symmetry

MDPI

Article

Edge Intelligence-Assisted Asymmetrical Network Control and Video Decoding in the Industrial IoT with Speculative Parallelization

Shuangye Yang [1,2,3,*], Zhiwei Zhang [2,3], Hui Xia [2,3], Yahui Li [2,3] and Zheng Liu [4]

[1] School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China
[2] CNPC Baoji Oilfield Machinery Co., Ltd., Baoji 721002, China; bszzw@cnpc.com.cn (Z.Z.); xiahui999@cnpc.com.cn (H.X.); liyahui999@cnpc.com.cn (Y.L.)
[3] CNPC National Engineering Research Center for Oil & Gas Drilling Equipment Co., Ltd., Baoji 721002, China
[4] School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China; liuzheng@xaut.edu.cn
* Correspondence: yangshuangye@cnpc.com.cn

Abstract: Industrial Internet of Things (IIoTs) has drawn significant attention in the industry. Among its rich applications, the field's video surveillance deserves particular interest due to its advantage in better understanding network control. However, existing decoding methods are limited by the video coding order, which cannot be decoded in parallel, resulting in low decoding efficiency and the inability to process the massive amount of video data in real time. In this work, a parallel decoding framework based on the speculative technique is proposed. In particular, the video is first speculatively decomposed into data blocks, and then a verification method is designed to ensure the correctness of the decomposition. After verification, the data blocks having passed the validation can be decoded concurrently in the parallel computing platform. Finally, the concurrent decoding results are concatenated in line with the original encoding order to form the output. Experiments show that compared with traditional serial decoding ones, the proposed method can improve the performance by 9 times on average in the parallel computing environment with NVIDIA Tegra 4 chips, thus significantly enhancing the real-time video data's decoding efficiency with guaranteed accuracy. Furthermore, proposed and traditional serial methods obtain almost the same peak signal-to-noise ratio (PSNR) and mean square error (MSE) metrics at different bit rates and resolutions, showing that the introduction of the speculative technique does not degrade the decoding accuracy.

Keywords: edge intelligence; asymmetrical network control; video decoding; speculative parallelization; IIoT

Citation: Yang, S.; Zhang, Z.; Xia, H.; Li, Y.; Liu, Z. Edge Intelligence-Assisted Asymmetrical Network Control and Video Decoding in the Industrial IoT with Speculative Parallelization. *Symmetry* **2023**, *15*, 1516. https://doi.org/10.3390/sym15081516

Academic Editors: Chao Fang, Peng Li and Christos Volos

Received: 8 June 2023
Revised: 9 July 2023
Accepted: 26 July 2023
Published: 1 August 2023

1. Introduction

The merging of Internet of Things (IoTs) with industrial manufacturing, i.e., industrial IoTs (IIoTs), has drawn great interest from both academia and industry, especially in the data collection from IIoT devices [1,2]. Among the rich industrial IoT applications, the ultra high-definition video surveillance in the field deserves special attention due to its advantages in better understanding the field operation and alerting for possible failures. More precisely, various IIoT devices, e.g., cameras or unmanned aerial vehicles (UAVs) can monitor the field operation on the edge of the industry by means of the space-air-ground integrated network (SAGIN). Nevertheless, the transmission of surveillance data is non-trivial, especially in remote sites far away from central cloud data centres. Thus, it is advocated that the streaming video is processed locally. Further, the industry calls for a fast way to make decisions at the edge, and the answer lies in edge computing. By incorporating the prevailing edge intelligence and machine learning into the field, the real-time decision-making and operational intelligence are enabled using video data [3].

The video data has become one major component in the industry [3], and the monitoring video decoding has drawn great attention in terms of low-density parity-heck code (LDPC) [4], in MPEG-5 [5], and in the dense video captioning [6]. China has a tremendous digital audio and video industry, yet with weaker intellectual property rights, which have long been by MPEG-2, MPEG-4, H.264, as well as other foreign standards. In recent years, with the rapid advents in the mobile Internet industry, the demand for audio and video applications has been increasing, e.g., tele-medicine, in-vehicle video, live video, tele-education, and industry monitoring [7–10]. To promote the standardization of audio and video applications in the mobile Internet era, China's audio video coding standard working group has launched the coding standard for mobile video, i.e., audio video coding standard for mobile (AVS-M). AVS-M is the seventh part of standard series *advanced audio and video coding for information technology*, is the second-generation source coding standard with independent intellectual property, and is also the foundation of the digital audio and video industry [11,12]. As a hybrid coding one based on spatial and temporal prediction, spatial-domain transformation, and statistical entropy coding, AVS-M's coding framework thus consists of these three modules. Conversely, its decoding framework includes entropy decoding, anti-quantization, inverse transformation, and loop filtering modules [13]. In particular, Cuozzo et al. in [14] proposed to monitor the production cycle of industrial goods with LoRa, by designing the network architecture and medium access control protocol, and Magrin et al. in [15] evaluated the performance of a LoRa-based network in which multiple IIoT devices communicate with a centralized unit. In particular, the federated learning is the potential for intelligent IIoT by coordinating several distributed units to train in the edge, thereby facilitating the protecting of security in information [16]. Since only the decoding in the field is analyzed, not considering the explicit transmission of encoded video (from distributed node to the central unit) and privacy issues (among multiple nodes), either LoRa or federated learning is not involved in this study.

In particular, the AVS-M encoded video comprises several network abstraction layer units (NALUs) in sequence. Therefore, the AVS-M decoder has to sequentially analyze the header information of NALUs, anti-quantize its residual information, execute reverse discrete cosine transform (DCT), perform both the intra-frame prediction and the inter-frame motion compensation at the macro-block level, and finally filter the information to recover the image [17]. Since the AVS-M data are encoded via variable-length encoding, the size of each NALU is unfixed, making the boundary between NALUs obscure. Therefore, existing decoding methods for AVS-M encoded video to advocate for sequentially analyzing NALUs. That is, starting from the initial position, the contents in the network abstraction layer (NAL) header are first read in sequence, and then some certain method is used to process the NAL loads in the NALU following instructions in the NAL header, until all NALUs are traversed. More precisely, AVS-M has to sequentially analyze the contents in NALUs and then merge the analytical results. Nevertheless, when the large data spans multiple NALUs (e.g., in the industry monitoring), this type of NALU sequential analyzing and decoding method could not fully utilize the computing resources, incurring a large processing delay and failing to make a real-time operational decision [18], and thus not adapt to rapidly changing industrial field situations. As such, the simultaneous speculative parallelization has been utilized to speed up the decoding efficiency [19–21]. In particular, Lee et al. in [21] have proposed simultaneous and speculative backend execution, to offset the performance loss by speeding up the execution in the thread migration, and meanwhile to sustain the low energy overhead.

For low-latency surveillance and monitoring services, a timely video processing and decoding method is a prerequisite. Nevertheless, traditional serial decoding methods are limited by the video coding order, and cannot decode the video data in parallel, resulting in low decoding efficiency and overlarge processing delay. Thus, boosted by recent advents in both edge computing and IIoT applications, and to track the industrial environment, we try to introduce the speculative parallelization technique [18], rather than use the traditional NALU sequential analyzing method, to save the surveillance video processing

and decoding delay in the industrial field. In particular, with advances in mobile chips and compelling techniques, the speculative parallel decoding can be promoted to some extent, thereby significantly improving the performance of AVS-M in terms of decoding efficiency. The contribution of this work can be summarized as follows:

- An edge intelligence-assisted and asymmetrical IIoT AVS-M decoding framework is designed. Not only cameras in terrestrial networks, but also UAVs in the SAGIN, are used to monitor the industrial field, and then the video data are locally processed and decoded at the edge server asymmetrically in parallel, without transmitted to the remote central cloud for analyses.
- A speculative parallelization-based video decoding method is proposed. That is, the data are first divided speculatively following the starting code, then one verification method is designed to improve the dividing accuracy, and segments are finally stitched together to reconstitute the video.
- The deployment of proposed method in the hardware platform Cortex-A15 is discussed, including the linking and compiling of the code, the structure-based code-level optimization, as well as the the memory space allocation strategy to speed up the running.
- Both function and performance environments are conducted for traditional serial and proposed parallel methods. Proposed method does improve the decoding efficiency in the industrial edge, in terms of running time, speed-up ratio, and parallel efficiency, particularly in the high-bit rate and high-resolution video.

The remaining is organized as follows. The preliminary is listed in Section 2, the proposed method is presented in Section 3, the implementation of the method to the platform is discussed in Section 4, and experiment results are shown in Section 5, respectively. Section 6 concludes this work.

2. Preliminary

2.1. Edge Computing and IIoT

IIoT is a collection of various devices (e.g., sensors, cameras, UAVs, and meters), working together to gather, monitor, and analyze data from an industrial environment. Such data analysis facilitates the visibility of operators and maintenance on facilities, also enhancing efficiencies, save costs, and improve security. In particular, in the oil and gas industry, the drilling scenario, including drilling rigs, construction yards, vessels, pipelines, production wells, etc., have been taken as one typical use case in IIoT [22,23]. Note that, in the industrial monitoring, to facilitate the fast detection and decoding of the surveillance video, the video data are no longer delivered to the remote cloud computing, but processed and decoded locally, thereby necessitating edge computing in the field. Many recent works have emerged in edge computing or edge intelligence in IIoTs, e.g., in terrestrial cellular networks [23], either using prevailing machine learning (neural networks) or statistics.

