ELSEVIER

Parallel Computing 28 (2002) 1039–1078

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Video compression with parallel processing [☆]

Ishfaq Ahmad [a,*], Yong He [b], Ming L. Liou [c]

[a] Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX, USA

[b] Motorola, San Diego, USA

[c] Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology, Hong Kong

Received 1 April 2001; received in revised form 1 September 2001; accepted 1 December 2001

Abstract

Driven by the rapidly increasing demand for audio-visual applications, digital video compression technology has become a mature field, offering several available products based on both hardware and software implementations. Taking advantage of spatial, temporal, and statistical redundancies in video data, a video compression system aims to maximize the compression ratio while maintaining a high picture quality. Despite the tremendous progress in this area, video compression remains a challenging research problem due to its computational requirements and also because of the need for higher picture quality at lower data rates. Designing efficient coding algorithms continues to be a prolific area of research. For circumvent the computational requirement, researchers has resorted to parallel processing with a variety of approaches using dedicated parallel VLSI architectures as well as software on general-purpose available multiprocessor systems. Despite the availability of fast single processors, parallel processing helps to explore advanced algorithms and to build more sophisticated systems. This paper presents an overview of the recent research in video compression using parallel processing. The paper provides a discussion of the basic compression techniques, existing video coding standards, and various parallelization approaches. Since video compression is multi-step in nature using various algorithms, parallel processing can be exploited at an individual algorithm or at a complete system level. The paper covers a broad spectrum of such approaches, outlining the basic philosophy of each approach and providing examples. We contrast these approaches when possible, highlight their pros and cons, and suggest future research

[☆] This work was supported by Research Grants Council of Hong Kong under grants HKUST6030/97E, HKUST 6228/99E, and HKTIIT98/99.EG02.

[*] Corresponding author. Address: Department of Computer Science and Engineering, The University of Texas at Arlington, P.O. Box 19015, 416 Yates St., 248 D/E Nedderman Hall, Arlington, TX 76019-0015, USA.

E-mail address: iahmad@cse.uta.edu (I. Ahmad).

directions. While the emphasis of this paper is on software-based methods, a significant discussion of hardware and VLSI is also included.

Keywords: Video compression; Image processing; Parallel algorithms; Cluster computing; Parallel architectures; Problem and data portioning

1. Introduction

The current revolution of information technology has created a vast number of innovative applications and services, such as digital TV/high definition television (HDTV) broadcasting, voice over the Internet, video streaming, networked multimedia conference, virtual classroom, personal video recording content-based database search and retrieval, etc. For these applications, digital video is now as essential as the traditional media such as text, images, and audio. But the massive amount of data to represent video poses serious storage and communication challenges [25], despite the advancements in disk and networking technologies. Video compression, therefore, is essential for reducing the storage and communication requirements of multimedia systems. For instance, digital television broadcast and video retrieval and streaming on the Internet are only feasible with compressed video data. For storage of digital video on small disks such as CD or DVD (digital versatile disk), compression is necessary. Similarly, other than compression, there is currently no other way of obtaining the quality demanded by the entertainment industry while at the same time storing feature length films, which may last up to two hours or longer.

Efficient digital representation of image and video signals has been a subject of prolific research over the past two decades. Digital video compression technology is developed into a mature field, with several techniques available for a wide range of applications. Taking advantage of spatial, temporal, and statistical redundancies in the video data, numerous compression algorithms (such as predictive, transform, subband, entropy and fractal coding) are now in existence and several new ideas are being explored [31]. Experts have harnessed several of these algorithms and techniques to develop video coding standards. Standardization of coding methodology is also necessary to ensure correct communication between encoders and decoders developed by various groups and industries [44]. The international video standardization development are mainly lead by International Telecommunication Union-Telecommunications (ITU-T, International Telecommunication Unit) (H.261, H.263, H.263+ and H.26L) [24,52,53,69] and Moving Picture Experts Group (MPEG) (MPEG-1, MPEG-2 and MPEG-4) [75–77]. These standards do not prescribe the encoding methods, rather they only specify formats (syntax) for representing data inputs to the decoder, and a set of rules (semantics) for interpreting them. This implies that there does not exist a unified way of implementing the encoder and, furthermore, continuing improvements in the encoder are possible with various kinds of optimizations even after the standard is complete.

