devices, circuits, and systems

References

[1] G. J. Sullivan, J. Ohm, W.-J. Han and T. Wiegand: IEEE Trans. Circuits Syst. Video Technol. **22** [12] (2012) 1649.

[2] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan and T. Wiegand: IEEE Trans. Circuits Syst. Video Technol. **22** [12] (2012) 1669.

[3] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux and T. Schierl: IEEE Trans. Circuits Syst. Video Technol. **22** [12] (2012) 1827.

[4] S. H. Jo, S. Jo and Y. H. Song: IEEE Trans. Consum. Electron. **56** [3] (2010) 1963.

[5] H.-H. Jo, Y.-J. Ahn, D.-B. Kang, B. Ji, D.-G. Sim and J.-J. Lee: J. Signal Process. Syst. Signal Image Video Technol. (2013).

[6] M. R. Garey and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman and Co., New York, 1979).

[7] F. Bossen: JCTVC-I1100, ISO/IEC-JCT1/SC29/WG11 (2012).

[8] F. Bossen: JCTVC-L1100, ISO/IEC-JCT1/SC29/WG11 (2013).

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014

1 Introduction

High Efficiency Video Coding (HEVC) [1] is the state-of-the-art video codec standard providing a higher coding efficiency than the previous ones (e.g. H.264/AVC) especially for a high resolution video [2], which requires a significant amount of computation to be coded.

High performance processors are often used to improve the execution efficiency of video codecs. One of the popular techniques to enhance the computation performance in video decoding is to divide the computation into smaller chunks and execute them in a *parallelized way* [3, 4, 5]. However, the performance enhancement is often limited by the dependency residing between data. That is, the data dependency often results in the serialization of computation and therefore decreases the utilization of processing elements.

Much previous research on parallelized decoding of HEVC bitstreams has been conducted to find a way to concurrently process multiple coding tree units (CTUs) [3], where a CTU is the basic decoding unit in HEVC. All of them have tried to exploit the best of inherent parallelism in video streams with the assumption that the data dependency between neighboring CTUs *presumed* by the intra prediction tool is *really* existent. However, some of the *presumed data dependencies* (PDs) are *non-existent* (or *virtual*) in fact. If the *virtual data dependencies* (VDs) can be identified and eliminated, the intra prediction could exploit better parallelism.

This paper proposes a dependency reduction technique that identifies VDs among CTUs in the intra prediction, and eliminates them to enhance and exploit the parallelism. To identify the VDs, the proposed technique draws the *actual data dependency* (AD) by analyzing the dependencies of the coding units (CUs) in a CTU. And then, the HEVC decoding parallelization is performed based on the AD.

The experimental results show that the proposed technique achieves 1.23x speeds up on 6-core processor and 5.74x speed up on many-core processors, when compared to the existing techniques.

2 Backgrounds

2.1 Data hierarchy in HEVC

HEVC uses the hierarchical data structure as depicted in Fig. 1. A video sequence consists of single or multiple groups of pictures (GOPs), each of which is then partitioned into a number of frames. A frame contains one or more slices/tiles that are further divided into multiple CTUs. The HEVC standard defines the CTUs to be processed in a *raster-scan* order. A CTU consists of CUs in a quad-tree (called *coding tree*) fashion, as shown in Fig. 1 [3]. In the tree (not shown in the figure due to the page limit), a leaf node represents a CU and all intermediate nodes are the collection of CUs. CUs are categorized into one of three types: *intra-coded CU*, *inter-coded*

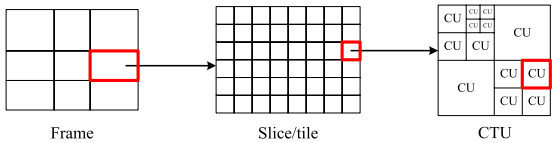

Fig. 1. Hierarchical data structure in HEVC.

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014

CU and *skipped CU*. Among them, the intra prediction processes *only* the intra-coded CUs.

2.2 Processing tasks in an HEVC decoder

The HEVC decoding process can be divided into three steps: the *entropy decoder* (Step 1), the *intra/inter prediction and inverse transform* (Step 2), and the *deblocking filter* and *sample adaptive offset* (SAO) (Step 3). The proposed parallelization technique can be applicable to Step 2. Considering that Step 2 takes about 44% of total decoding time on average [3], the parallelization of Step 2 is therefore expected to have significant impact on performance enhancement.

Among the three coding tools in Step 2, only intra prediction is affected by the CTU-level data dependency. The amount of dependency exposed to intra prediction is the dominant factor in the performance enhancement.