Further, affected by the encoded data structure, the decoding is performed serially and thus with low efficiency, and even the hardware investment cannot improve the efficiency. Thus, to improve the video decoding efficiency in edge servers, the parallel decoding method is preferable to the serial one, which could compete with finite resources to speed up the decoding, without degrading the decoding correctness.

For illustration, we present an edge intelligence-assisted drilling field in the oil and gas industry in Figure 1, where three edge servers and one cloud server coexist. In particular, to promote video data analysis, both cameras and UAVs (in SAGIN) are used to record the maintenance and operation of drilling rigs, mud pumps, or pipelines. Further, to lower the transmission and propagation delay and enable a fast adaptation to the industrial environment, captured video data are delivered directly to the edge server, rather than to the remote cloud computing server for decoding. Although the processing delay could be further reduced in the cloud server, it suffices to dispose not too large a volume of data in the edge server, aided by the built-in speculative parallelization.

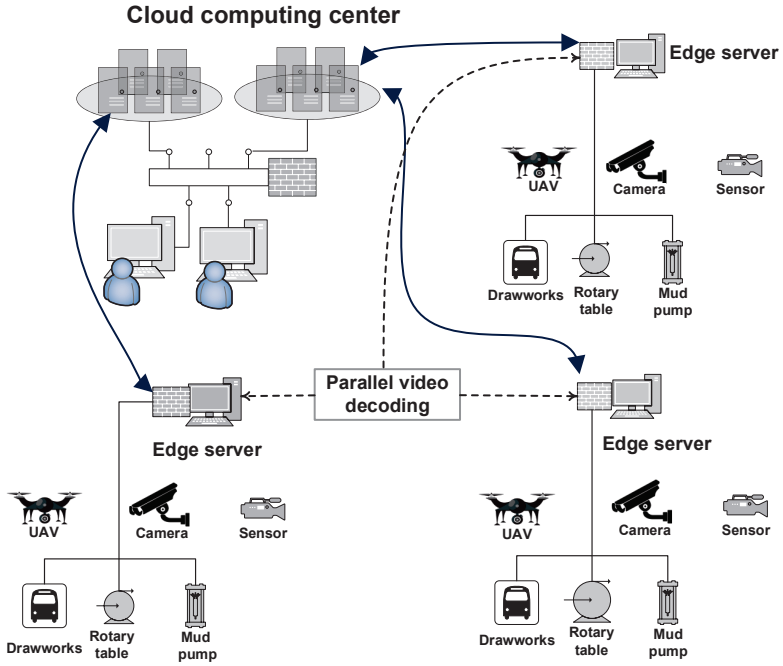

Figure 1. Edge intelligence-assisted drilling scenario.

2.2. Decoding in AVS-M

One typical decoding method for AVS-M encoded video is illustrated in Figure 2. The AVS-M's decoding structure consists of two layers, i.e, the video coding layer (VCL), representing the video data sequence after compression and encoding, and the network abstraction layer (NAL), responsible for formatting the data and providing headers to ensure the successful transmission of data over various channels and storage media. In particular, the data from VCL is first packeted at the NAL layer unit, and then packed by the network protocol to constitute the NAL unit. This type of packetization of NALUs is shown in Figure 3, where each NALU is byte-aligned, and consists of a 3-byte starting code (0x000001), a 1-byte NAL header, and a variable-length NAL load.

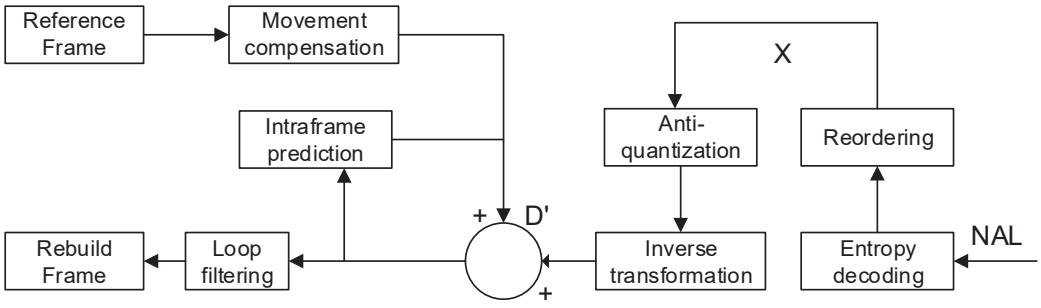

Figure 2. Workflow of traditional decoding for AVS-M coded video.

As shown in Figure 3, the starting code marks the beginning of the NALU, the raw byte sequence payload (RBSP) is stored in the ending NAL load, and the NAL header between them specifies the property and type of RBSP and other information. In particular, the NAL header contains three parts: (1) prohibited bit (1 bit), indicating whether or not there exists a bit error code in the current NALU (0 means that there is no bit error code, 1 vice versa); (2) NAL reference number (2 bits), indicating the property of load hosted in the current NALU (0 means that the load is non-reference frame data, and $\neq$0 means that the load is reference frame data; and (3) NALU type (5 bits), indicating the load type in the current NALU, wherein values 1–6 are valid, and other values are reserved or not defined.

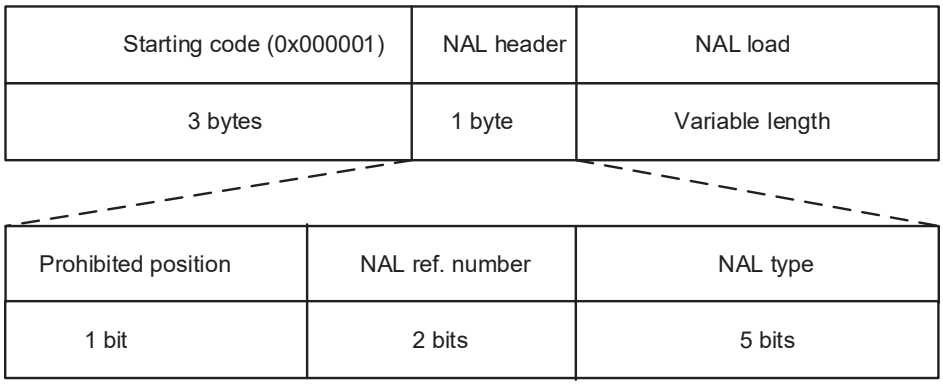

Figure 3. NALU structure.

Since the AVS-M standard uses variable-length coding to encode video data, the size of each NALU is uncertain, and the boundary between NALUs in the video is undetermined. Note that, the traditional decoding method for AVS-M encoded video has to sequentially analyze NALUs. First, the initial position of the video is accessed. Then, after the starting code, the contents in the succeeding NAL header are read. Next, following the instruction in the NAL header, one certain method is used to process the NAL load hosted in the NALU. Finally, the NAL load is successfully processed, and the above procedure repeats until all NALUs are traversed. Yet, in the case of large data volume, particularly in the industry monitoring domain in Figure 1, the traditional AVS-M decoding method (sequentially analyzing NALUs) could not fully utilize the computing resources, thus incurring a large processing delay. Further, the computing resources in edge servers are essentially limited compared to those in cloud centres, asking for a more efficient decoding method residing in edge servers.

2.3. Speculative Parallelization

To make the boundary between NALUs obvious and speed up the decoding rate in edge servers in IIoT, we intend to use the speculative parallelization technique to enable parallel decoding. As a mature parallel accelerating tool on multi-core platforms, thread-level speculation (TLS) has emerged to empower thread-level speculative parallelization with domain knowledge [18]. By tolerating inter-unit dependencies, TLS allows threads or computations (that were thought to be non-parallel) to proceed in the speculative parallelization, thereby gaining the parallelization when speculation succeeds. Figure 4 illustrates the execution of TLS-based parallelization.

As shown in Figure 4, since threads T1 to T2 have data dependencies, they have to be synchronized, and T2 can only be executed after T1 in the serial execution (Figure 4a). In the TLS case (Figure 4b), if the value prediction succeeds, then the parallel execution is performed, and the program running time is significantly reduced compared to the traditional serial one. Even if the prediction fails, the parallel program would roll back and be executed again, taking the time equal to the serial execution time plus some parallel overhead (Figure 4c). Thus, TLS is utilized to parallelize the irregular algorithms with complex data dependencies on multi-core platforms, e.g., the ones including many do-while loops [12]. Furthermore, TLS can improve the success rate of speculative execution by using domain knowledge to guide thread division, thus obtaining a higher speed-up ratio for programs [24].

Note that existing speculative parallelization techniques require high flexibility among concurrent units and consume a lot of extra overhead to maintain the inter-instructional biased order relations of algorithm semantics, and thus are mainly used to parallelize algorithms with a large number of complex control dependencies inside. The challenge in this work, however, is that there are dependencies between the input data that prevent parallelization of the decoding, which in turn manifests itself in the form of data flow and control flow during the data decoding. Therefore, the speculative parallelization cannot

directly participate in parallelizing the decoding itself. Instead, the whole algorithm has to be divided into four parts: data decomposition, result verification, decoding, and result merging. Then only the first three parts can be parallelized, making the proposed method differ from other existing speculative parallelization method.