Compression algorithms require a substantial amount of processing, which may range from 100 mega operations per second (MOPS) to more than one tera operations per second, depending on the target quality and data throughput. Broadly speaking, there are two approaches in video compression: hardware-based and software-based. A hardware solution with a dedicated architecture design may achieve a high compression speed. In addition, some special purpose programmable digital signal processors (DSPs) or multiprocessors-on-chip can be used flexibly for various video processing with high performance [20]. However, hardware is less flexible and unsuitable for frequent updates. Due to the ever-changing communication and signal processing standards, most application systems should be preferably programmable. A software solution using a general-purpose computing platform is more flexible, and allows optimal algorithm updates from time to time. The main obstacle in software approach is that it requires a massive amount of computing power to support real-time encoding operations. However, the latest developments in parallel and distributed systems promise a higher degree of performance. Now, with the proliferation of networked computers using off-the-shelf components and portable parallel programming environments such as PVM, and MPI, parallel processing can be exploited in a cost effective manner. This paper is motivated by an extensive amount of research work done in video compression using parallel processing. While our focus is on software-based approaches, we provide an overview of hardware-based parallel architectures as well. A wide range of parallel processing approaches exists for video compression, each with its own basic philosophy and merits as well as demerits.

This paper is organized as follows: Section 2 begins by describing the fundamentals of digital video and its representation, followed by an overview of existing compression standards. The same section touches upon the basics of hardware and software-based approaches. Section 3 presents an overview of parallel processing technologies, including architectures and parallel programming software. Section 4 presents a survey of the video compression techniques using parallel processing subdividing the section into four subsections: The first subsection describes parallel VLSI implementations of various parts of compression, such as vector quantization (VQ), discrete cosine transform (DCT), wavelets, variable length coding (VLC), and motion estimation (ME). The second subsection presents complete encoders implemented using parallel hardware. The third subsection describes encoding techniques that are entirely implemented in software. This section also includes a discussion on various parallelization approaches. The last subsection explains how to exploit parallelism even from a single processor by processing multiple data items within the same instruction. Finally, Section 5 provides some concluding remarks and observations.

2. Digital video and compression technologies

This section presents some fundamentals of digital video, core compression techniques, an overview of existing video compression standards, and basic implementation approaches.

2.1. Fundamentals

A still picture or an image represents the distribution of light intensity and wavelengths over a finite size area. Video is a sequence of still images (frames) of a scene taken at various successive time intervals. Human visual system cannot react to rapid changes of intensity, so that a sequence of slightly different frame is perceived to be a smooth motion. The minimum number of frames necessary for smooth motion rendition is 15 frames per second (fps), but annoying flicker may still be perceived. Films use 24 fps (to avoid flicker), computer uses 60 fps, and television standards, such as PAL and NTSC, use 25 and 29.97 fps, respectively.

Video data originates from a camera wherein a transducer converts the light intensity to an analog electrical signal. The analog signal is then digitized by taking samples of the input signal at a minimum rate (called *Nyquist rate*) of twice the highest frequency present in the signal. Usually 8 bits (256 levels) per pixel are sufficient for an end-to-end system. The video signal consists of Red, Green and Blue components, which are then converted to luminance (Y) and two color components, U and V (or I and Q). Color components are usually subsampled as the human eye is less sensitive to colors, resulting in the size of U and V components reduced by a factor of 4.

The data rates for digital video are simply enormous. For example, uncompressed CCIR (ITU-R) 601 with resolution of 720 pixels/line and 576 lines (RGB) has a data rate of close to 300 Mbps without color space subsampling. The first step is usually color subsampling, which itself is compression. After color subsampling, the bit rate of CCIR 601 video (YUV) is reduced close to 150 Mbps. After compression, MPEG-2 coding of the CCIR601 video sequence may require only 4–15 Mbps with acceptable visual quality. The bit rates for uncompressed HDTV signal may be around one Gbps. Even a lower resolution video conferencing application using CIF resolution (352×288) may need more than 15 Mbps. The main objective of video compression is to reduce the data rate while preserving the video quality. In general, the higher the compression ratio (defined as the data size of the uncompressed video over the size of compressed video), the lower the picture quality. In general compression ratios in access of 50 may be needed for practical purposes. Table 1 provides estimated uncompressed and compressed bit rates for various applications [45].