In the proposed parallelism, Step 2 is performed in two phases. Phase 1 is to discover ADs among CTUs in a given frame, while Phase 2 is to perform the parallelized decoding of CTUs. The decoding process uses multiple threads for the high performance. The CTUs are allocated to threads based on the ADs discovered at Phase 1.

3 CTU-level data dependency reduction

3.1 CTU-level dependency analysis

A CTU could have four PDs between neighboring CTUs, as shown in Fig. 2 (a). Some of PDs are actually non-existent. If not identified, the VDs would cause excessive serialization in the decoding process.

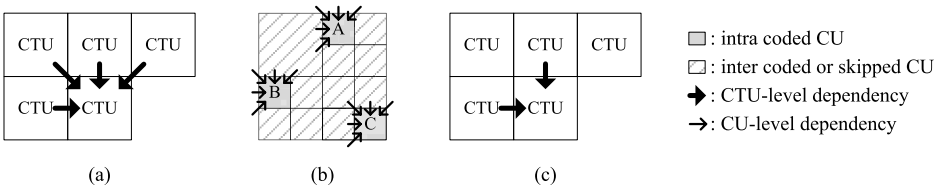

Fig. 2. Data dependencies for a CTU: (a) presumed data dependencies, (b) CU-level data dependency inside the CTU and (c) actual data dependencies.

The identification of VDs requires analyzing how an inter-CTU dependency is created. Fig. 2 (b) shows the CUs in a CTU and the dependencies between them. In this example, three CUs (A, B, and C) are intra-coded. The other CUs are not subject to CU-level data dependency. Now, it is possible to deduce the actual CTU-level data dependency from CU-level dependencies. For example, CU A has five CU-level dependencies. However, three dependencies from the left-upper, upper and right-upper sides are related with the upper CTU. These CU-level dependencies are, therefore, converted into an upper-side CTU-level dependency, as shown in Fig. 2 (c). A similar conversion can be made on CU B. In this case, the only difference is the direction of dependency. In the case of CU C, there exist the dependencies to the right and left-lower CTUs. However, these CTUs are not decoded yet, so these dependencies can be ignored. For this reason, the CU-level dependency in CU C does not need to be converted into CTU-level dependency. Finally, only upper-side and left-side CTU-level depen-

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014

dencies remain, as depicted in Fig. 2 (c). In other words, the original PDs include VDs to the left-upper and right-upper CTUs.

The deduction of CTU data dependencies from CU dependencies is done based on the location of a CU in the CTU. A dependency pattern deduced from a CU is called an *AD deduction type (ADDT)*. The deduction types are listed in Table I. Each row in the table represents an ADDT. The first column is the type number for each ADDT. The other columns indicate the existence of CTU-level dependencies in the four directions. For example, the ADDT 1 is the case when the CU has a dependency with the left CTU, like CU B in Fig. 2 (b). According to the table, CU A and C are ADDT 3 and 0, respectively. The ADDTs are illustrated in Fig. 3.

Table I. ADDTs for a CU.

Type Number	AD				Location of a CU in CTU
	L	UL	U	UR	
0	X	X	X	X	Not at left edge nor upper edge
1	O	X	X	X	Left edge except left upper edge
2	O	O	O	X	Left upper edge
3	X	X	O	X	Upper edge, except left upper and right upper edge
4	X	X	O	O	Right upper edge
5	O	O	O	O	The size of the CU is same as the CTU

L, *UL*, *U* and *UR* represent CTU-level dependency from left, upper-left, upper and upper-right CTUs, respectively. The mark *O* indicates that the corresponding CTU-level data dependency exists.

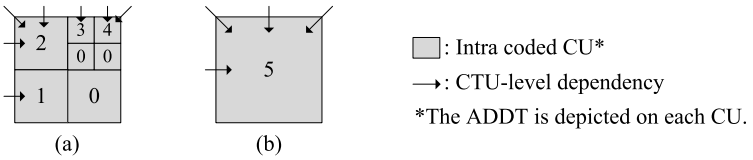

Fig. 3. The ADDTs of CUs in a CTU. (a) ADDT 0, 1, 2, 3, and 4, and (b) ADDT 5.

3.2 Dependency deduction method

This section explains our dependency deduction method to discover ADs of a given CTU. Once we know the location of a CU, it is easy to deduce its dependency information. However, the challenge is to figure out the location of a CU, because no information on the location is included in a bitstream. Instead, a bitstream only contains syntax elements that indicate how a CTU is partitioned into CUs.