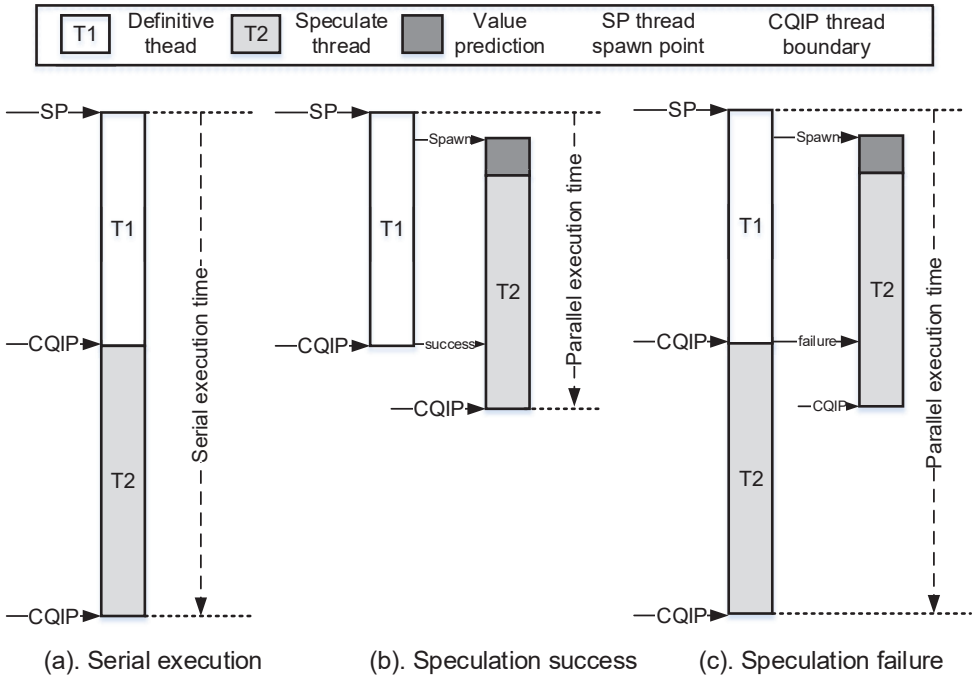

Figure 4. Illustration of speculative parallelization model.

3. Speculative Parallelization-Based Decoding in Industrial Edge

3.1. Internal Dependency Dnalysis

To design the speculative parallel decoding method, we first need to analyze the traditional decoding for AVS-M encoded video, finding the key factor affecting the decoding. Then, the speculative parallelization method is designed following this factor. As described in Section 2.2, AVS-M uses variable-length encoding to encode data, and the decoding method is also performed by sequentially analyzing NALU. The flow can be summarized as follows: after obtaining the video data, the traditional decoding method would read the starting code of the first NALU from the beginning of the video, then read the contents in the NAL header, and further process the NAL load hosted in that NALU following the instruction in the NAL header. Repeat the procedure until all NALUs are traversed. Figure 5 represents the general process of the AVS-oriented decoding method and its corresponding dependency graph.

As shown in Figure 5a, one NALU is decoded in each loop. When the NALU is decoded, variable *pos* (representing the position information) is updated and will point to the starting position of the next NALU. *out_buf* stores the decoded data, and *stcd* and *err* indicate whether or not a starting code is found and whether or not an error occurs. If there are no errors, then the result is written to the output file *out_stream* after all NALUs are decoded. In Figure 5b, each loop includes three types of dependency to prevent the concurrent execution of multiple loops, i.e, the control dependency by the variable *stcd*, the data dependency by the variable *pos*, and the data dependency by the variable *out_stream*. In the case of large data volume, the input data contains a large number of NALUs, and the embedded dependencies could seriously degrade the performance. Therefore, it is a prerequisite to integrate the AVS-M video with speculative parallelization to exactly find the starting code, and then slice the data (before decoding) following the starting code to enable the parallel decoding.

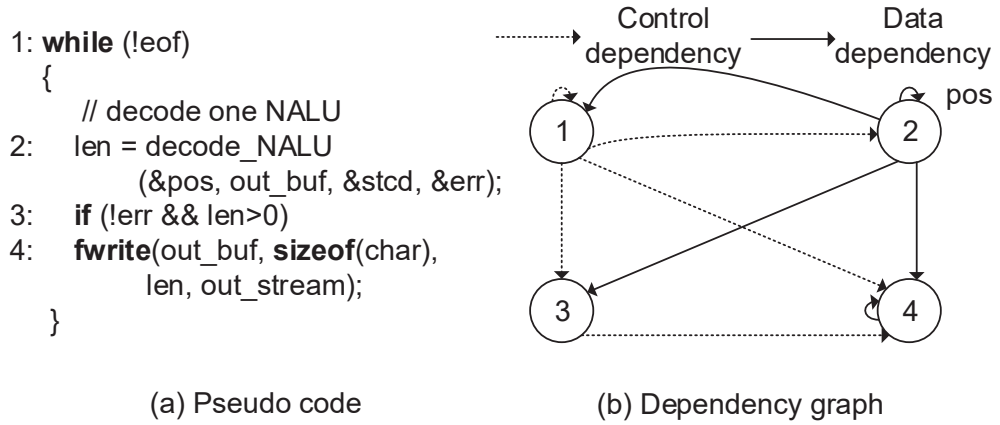

Figure 5. Pseudo code of dependency graph of traditional AVS-M method.

3.2. Speculative Data Division

As seen in Figure 5, one simple way to divide the video is using the starting code as the identifier to slice the data. However, despite the starting code's single structure and simple content, the video encoding sequence might still coincide with the content of the starting code; if the division is just simply performed using 0x000001, then the division results would go wrong and destroy the file's integrity. Therefore, we propose to speculatively divide the data following the content in the starting code. Further, based on the speculative division, one dividing verification method is designed to improve the accuracy. Finally, division results are sent to the embedded device to improve the AVS-M video decoding efficiency. More precisely, as shown in Figure 6, the starting code 0x000001 is used as a *speculation pointer* to scan the video data byte by byte, and all data segments with the same starting code are recorded, which would be taken as the possible speculative division points.

Although the division in Figure 6 can be performed in serial, the parallel dividing can also be implemented [25]. In the possible parallel dividing, apart from parallel compilation techniques, a sufficiently large memory is required as hardware. In particular, the data to be divided must reside in the memory and be uniformly executed via the bus system [26]. More precisely, the main thread is in the pending state and instructs the parallel task by monitoring the memory state. After the speculative division, the main thread resumes from the pending state and proceeds to verify the division results.

3.3. Division Result Validation

In Section 3.1, the starting code in AVS-M is taken as the criteria to divide the input data, possibly incurring the wrong division. If the speculation is used to slice the input data and verify the division results, then the input data integrity can be preserved, and the dependency between different NALUs can be broken. As such, the video data can be divided into a collection of NALUs decoded independently, thus supporting for decoding video data in parallel.

Note that the position obtained by searching for the header identifier is not necessarily the starting code of NALU since the concatenating segments of binary strings in the encoded data are possibly the same as the starting code of NALU. Therefore, it is necessary to propose one verification method based on the NALU structure. If an error occurs and is not detected via the validation scheme, then all the succeeding data might go incorrect from the point where the error occurs, thus incurring more cost. Thus, we use the NAL header to decide whether or not the starting code-based division results are correct.

More precisely, we further use both the *forbidden bit check* and the *NAL cell type check* to check the legitimacy of the NAL header. The forbidden bit check decides whether or not the forbidden bit in the NAL header is 0; if that in the NAL header of one segment is not 0, then its division result is incorrect. Further, the NAL cell type check scans the last

5 bits of the NAL header: when the NAL cell type value lies in $[1, 6]$, the division is correct. In particular, although the header legitimacy can be ensured by either forbidden bit or NAL cell type checking, the data integrity could only be ensured by both types together. As shown in Table 1, only the load data ranging from 1 to 6 is specified.

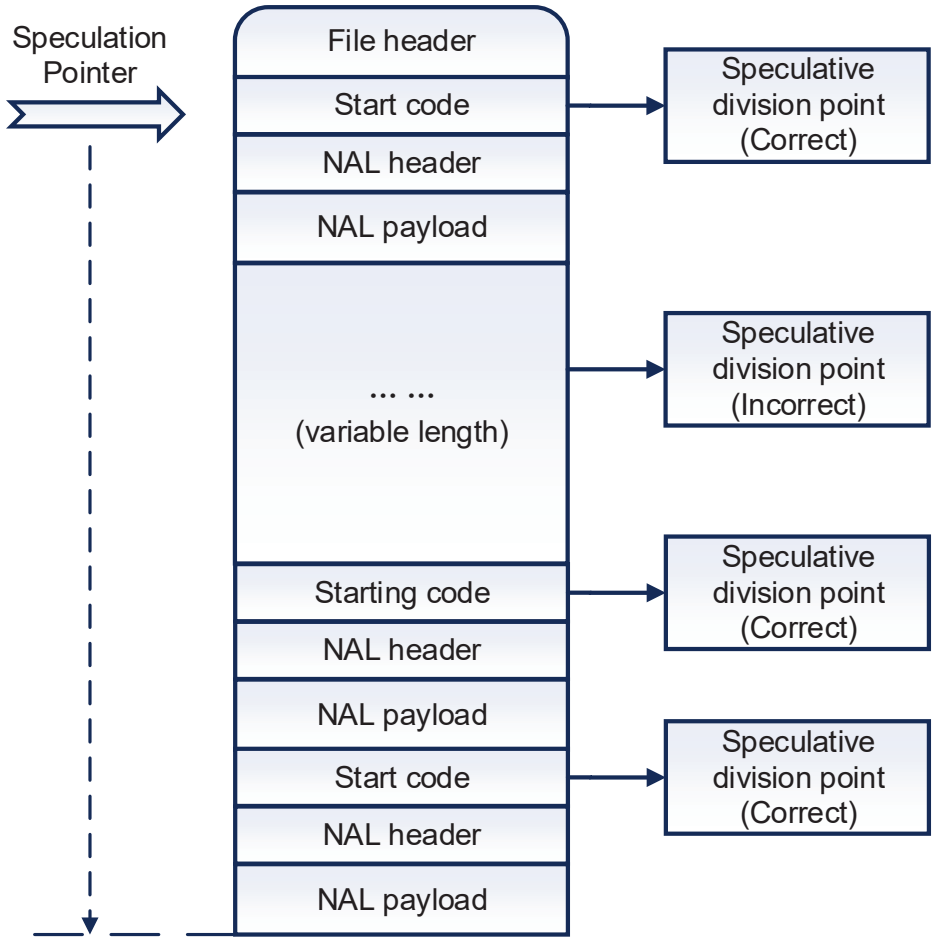

Figure 6. Speculative data division.