2.2. Basic compression techniques

As illustrated in Fig. 1, the entire compression and decompression process requires a codec consisting of an encoder and a decoder. The encoder compresses the data at a target bit rate for transmission or storage while the decoder decompresses the video signals to be viewed by the user. In general encoding is considerably more complex than decoding. Therefore, research and implementation efforts are more focused on encoding.

Compression can be lossless or lossy. In lossless compression, an image after compression/decompression is numerically identical to the original image on a pixel-

Table 1
Estimated bit rates for various applications

System	Video resolution (pels × lines × frames/s)	Uncompressed bitrate (RGB)	Compressed bitrate
Film (USA and Japan)	(480 × 480 × 24 Hz)	66 Mbits/s	3 to 6 Mbits/s
CCIR 601 video (digital TV)	(720 × 576 × 30 Hz)	150 Mbits/s	4 to 15 Mbits/s
HDTV video	(1920 × 1080 × 30 Hz)	747 Mbits/s	18 to 30 Mbits/s
HDTV video	(1280 × 720 × 60 Hz)	664 Mbits/s	18 to 30 Mbits/s
ISDN videophone (CIF)	(352 × 288 × 29.97 Hz)	36 Mbits/s	64 to 1920 kbits/s
PSTN videophone (ACIF)	(176 × 144 × 29.97 Hz)	9 Mbits/s	10 to 30 kbits/s
Two-channel stereo audio		1.4 Mbits/s	64 to 384 kbits/s
Five-channel stereo audio		3.5 Mbits/s	384 to 968 kbits/s

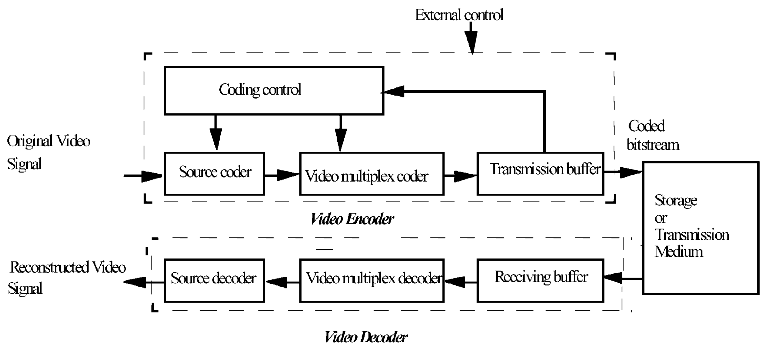

Fig. 1. Block diagram of a video codec.

by-pixel basis. Only a modest amount of compression, generally in the range of 1.5:1 to 4.0:1 (depending on the original image characteristics such as image detail, noise, resolution and bit-depth), can be achieved. In lossy compression, the reconstructed image contains degradations relative to the original image. As a result, much higher compression can be achieved. In general, lower bit rates are achieved by allowing more degradation. These degradations may or may not be visually apparent, depending on the viewing conditions. The term visually lossless is sometimes used to characterize lossy compression schemes that result in no visible degradation. This definition, however, is subjective and highly depends on the viewing conditions. An algorithm that is visually lossless under certain viewing conditions (e.g., a 19-in. monitor viewed at a distance of 4 feet) could result in visible degradations under more stringent conditions.

Compression can be done with variable bitrate (VBR) 3+ + … or constant bitrate (CBR). In VBR video, the encoder tries to achieve best encoding and pays attention to output buffer (no overflow or underflow). VBR video in general has a better picture quality, for example the DVD video. In CBR video, which is typically used in video streaming on the Internet and LANs with specified bandwidth, the objective is to maintain a specified bitrate.