Now, the proposed method works as follows. For a given CTU, it traverses the coding tree from the root towards leaf nodes, and identifies the ADDTs of all the nodes. Once an ADDT for a node is identified, the child nodes, if any, are destined to have a set of pre-determined ADDTs, which will be explained below. This process repeats until all the leaf nodes are visited. Now, the ADDT of an intra-coded CU is translated to CTU-level dependencies using the Table I. The ADs of a CTU can be identified using by merging the ADs of the intra-coded CUs.

As aforementioned, the ADDT of a node in a coding tree determines the ADDTs of its children, as listed in Table II. In this table, the leftmost column is the ADDT of a parent node and the other columns are the ADDTs of the four child nodes. The child nodes are located at north-west (NW), north-east (NE), south-west (SW) and south-east (SE) quadrants.

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014

Table II. ADDT inheritance from a parent node to child nodes.

ADDT of Parent Node	ADDTs of Child Nodes			
	NW	NE	SW	SE
0	0	0	0	0
1	1	0	1	0
2	2	3	1	0
3	3	3	0	0
4	3	4	0	0
5	2	4	1	0

An example of the determination of ADDT for child nodes is shown in Fig. 4. The ADDT of a root node in a coding tree is always 5 (see Table I). Then, the ADDTs of four child nodes should be 2, 4, 1 and 0 in the order of NW, NE, SW and SE, as illustrated in Fig. 4 (b). If the NW quadrant has four child nodes once again, as in Fig. 4 (c), the child nodes should have 2, 3, 1, and 0 as ADDT. Similarly, the NE and SW quadrants can be divided a bit further. Note that once a node has 0 as ADDT, its children do not require any more traverse. In order to deduce the ADs, the proposed method traverses the coding tree of a given CTU with the depth-first search (DFS). The algorithm of dependency deduction is presented in Fig. 5. The return value of *addt_table*[*t*] is ADs as shown in Table I. and *inheritance_table* is a coded version of Table II.

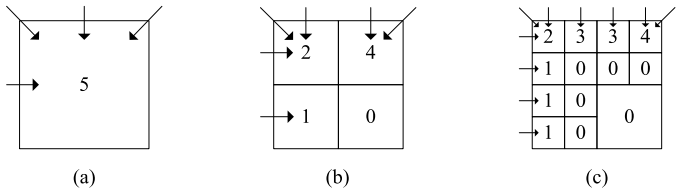

Fig. 4. An example of ADDT inheritance from a parent node to child nodes.

Input: *c* (CTU), **Output:** *d* (a set of dependency contained in AD)

```
DDM (c)
01:  d ← ∅
02:  Call Coding_tree_DFS (c, 5)

Coding_tree_DFS (u, t) // u: current unit, t: an ADDT of the unit
03:  if u is a CU then
04:      if u is an intra-coded CU then
05:          d ← d ∪ addt_table [t]
06:      end if
07:  else
08:      for each child unit w in the u
09:          x ← inheritance_table [t][the quadrant of w]
10:          if x is not 0 then
11:              call Coding_tree_DFS (w , x)
12:          end if
13:      end for
14:  end if
```

Fig. 5. The pseudo code of the algorithm for dependency deduction.

4 Dependency-aware scheduling

Our parallel HEVC decoder models the ADs of CTUs using a *directed dependency graph* (DDG) to easily identify which CTUs are ready for scheduling. In this representation, a node represents a CTU and a uni-

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014

directional edge represents a data dependency between CTUs. Once a node is processed, it is detached from the graph along with all the edges that originate from the node. A node becomes ready for scheduling, which is denoted as a *ready node*, only when all the precedent nodes are processed (no in-bound edges remain). If there are multiple nodes schedulable, they can be processed in parallel.

The DDG-based scheduling can be applicable to Step 2. When a processing unit becomes idle, it fetches one of the CTUs that is ready for scheduling. Once a CTU is decoded, the dependencies from the CTU become eliminated. In Fig. 6, for example, when the decoding of CTU *0* has been decoded, three dependencies from the CTU are removed. Consequently, the CTU *1* and *5* are ready after CTU *0* is decoded. The scheduling efficiency depends on the selection of CTUs to be processed first. The optimal selection is a *NP-complete* problem [6]. And also, the optimal algorithm for this problem is not mandatory. Instead, we decide to rely on a simple replacement: selecting a CTU in a raster-scan order in a frame.

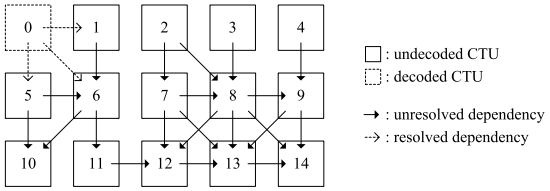

Fig. 6. A DDG example to represent CTU-level data dependencies.