Based on the legitimacy check criteria, we would use OpenMP (i.e., a portable and scalable model to develop the shared-memory parallelization for multi-platforms), together with the ARM-Anrdoid-NDK compiler, to enable the parallel optimization of legitimacy check against the NAL header [24]. Multiple threads are opened to check in parallel via the high-level description provided by OpenMP, together with the self-contained *pragma* instruction. Figure 7 illustrates the workflow of parallel verification on inferred results. As shown in Figure 7, three sets of speculative division results are obtained from the starting code, the verification is as follows. First, in T1, the forbidden bit in the third data block is 1, not meeting the standard NALU format, and failing in the check. Furthermore, the remaining two groups of data could pass the header check. Then, the NAL cell type check is executed on the data having passed the forbidden bit check. In T2, the NAL cell type data is 0 in the second data segment, not falling into interval $[1, 6]$, also with the wrong result. Finally, only the first speculative data segment passes all checks, implying that the first data segment is correctly divided, with the true starting code residing in it.

Table 1. NAL cell type.

NAL Cell Type Value	Description
0	Not specified
1	Striping of non-IDR images: *slice_layer_rbsp( )*
2	Striping IDR images: *slice_layer_rbsp( )*
3	Image header: *picture_header_rbsp( )*
4	Sequence parameter set: *seq_parameter_set_rbsp( )*
5	Image parameter set: *pic_parameter_set_rbsp( )*
6	Auxiliary enhancement information: *sei_rbsp( )*
7–23	Reserved bits
24–31	Not specified

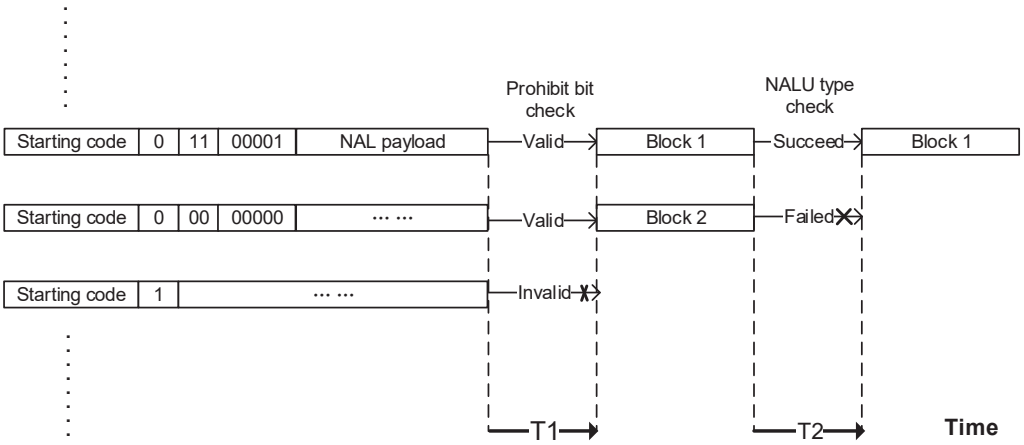

Figure 7. Parallel checking on division results.

3.4. Parallel Decoding

After receiving the speculative parallel task from the main thread, all working threads simultaneously decode the chunked data and perform the same task but with different data. FFmpeg is one of the most popular open-source multimedia tools, used extensively in video websites and commercial software (e.g., Youtube and iTunes), and is the standard encoding/decoding technique in many audio and video formats [27,28]. The flow of each working thread calling for FFMPEG decoding is illustrated in Figure 8.

As shown in Figure 8, the decoding consists of entropy decoding, anti-quantization, inverse variation, intra-frame and inter-frame mode selection, and loop filtering. First, the stream information is read by FFMPEG, and the residual data are obtained after entropy decoding, reordering, anti-quantization, and inverse variation in NALU units. Meanwhile, the intra-frame prediction or inter-frame motion compensation is performed. In particular, the NAL load in the data segment has to be extracted, and then the information in the NAL header is analyzed, followed by the processing of extracted NAL load. If the NAL unit type value of NALU is 1, then its NAL load is justified to be a strip of non-instantaneous decoder refresh (IDR) images, and FFmpeg for the strip of non-IDR images is called to process the NAL load. Finally, the residual and predicted data are fused, which further input the loop filter to reconstruct the image.

After the parallel processing, all working threads would immediately terminate the task and notify the main thread, which then resumes from the pending state and collects the results from all working threads. Next, the main thread would destroy all working threads and restore the computing resources. More precisely, following the dividing order, all working threads' results are stitched together to constitute the decoding result.

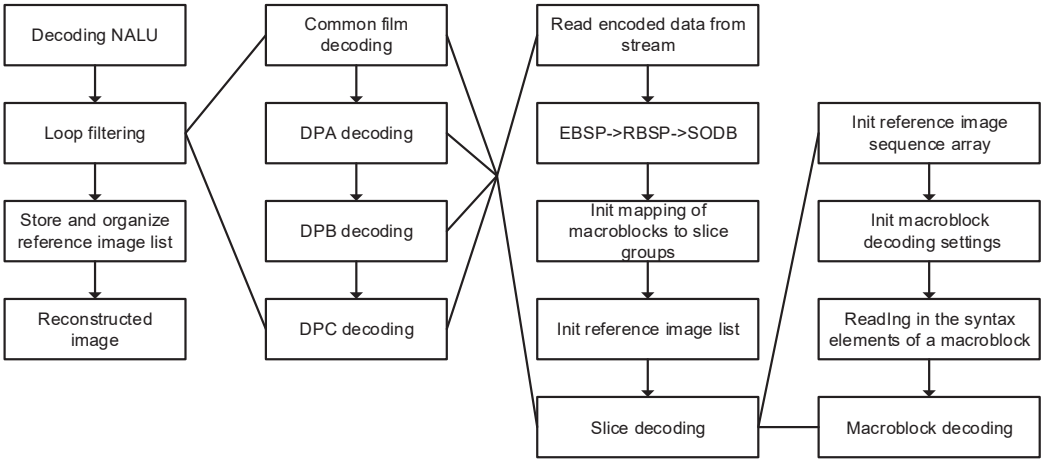

Figure 8. Parallel decoding of AVS-M video.

3.5. Error Handling

When the decoding error occurs, i.e., the main thread receives an error decoding message from the working thread, it would switch from the pending to the working state. Then, the error decoding message is recorded, the preceding decoding results are saved, the error decoding results are discarded, and the incorrect boundary information is also removed from the data segment boundary list. Finally, the residual data are rolled back and processed in parallel. Error handling could further bring robustness in decoding. In particular, the error-handling part is mainly used to improve the overall robustness. Since the multiple verification process for the division results is established before decoding data in parallel, the division results having passed such verification must be correct. The potential misprediction is due to the partial error code that might be generated during the transmission of the binary code, and the probability of this occurrence in the short-distance transmission is very low, which barely happens enough to degrade the algorithm's performance. Finally, frequent mispredictions do affect the performance of parallel programs theoretically; nevertheless, the error handling part only handles the wrong blocks, while the other correctly predicted ones would still be decoded normally.

4. Algorithm Implementation

4.1. Hardware

We select ARM's Cortex-A15 as the implementation platform for the proposed method. Cortex-A15 uses the ARMv7a instruction set, allowing the CPU to increase its memory addressing to 1 TB, no longer restricted by the 4GB memory addressing space of 32-bit processors. ARMv7a's NENO SIMD instruction set can also operate the 128-bit register and support virtualization. In addition, the CPU uses a 128-bit AX14 bus, with its physical addressing space upgraded to 40-bit [29,30], and the Cortex-A15's certificate pipeline depth is extended to 10–12 levels, slightly boosting the frequency. Finally, the Cortex-A15 architecture is powered by the latest Mali-T622 graphics processor, which improves the performance by 50% than its predecessor T600 series, and supports OpenGL ES 3.0 for better 3D visualization. Furthermore, Mali-V600 video accelerator residing in Cortex-A15 is tailored for video acceleration, including one 8-core engine to handle video up to 4K 120 frames per second (FPS). Figure 9 illustrates the architecture of the Mali-T604 graphics processor in Cortex-A15.