Video compression may be done with one or more of the following objectives, several of which conflict with one another:

- high degree of compression;
- high quality of video;
- low complexity for implementation;
- low cost;
- small delay;
- low power and memory requirement;
- resilience to transmission errors.

According to different target applications, video compression can also be categorized as real-time or non-real-time. In the former case, video compression must be achieved on-line, implying the video signal is continuously generated from a source, which is then digitized and encoded at about 30 fps. In the latter case, data may be pre-stored on a storage device, allowing compression can be done off-line without strict compression speed requirements.

A generic encoder showing transform and predictive coding is depicted in Fig. 2, a structure [1] common to MPEG-1, MPEG-2, MPEG-4 (with additional functions), H.261, H.263, and H.263+ (see Section 2.3). The first step is to perform ME by block matching. ME refers to finding the displacement, henceforth referred to as motion vector, of a particular macroblock (MB) (16×16 or 16×8 pel area) of the current frame with respect to a previous or future reference frame or both of them. All searches are based on the minimum-error matching criteria (see Section 4.1.5). The next step is motion-compensated prediction in which the motion vector is used to reconstruct the predicted frame. The predicted frame is then compared with the original frame to determine the difference signal, i.e., the prediction error. Spatial redundancies in the original frame are also transformed into the frequency domain, using a block transform coding technique such as two-dimensional (2D) 8×8 DCT. The prediction error is also compressed using the DCT. The resulting 63 AC transform coefficients are mapped to a 1D data before it is quantized in an irreversible process that discards the less important information. Several variations exist between standards on ME, DCT coefficient scanning, quantization, and so on.

Finally, the motion vectors are combined with the DCT information, and transmitted using variable length codes. The VLC tables may be optimized for a limited range of compression ratios appropriate for the target applications. The output video buffer may be observed regularly to tune the quantization step size. The de-

[1] At a finer detail level, there are several differences among the coding methods employed by these standards.

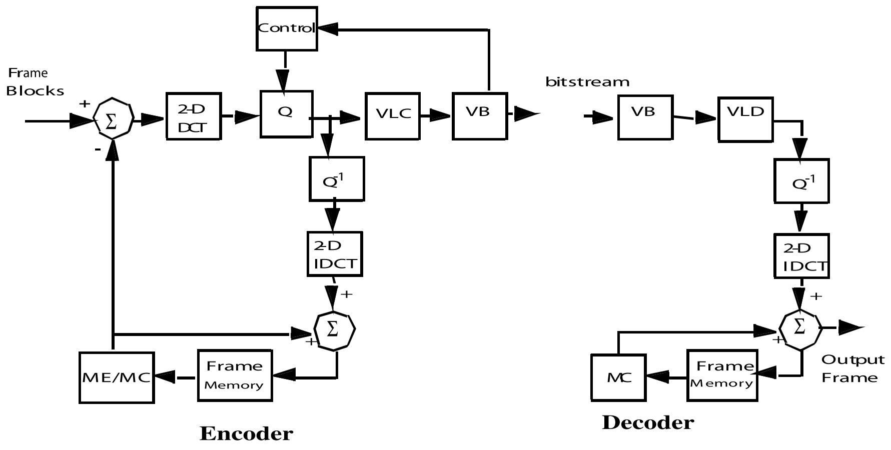

Fig. 2. Generic structure of a video encoder and decoder (similar to MPEG-1, MPEG-2, H.263, etc.).

coder goes through a similar process but in the reverse order, with some exceptions such as it does not perform ME.

2.3. Video compression standardization

Standardization of audio–visual information compression is necessary to facilitate global interchange of digitally encoded video data, and to enable the sharing of technology across diverse industries. In addition, coding standards are efficient as they support good compression algorithms and allow efficient implementation of encoder and decoder. For multimedia communication, two major standard organizations are the ITU-T, and the International Organization for Standardization (ISO). During the last decade, a number of ISO and ITU standards, such as MPEG-1, MPEG-2 and H.261, H.263, have been developed targeted towards different application domains. These standards define the bitstream syntax of audio–visual data and dictate a set of guidelines and rules that must be followed in developing hardware or software compression solutions. Video compression standards do not strictly prescribe the coding process but they do take advantage of the prevailing research in compression techniques and recommend a set of tools or algorithms for achieving an efficient compression and decompression. MPEG-1 and MPEG-2 standards are well adapted by the industry while new standards such as MPEG-4 and H.263+, are generating immense interest due to their potential to exploit new functionalities and capabilities. Table 2 provides an overview of various standards.