5 Experiments

The test sequences from Class A (2560p) and B (1920p) specified in [7] were encoded using an HM 10.0 [8] with main profile and random access coding structure. Each video is encoded four times with four quantization parameter (QP) values (22, 27, 32, and 37). We implement the proposed method on top of HM 10.1 decoder using OpenMP 2.5; the performance is measured on a six-core processor, Intel Core i7. In addition, a simulator has been developed to estimate the performance enhancement for modeling a homogeneous many-core system. The simulation model assumes that the decoding time is the same for all CTUs and that the memory system causes no performance degradation.

To compare the characteristics of PD- and AD-DDG in parallel processing, the length of critical path (LCP) and the number of CTU-level data dependency (NCD) are analyzed. In the case of the 1920p sequence, the LCP and NCD of AD-DDGs are only 20.1% and 11.0% on average compared to that of PD-DDG, respectively. The decrease in NCD and LCP implies that the parallelization based on AD (denoted as *ADP*) is likely to yield better performance than the one based on PD (denoted as *PDP*).

The execution time of our parallelized decoder has been profiled on a real system for all test sequences. The average speedups for the 2560p and 1920p sequences are illustrated in Fig. 7. For all sequences, the performance is better for ADP than for PDP.

The ADP requires the *runtime* discovery of ADs (Phase 1). The

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014

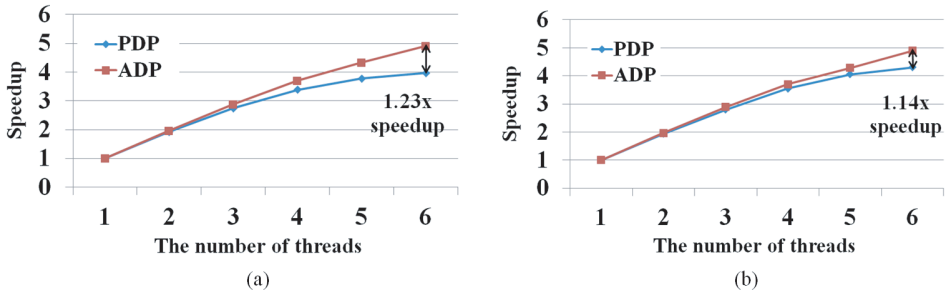

Fig. 7. The parallelization speedup on a six-core system. (a) The average speedup of 1920p sequences. (b) The average speedup of 2560p sequences.

discovery overhead takes only 0.8% of the sequential decoding time in Step 2. Also, it can be easily reduced by making the discovery process parallelized in many-core systems.

To examine the performance scalability, the execution time of the proposed parallelism is simulated for a large number of cores as shown in Fig. 8. The performance difference between ADP and PDP is dramatically increases after eight cores. Thus, the ADP becomes more effective as the number of cores increases.

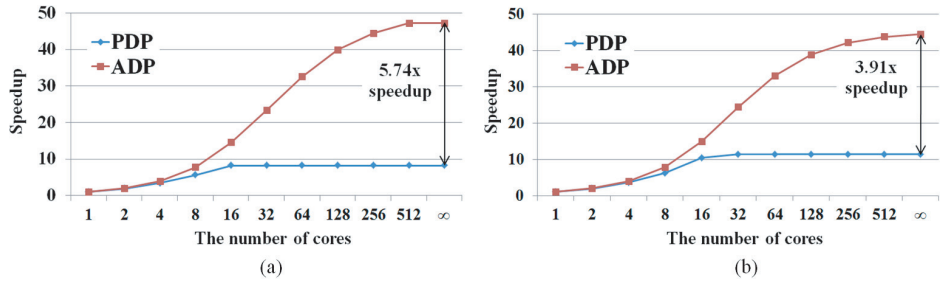

Fig. 8. The parallelization speedup on many-core systems. (a) The average speedup of the 1920p sequences, and (b) the average speedup of the 2560p sequences.

6 Conclusions

This paper presents a novel technique that eliminates the virtual data dependencies between CTUs to increase the performance benefit of parallelized HEVC decoding. The experimental results show that the parallelized decoding with the elimination of virtual data dependencies contributes to achieve a significant speedup.

Acknowledgments

This work was supported by the Ministry of Science, ICT & Future Planning (MSIP) of Korea, under the Information Technology Research Center (ITRC) support program supervised by the National IT Industry Promotion Agency (NIPA) (NIPA-2013-H0301-13-1011).

DOI: 10.1587/elex.11.20140027
Received January 14, 2014
Accepted March 05, 2014
Publicized March 14, 2014
Copyedited April 10, 2014