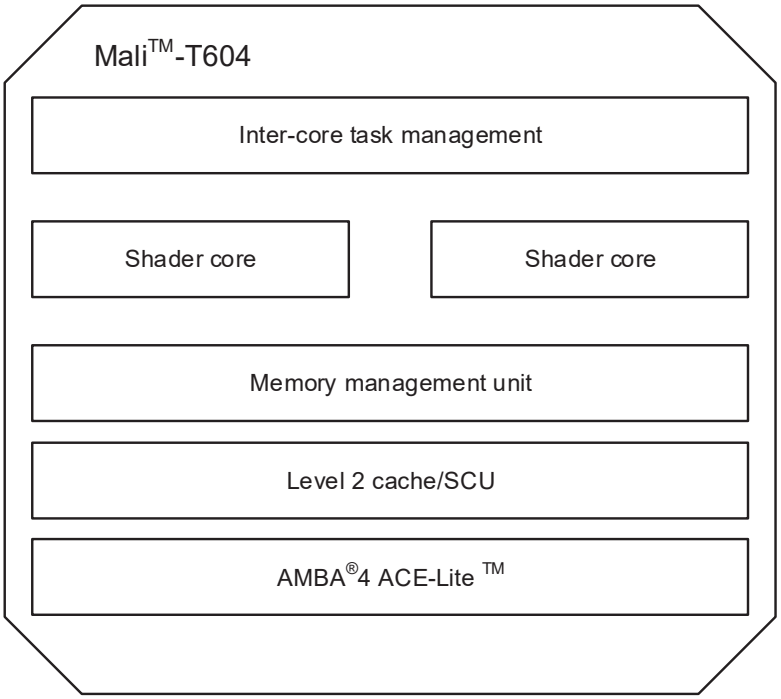

Figure 9. Architecture of Mali-T604 graphics processor.

More precisely, to support the multi-threaded parallelization, Cortex-A15 has two decoding units and focuses on its branch prediction capabilities. ARM has transferred the branch predicting unit from Cortex-A53 to Cortex-A15, and the latter is equipped with two units (naming and sending). Apart from the ability to execute in chaotic, Cortex-A15 also has abundant instruction launch ports and computing resources. Via the multi-level branch caching, Cortex-A15's branch prediction capability is more powerful than that of its predecessor, enabling a more efficient A15 pipeline execution. Further, by embedding a VFPv4 floating-point unit, Cortex-A15 can run both fused-multiply-add (FMA) and hardware division instructions, while Cortex-A9's floating-point capacity is only half of that of Cortex-A15 [31].

4.2. Algorithm Deployment and Optimization

To deploy the proposed method on Cortex-A15, we have to compile and link the code via the integrated development tool *codewarrior for ARM developer suite* [32]. Then, the open-source library FFmpeg would generate four executables (including FFmpeg for transcoding and pushing media streaming, FFplay for playing media files, FFprobe for obtaining media file information, and Fserver as a simple streaming media server), and eight static libraries (i.e., modules of FFmpeg) after compilation. In the algorithm implementation, it is a prerequisite to use AVCodec (i.e., the library module among eight modules), to determine the type of NALU and select the method to handle the NAL load. Furthermore, code-level optimization is required to improve the parallel algorithm on embedded devices. Specifically, the input data is stored in structures, putting the data and sub-structures in one large structure [33]. As such, the proposed method can dynamically allocate memory to meet different decoding requirements, save memory space and avoid system resource scarcity due to memory leakage. Each encoded video is indexed by the outer structure, but the nested structure has at most two levels, thus saving the time for addressing.

Finally, it is necessary to optimize the memory allocation strategy in parallel. In the intra-frame prediction, local variables are used in the left and upper segments, to temporarily store prediction results. Since local variables are placed in the stack, their space would be released and reused by the next function once the current one ends. Further, continuously reusing the same memory space can improve the caching hit rate, thus speeding up the

algorithm. More precisely, the request number is lowered to save space. Further, it is undesired to assign values for each variable (array), but use batch assignment statements (e.g., MEMCPY and MEMSET) to duplicate data. Thus, the above optimization could greatly not only simplify the speculative parallelization-based AVS-M decoding method, but also improve the performance of embedded devices in the Cortex-A15 architecture. The overall flowchart of the proposed method is illustrated in Figure 10.

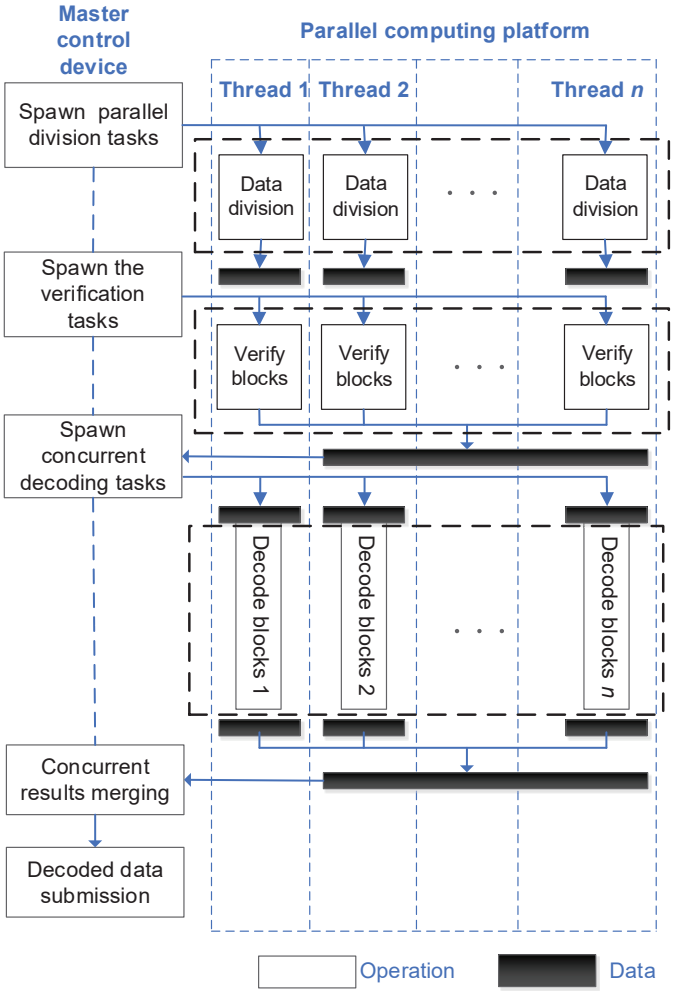

Figure 10. Flowchart of proposed parallel video decoding method.

5. Experiments Results and Analyses

5.1. Experiment Configuration

To verify the performance of the speculative parallelization-based AVS-M video decoding method, experiments are conducted as follows. First, Nvidia Tegra 4 mobile chip built on Cortex-A15 architecture is taken as the experiment platform. In particular, Nvidia Tegra 4 packs 72 customized GeForce GPU cores and a quad-core ARM Cortex-A15 CPU, reaching the maximum frequency of 1.9 GHz [34], and can dynamically activate either the single core or all four cores simultaneously through its built-in symmetric multi-processing architecture, in line with the resource consumption of current task. In this experiment, all four cores are activated by default.

Further, the experiment data are obtained from the drilling machining monitoring data set collected by CNPC national engineering research center for oil & gas drilling equipment Co., Ltd. Four stream segments are selected as the experiment data to verify the proposed method's performance in terms of different bit rates and resolutions. In particular, the speculative parallelization-based decoding method is compared with the traditional

serial decoding one. We first conduct function experiments on both methods, to exhibit that the proposed method can achieve a comparable effect as the serial one, and the introduction of the speculative parallelization does not degrade the decoding accuracy. Further, the proposed method's efficiency is verified, to prove that the speculative parallelization can improve the decoding efficiency for AVS-M encoded video, in terms of the execution time, decoding efficiency, and speed-up ratio.

5.2. Accuracy Validation

Before verifying the acceleration performance of the proposed method, the basic decoding accuracy has to be validated, i.e., the parallel method can achieve comparable accuracy as the serial one. That is, the introduction of speculative parallelization would not destroy the decoding accuracy. In particular, peak signal-to-noise ratio (PSNR), i.e., the ratio of maximum possible power of the signal to noise power, is taken as the frame evaluation metric [35,36]. Since signals are extended in a large dynamic range, PSNR is taken in the logarithmic decibel units. First, before obtaining the PSNR, the mean square error (MSE) between two frames is expressed as follows:

$$\mathrm{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2, \tag{1}$$

where I and K denote the $m \times n$-sized decoded and original frames, respectively.

Then, to evaluate the difference between decoded and original frames, the MSE-based PSNR is expressed as

$$\mathrm{PSNR} = 10 \log_{10} \left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}} \right) = 20 \log_{10} \left(\frac{\mathrm{MAX}}{\sqrt{\mathrm{MSE}}} \right), \tag{2}$$

where MAX is the possible maximum pixel number for the frame. For instance, if an 8-bit binary represents each pixel, then MAX equals 255. In particular, the MAX value for floating-point data equals 1. Assume the pixel value is represented by a B-bit binary, and we have $\mathrm{MAX} = 2^B - 1$. More precisely, PSNR above 40 dB, at 30–40 dB, at 20–30 dB, and below 20 dB, respectively, indicate superior (close to the original frame), good (detectable but acceptable distortion), poor, and unacceptable quality.