H.261 [69] is a video coding standard defined by the ITU-T for providing videophone and video-conferencing services at $p \times 64$ kbps ($p = 1, \ldots, 30$) bit rate which covers the entire ISDN channel capacity. The basic coding structure of H.261 includes spatial and temporal coding schemes. Spatial coding involves taking the DCT of 8×8 pixel blocks, quantizing the DCT coefficients based on perceptual

Table 2
Various video compression standards

	H.261	H.263	MPEG-1	MPEG-2	MPEG-4
Format	CIR/QCIF 29.97 fps (max)	Flexible mostly QCIF	SIF 30 fps	Flexible	Flexible
Compressed bitrate	$p \times 64$ kbps $p = 1, 2, \ldots, 30$	Mostly <28.8 kbps	1.5 Mbps	>2 Mbps	Flexible
Applications	Videophone/ Videoconferencing	Mostly videophone	VCR quality entertainment video	Wide range of applications	Multimedia, Web video
Transport	N-ISDN	Mostly telephone line or wireless	Primarily ADSL	DVD and digital TV broadcast	Various media

weighting criteria, storing the DCT coefficients for each block in a zig-zag scan, and performing a variable run-length coding for the resulting DCT coefficient stream; temporal coding is achieved by motion-compensated prediction.

H.263 [52] is aimed at compressing the moving picture component of audio–visual services and applications at a very low bitrate. Although H.263 video codec is based on the same DCT and motion compensation (DCT–MC) techniques as used in H.261, H.263 provides better picture quality at low bit rates. H.263+ [24] is specified by ITU-T to enhance the H.263 standard in terms of coding efficiency, error resilience, and functionalities. H.263+ is backward compatible with H.263 and extends the negotiable options enabling either improved quality, or additional capabilities to broaden the range of applications.

MPEG-1 [36] is defined by the MPEG of the ISO and International Electrotechnical Commission. It is a widely successful video coding standard with the coding capable of VHS video quality, or better, at about 1.5 Mbps and covering a bit rate range of about 1–2 Mbps.

The second standard developed by the MPEG, named MPEG-2 [45], is a generic coding standard for low to high-resolution moving pictures and the associated audio data with the bit rate ranging from 2 to 30 Mbps. MPEG-2 was designed to encompass MPEG-1 and to also provide better quality with interlaced video sources at higher bit rates. In addition, MPEG-2 supports a variety of packet formats and provides an error correction capability that is suitable for transmission over cable TV and satellite links.

As Fig. 3 illustrates, MPEG-2 video is structured in a hierarchical fashion. A video sequence (as in encoded order) consists of groups of pictures (GOPs), with each group starting with a reference picture, also called I frame (intra coded) followed by a number of frames. I frames are only intra coded and provide random access in a scene. The rest of the frames are either P (predicted) or B (bi-directionally predicted). Each frame can be partitioned into slices and each slice consists of a number of MBs. A MB consists of 6 blocks, 4 luminance and 2 chrominance blocks, where each block is an 8×8 pixels data unit. From parallel processing point of view, this structure offers various levels of granularity (the partitioned amount of data assigned