Finally, the accuracy validation results are shown in Table 2, with four data sets. It is evident that the PSNR of both methods are identical, indicating that the optimization does not affect the decoding. Thus, the proposed method is undistorted as the serial one, keeping the decoded frame quality unchanged. That is because the speculative parallelization only changes the decoding order, but does not optimize the decoding itself. Further, the verification also ensures the correctness of speculative results, thus ensuring the accuracy of speculative decoding results. Thus, the speculative parallelization would not degrade the decoding quality of AVS-M encoded video.

Table 2. Accuracy validation results.

Test Data	Format	Resolution	PSNR (Serial)	PSNR (Parallel)
Stream 1	AVI	720 × 480	44.02	44.02
Stream 2	MP4	1280 × 720	42.03	42.03
Stream 3	WebM	1920 × 1080	43.59	43.59
Stream 4	MKV	2560 × 1440	45.01	45.01

5.3. Acceleration Performance Validation

Based on the above function experiments, the following performance experiments are conducted. First, the decoding efficiency of the test data on different resolutions and bit rates is recorded on the experiment platform (composed of Nvidia Tegra 4 and

using the traditional serial decoding method). Then, also on this experiment environment, the efficiency of the speculative parallelization decoding method is tested by decoding data with different resolutions and bit rates. Note that, Nvidia's virtual symmetric multi-processing mechanism is turned off throughout the experiment, allowing both serial and parallel methods to run on a multi-core and multi-threaded CPU. Further, the results of the performance experiment are shown in Tables 3 and 4, which show the decoding efficiency of serial and parallel methods, respectively. From both Tables 3 and 4, it is evident that for any bit rate and resolution in AVS-M encoded video, the decoding efficiency of the parallel method in a multi-core environment is significantly improved over the serial one. In statistics, the average decoding efficiency of the parallel method is 7.9, 8.1, 9.2, and 11.5 times higher than that of serial one, respectively, at 480p, 720p, 1080p, and 2K resolutions, thus verifying the superior performance of the proposed method. More precisely, it is more evident in the higher video quality, and proposed method is favored to process high bit rate and high-resolution videos.

Table 3. Decoding efficiency of serial method.

Bit Rate (Mbps)	Stream 1 (720 × 480)	Stream 2 (1280 × 720)	Stream 3 (1920 × 1080)	Stream 4 (2560 × 1440)
1	20.54	9.42	4.67	2.82
2	18.71	8.65	4.08	2.16
4	18.01	7.59	3.81	1.83
6	16.92	7.30	3.54	1.17
8	15.83	6.81	3.39	0.94

Table 4. Decoding efficiency of parallel method.

Bit Rate (Mbps)	Stream 1 (720 × 480)	Stream 2 (1280 × 720)	Stream 3 (1920 × 1080)	Stream 4 (2560 × 1440)
1	155.32	74.47	40.33	21.83
2	147.58	72.19	37.27	20.09
4	140.12	70.93	35.31	18.38
6	135.37	68.24	33.76	16.69
8	128.81	62.05	32.51	15.51

Further, to show the performance improvement of speculative parallelization in more depth, the running time, as well as a speed-up ratio are also evaluated, and the latter is defined as

$$S_p = \frac{T_l}{T_p}, \tag{3}$$

where T_l and T_p separately denote the running time for serial and parallel decoding. With (3), we can explain how much faster the parallel decoding is than the serial one. More precisely, Figure 11 shows the variation of speed-up ratio at different resolutions and bit rates. As shown in Figure 11, the parallel method has a significant speed-up ratio when decoding video for different bit rates and resolutions in a multi-core environment. In particular, for high-definition and high-bit-rate video, the speed-up ratio can reach 11.22. That is, although the speculative parallelization and verification may incur some overhead, dividing the video data (which should have been serially decoded from beginning to end) can bring forth a set of tasks decoded simultaneously. Once all tasks are decoded, the results are assembled sequentially in line with the division order. More precisely, with the growth of bit rate and resolution, the computational complexity also increases, incurring a widened gap in decoding efficiency between serial and parallel methods. On one hand, it shows that the speed-up ratio almost increases linearly with the growth of bit rate (Mb/s), since the serial decoding method's running time enlarges linearly but the parallel one could remain the running time due to its simultaneous decoding principle; on the other hand, with the

increment of resolutions, the larger resolution would incur more pixels, thereby bringing the linearly increased speed-up ratio. Thus, the speculative parallelization better fits the high-quality videos.

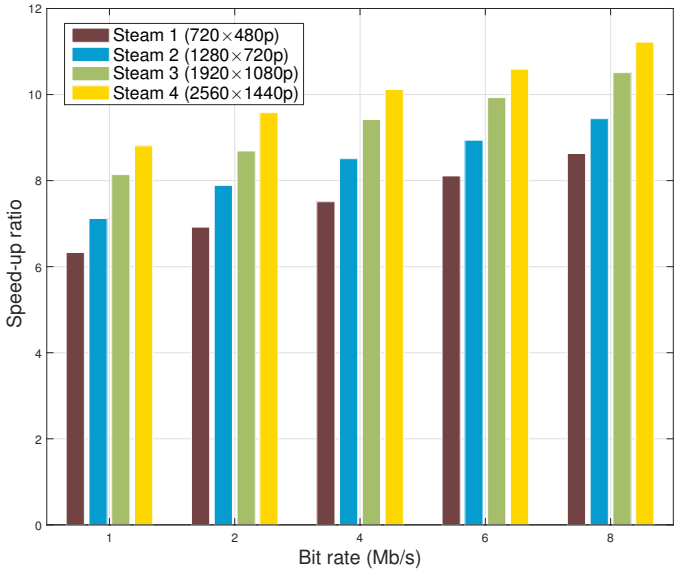

Figure 11. Speed-up ratio of proposed method versus bit rate.

Next, on the basis of the speed-up ratio, we further introduce the *parallel efficiency* metric to exhibit the performance in a multi-core system-on-a-chip (SOC) environment in more depth, which is calculated as

$$E = \frac{S_p}{N}, \tag{4}$$

where S_p and N separately represent the speed-up ratio and the number of SOC cores involved, respectively.

With (4), the decoding efficiency of the proposed parallel method under different hardware conditions is shown in Figures 12–16, i.e., efficiency versus core number. Since Nvidia Tegra 4 uses a 4-core Cortex-A15 architecture in all figures, the x-axis indicates the core number involved in parallel when one to four SOCs are used in the decoding.

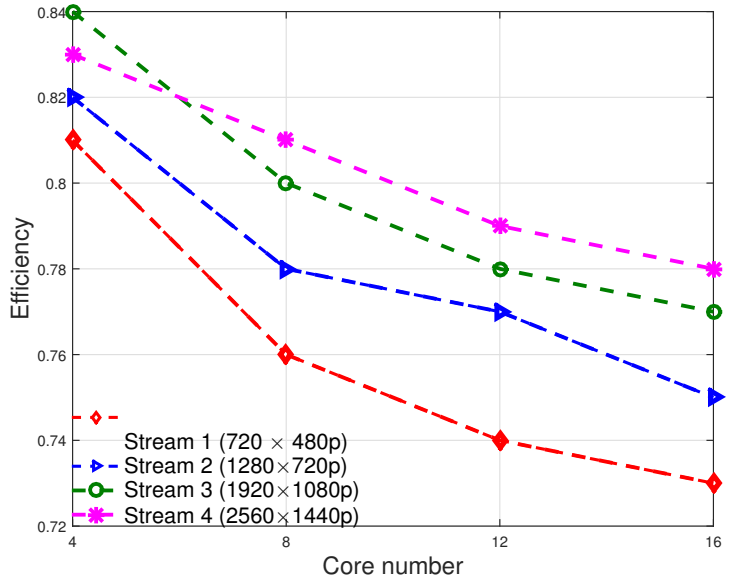

Figure 12. Efficiency of proposed method versus bit rate (1 Mbps).

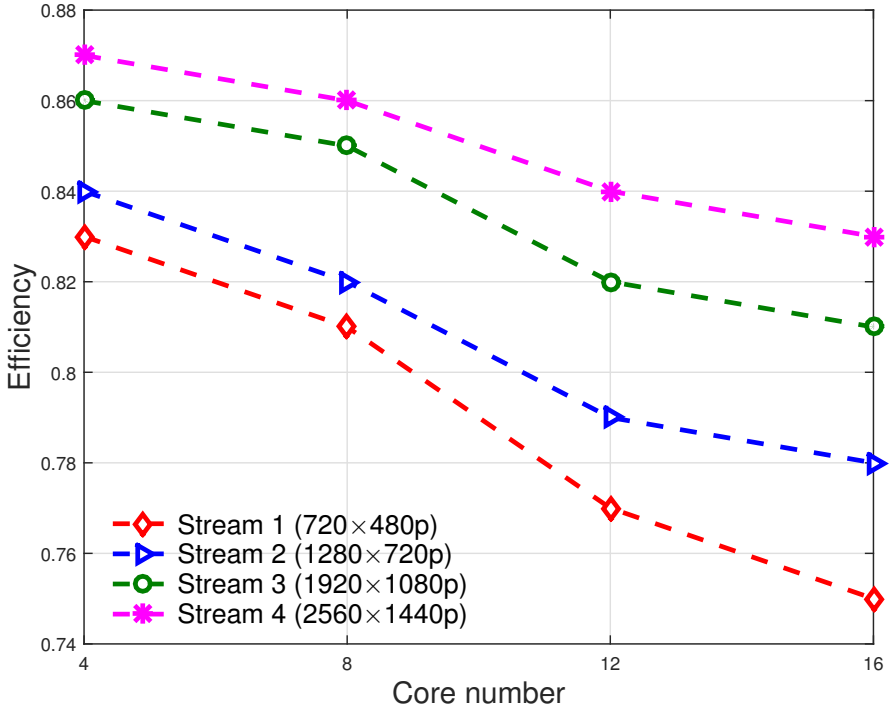

Figure 13. Efficiency of proposed method versus bit rate (2 Mbps).

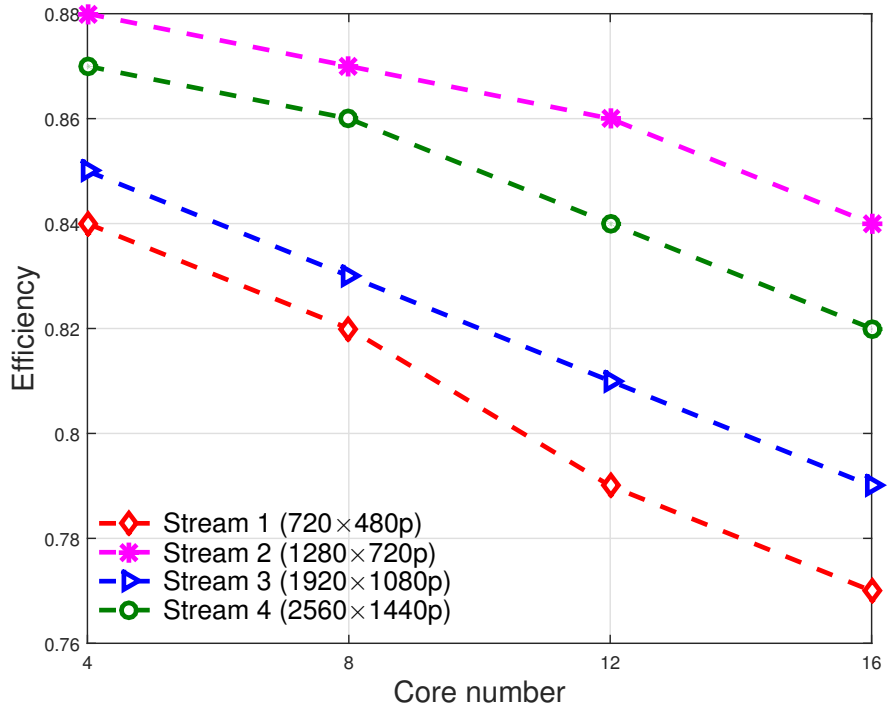

Figure 14. Efficiency of proposed method versus bit rate (4 Mbps).

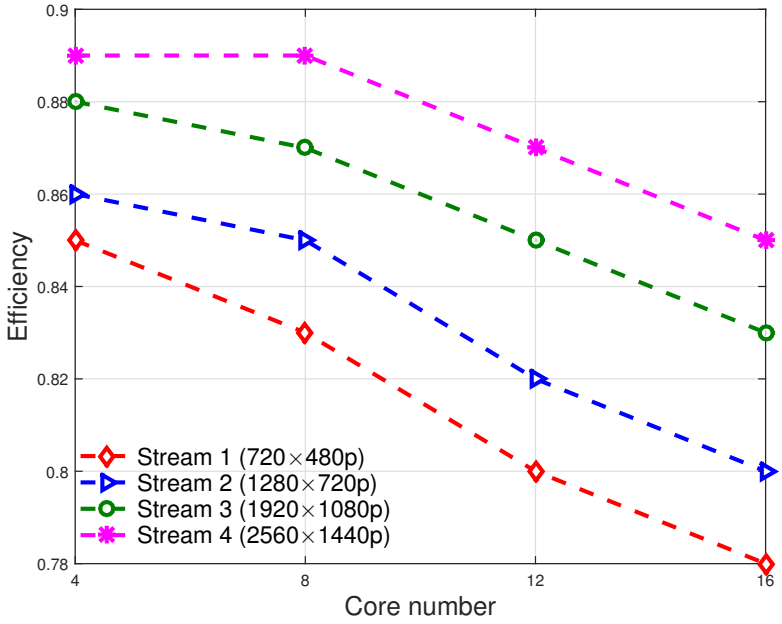

Figure 15. Efficiency of proposed method versus bit rate (6 Mbps).

As can be seen from Figures 12–16, the efficiency of the proposed decoding method always lies between 0.75 and 1, regardless of the bit rate and resolution. On one hand, the larger the bit rate, the higher the parallel efficiency of the proposed method, further showing the advantage in the real-time decoding of high-quality video. On the other hand, the efficiency decreases as the core number enlarges while keeping the bit rate constant. It is because that with the growth of parallel size, the overhead also proliferates, thus degrading the parallel efficiency. Further, the parallel overhead typically comes from the bandwidth limitation of the bus, the communication delay between cores, and the load balancing between tasks. Further, with the increment of bit rate from 1 Mbps to 8 Mbps, the efficiency across different resolutions grows by 11.2% on average. In particular, in Figures 12–15, the efficiency almost decreases linearly between 4 and 12 cores, yet with a lowered slope between 12 and 16 cores. Conversely, the efficiency decreases with a enlarged slope between 12 and 16 cores in Figure 16.

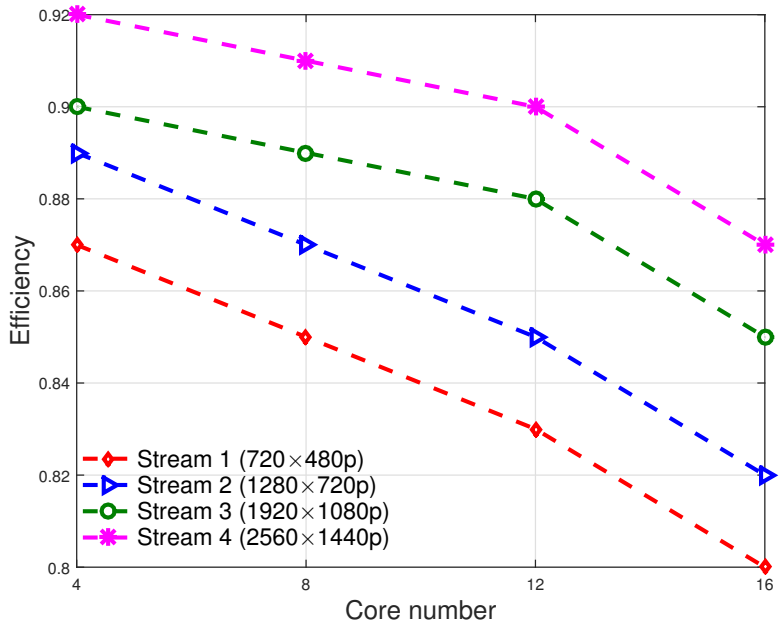

Figure 16. Efficiency of proposed method versus bit rate (8 Mbps).

6. Conclusions and Future Work

This work investigated an edge intelligence-assisted and speculative parallelization-based asymmetrical video decoding paradigm to save time and cost in industrial manufacturing. Both theoretical and environmental results have been provided to show the superiority of the proposed parallel method over serial ones in terms of the decoding efficiency in the industrial edge. Specifically, an edge computing-assisted asymmetrical video decoding framework was presented in the well-drilling field to show the importance of timeliness in field operation. First, a parallel decoding method was proposed to solve the low efficiency brought by the traditional video decoding method. The proposed method could break up the data dependency by using speculative and multi-threaded parallelization. In particular, the proposed method disrupted the serial decoding order, thereby allowing simultaneous parallel decoding. Also, a speculative verification method was designed to achieve efficiency gains while guaranteeing decoding accuracy. The experiments exhibited that the proposed parallel method is well-suited for real-time video decoding scenarios in industrial manufacturing. On Nvidia Tegra 4 embedded chips, the proposed method could achieve impressive decoding results for high-bit-rate and high-resolution video, thus showing the advantage of speculative parallelization in decoding real-time surveillance video and live streaming for industrial manufacturing. In future work, the blockchain and consensus mechanism would be embedded into the speculative parallelization to further secure the parallel decoding. Further, since the final result merging step becomes the one that most affects the overall performance and it is impossible to improve the merging efficiency by the speculative parallelization, some other approaches have to be exploited.

Author Contributions: Conceptualization, S.Y. and H.X.; methodology, S.Y. and Z.Z.; software, Z.L.; validation, Z.Z., Y.L. and H.X.; formal analysis, Z.Z.; investigation, Z.L.; resources, Z.L.; data curation, S.Y. and Z.Z.; writing—original draft preparation, H.X.; writing—review and editing, Z.L.; visualization, Z.L.; supervision, S.Y.; project administration, S.Y.; funding acquisition, Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was in part supported by the Science and Technology Project of China National Petroleum Corporation under Grant 2022ZG06-01 (Ultra Deep Intelligent Drilling Rig Development), in part by the Foundation of National Engineering Research Center for Oil & Gas Drilling Equipment under Grant BOMCO-J118-JKY003-2022 (Top Layer Design and Platform Frame of Intelligent Control Platform of Drilling Rig), and in part by the Natural Science Foundation of Shaanxi Province of China under Grant 2020JQ-647.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The authors would like to thank the editor and all reviewers for their valuable comments and efforts on this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, J.; Wang, R.; Wang, K. Service function chaining in industrial Internet of Things with edge intelligence: A natural actor-critic approach. *IEEE Trans. Industr. Inform.* **2022**, *19*, 491–502. [CrossRef]
2. Fang, C.; Yao, H.; Wang, Z.; Wu, W.; Jin, X.; Yu, F.R. A survey of mobile information-centric networking: Research issues and challenges. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2353–2371. [CrossRef]
3. Wan, S.; Xu, X.; Wang, T.; Gu, Z. An intelligent video analysis method for abnormal event detection in intelligent transportation systems. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 4487–4495. [CrossRef]
4. Zhang, Y.; Sun, L.; Cao, Q. TLP-LDPC: Three-level parallel FPGA architecture for fast prototyping of LDPC decoder using high-level synthesis. *J. Comput. Sci. Technol.* **2022**, *37*, 1290–1306. [CrossRef]
5. Choi, K.; Chen, J.; Rusanovskyy, D.; Choi, K.P.; Jang, E.S. An overview of the MPEG-5 essential video coding standard [standards in a Nutshell]. *IEEE Signal Process. Mag.* **2020**, *37*, 160–167. [CrossRef]

6. Zhou, L.; Zhou, Y.; Corso, J.J.; Socher, R.; Xiong, C. End-to-end dense video captioning with masked transformer. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 8739–8748.
7. Gao, L.; Lei, Y.; Zeng, P.; Song, J.; Wang, M.; Shen, H.T. Hierarchical representation network with auxiliary tasks for video captioning and video question answering. *IEEE Trans. Image Process.* **2021**, *31*, 202–215. [CrossRef]
8. Zhou, Y.; Tian, L.; Zhu, C.; Jin, X.; Sun, Y. Video coding optimization for virtual reality 360-degree source. *IEEE J. Sel. Top. Signal Process.* **2020**, *14*, 118–129. [CrossRef]
9. Kaye, D.B.V.; Chen, X.; Zeng, J. The co-evolution of two Chinese mobile short video apps: Parallel platformization of Douyin and TikTok. *Mob. Media Commun.* **2021**, *47*, 229–253. [CrossRef]
10. Ma, S.; Zhang, L.; Wang, S.; Jia, C.; Wang, S.; Huang, T.; Wu, F.; Gao, W. Evolution of AVS video coding standards: Twenty years of innovation and development. *Sci. China Inf. Sci.* **2022**, *65*, 1–24. [CrossRef]
11. Ji, Z.; Jiao, F.; Pang, Y.; Shao, L. Deep attentive and semantic preserving video summarization. *Neurocomputing* **2020**, *406*, 200–207. [CrossRef]
12. Li, J.; Li, B.; Lu, Y. Deep contextual video compression. In Proceedings of the NeurIPS, Montreal, QC, Canada, 11–12 December 2021; pp. 18114–18125.
13. Wang,T.; Zhang, R.; Lu, Z.; Zheng, F.; Cheng, R.; Luo, P. End-to-end dense video captioning with parallel decoding. In Proceedings of 2021 IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, 10–17 October 2021; pp. 6827–6837.
14. Cuozzo, G.; Buratti, C.; Verdone, R. A 2.4-GHz LoRa-based protocol for communication and energy harvesting on industry machines. *IEEE Internet Things. J.* **2022**, *9*, 7853–7865. [CrossRef]
15. Magrin, D.; Capuzzo, M.; Zanella, A.; Vangelista, L.; Zorzi, M. Performance analysis of LoRaWAN in industrial scenarios. *IEEE Trans. Industr. Inform.* **2021**, *17*, 6241–6250. [CrossRef]
16. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Niyato, D.; Poor, H.V. Federated learning for industrial Internet of Things in future industries. *IEEE Wirel. Commun.* **2021**, *28*, 192–199. [CrossRef]
17. Yun, H.; Yu, Y.; Yang, W.; Lee, K.; Kim, G. Pano-AVQA: Grounded audio-visual question answering on 360° videos. In Proceedings of 2021 IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, 10–17 October 2021; pp. 2011–2021.
18. Wang, Z.; Hei, X.; Ma, W.; Wang, Y.; Wang, K.; Jia, Q. Parallel anomaly detection algorithm for cybersecurity on the high-speed train control system. *Math. Biosci. Eng.* **2021**, *19*, 287–308. [CrossRef] [PubMed]
19. Kumar, S.; Singh, S.K.; Aggarwal, N.; Gupta, B.B.; Alhalabi, W.; Band, S.S. An efficient hardware supported and parallelization architecture for intelligent systems to overcome speculative overheads. *Int. J. Intell. Syst.* **2022**, *37*, 11764–11790. [CrossRef]
20. Feliu, J.; Ros, A.; Acacio, M.E.; Kaxiras, S. Speculative inter-thread store-to-load forwarding in SMT architectures. *J. Parallel Distrib. Comput.* **2023**, *173*, 94–106. [CrossRef]
21. Lee, C.; Ro, W.W. Simultaneous and speculative thread migration for improving energy efficiency of heterogeneous core architectures. *IEEE Trans. Comput.* **2018**, *67*, 498–512. [CrossRef]
22. Duggal, A.S.; Malik, P.K.; Gehlot, A.; Singh, R.; Gaba, G.S.; Masud, M.; Al-Amri, J.F. A sequential roadmap to industry 6.0: Exploring future manufacturing trends. *IET Commun.* **2022**, *16*, 521–531. [CrossRef]
23. Fang, C.; Xu, H.; Yang, Y.; Hu, Z.; Tu, S.; Ota, K.; Yang, Z.; Dong, M.; Han, Z.; Yu, F.R.; et al. Deep-reinforcement-learning-based resource allocation for content distribution in fog radio access networks. *IEEE Internet Things J.* **2022**, *9*, 16874–16883. [CrossRef]
24. Wang, Z.; Qi, J.; Ma, W.; Lv, Y.; Yang, D. An expansion planning approach for intelligent grids with speculative parallelism. *J. Circuits, Syst. Comput.* **2023**, *32*, 2350046. [CrossRef]
25. Jayatilaka, T.; Ueno, H.; Georgakoudis, G.; Park, E.; Doerfert, J. Towards compile-time-reducing compiler optimization selection via machine learning. In Proceedings of the International Conference on Parallel Processing (ICPP), Lemont, IL, USA, 9–12 August 2021; pp. 1–6.
26. Lv, Z.; Chen, D.; Singh, A.K. Big data processing on volunteer computing. *ACM Trans. Internet Technol.* **2021**, *21*, 1–20. [CrossRef]
27. Jiang, C.; Wang, Y.; Huang, Q.; Wang, Y.; Dai, Y. Intelligent video surveillance platform based on FFmpeg and Yolov5. In Proceedings of the ACM Multimedia Asia Conference, Tokyo, Japan, 13–16 December 2022; pp. 1–3.
28. Sheng, X.; Li, J.; Li, B.; Li, L.; Liu, D.; Lu, Y. Temporal context mining for learned video compression. *IEEE Trans. Multimed.* 2022, *to appear*. [CrossRef]
29. Anastasova, M.; Azarderakhsh, R.; Kermani, M.M. Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. *IEEE Trans. Circuits Syst. I. Regul. Pap.* **2021**, *68*, 4129–4141. [CrossRef]
30. Mandal, S.K.; Bhat, G.; Patil, C.A.; Doppa, J.R.; Pande, P.P.; Ogras, U.Y. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2019**, *27*, 2842–2854. [CrossRef]
31. Wu, C.; Wang, M.; Chu, X.; Wang, K.; He, L. Low-precision floating-point arithmetic for high-performance FPGA-based CNN acceleration. *ACM Trans. Reconfig. Technol. Syst.* **2021**, *15*, 1–21. [CrossRef]
32. Dörflinger, A.; Albers, M.; Kleinbeck, B.; Guan, Y.; Michalik, H.; Klink, R.; Blochwitz, C.; Nechi, A.; Berekovic, M. A comparative survey of open-source application-class RISC-V processor implementations. In Proceedings of the ACM International Conference on Computing Frontiers, Sicily, Italy, 11–13 May 2021; pp. 12–20.
33. Ford, B.W.; Qasem, A.; Tesic, J.; Zong, Z. Migrating software from x86 to ARM architecture: An instruction prediction approach. In Proceedings of the International Conference on Networking, Architecture, and Storage, Riverside, CA, USA, 24–26 October 2021; pp. 1–6.

34. Dong, J.; Fan, G.; Zheng, F.; Lin, J.; Xiao, F. TX-RSA: A high performance RSA implementation scheme on NVIDIA Tegra X2. In Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications (WASA), Nanjing, China, 25–27 June 2021; pp. 210–222.
35. Dziembowski, A.; Mieloch, D.; Stankowski, J.; Grzelka, A. IV-PSNR—The objective quality metric for immersive video applications. *IEEE Trans. Circuits Syst. Video Technol.* **2022**, *32*, 7575–7591. [CrossRef]
36. Fang, C.; Guo, S.; Wang, Z.; Huang, H.; Liu, Y. Data-driven intelligent future network: Architecture, use cases, and challenges. *IEEE Commun. Mag.* **2019**, *57*, 34–40. [CrossRef]