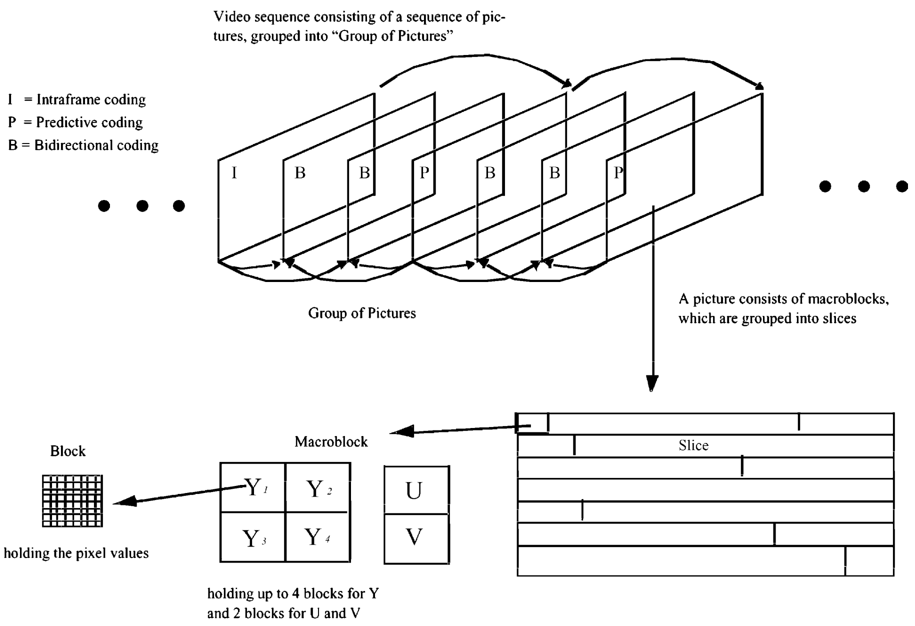
Video sequence consisting of a sequence of pictures, grouped into "Group of Pictures"
I = Intraframe coding
P = Predictive coding
B = Bidirectional coding
I
B
B
P
B
B
P
Group of Pictures
A picture consists of macroblocks, which are grouped into slices
Block
holding the pixel values
Macroblock
Y1
Y2
Y3
Y4
U
V
holding up to 4 blocks for Y and 2 blocks for U and V
Slice

Fig. 3. The hierarchical structure of MPEG-2 video.

to each processor). The granularity can be coarse (with one or more frame assigned to each processor) or fine (with one MB assigned to each processor). An MB is the smallest data unit in parallel processing as ME is performed at MB level.

Due to the rapid progress in information technology, the traditional boundaries between areas of telecommunication, computer and TV/film have become rather blurred. In order to handle the special requirements from rapidly developing multimedia applications, MPEG developed a new standard, MPEG-4 [96]. MPEG-4 is a multimedia standard that specifies coding of audio and video objects, both natural and synthetic; a multiplexed representation of many such simultaneous objects; and the description and dynamics of the scene containing these objects. MPEG-4 supports a number of advanced functionality's: (a) efficient encoding of multimedia data such as audio, video, graphics, text, images, and animation; (b) scene description for multimedia scene; (c) error resilience to enable robust transmission of compressed data over noisy communication channels; (d) support for arbitrary-shaped video objects; (e) multiplexing and synchronizing the data associated with these objects so that they can be transported over network channels providing a quality of service; and (f) object-based interaction and manipulation with the scene.

MPEG-4 includes most of the technical features of the prior video and still-picture coding standards, as well as a number of new features such as zero-tree wavelet coding of still pictures, segmented shape coding of objects, and coding of hybrids of synthetic and natural video objects [95]. MPEG-4 covers essentially all bit rates, picture formats and frame rates, including both progressive and interlaced video pictures. As can be imagined, MPEG-4 supports a diverse set of functionalitys and thus a very wide range of applications.

2.4. Implementation strategies

There are a number of approaches to implement the video compression. These approaches can be roughly divided into two categories: hardware-based implementation using dedicated VLSI and software-based implementation using general purpose computing platforms.

Since, a software solution also needs a hardware circuitry to run the application, there exists some overlap between software and hardware, which can be explained through Fig. 4. The size of the circle in the figure roughly represents the "total resources" needed for each platform. The application specific integrated circuit (ASIC) solution is totally hardware and is the most efficient approach in terms of required silicon area. Programmable ASICs and video signal processors (VSP) require a little more circuitry in order to accommodate a limited degree of software. DSPs require less "total resources" than general-purpose processor because they are designed for scientific applications but have more software capability and hence more circuitry than the programmable ASICs (or VSPs). Finally general-purpose processors need more resources and have the maximum flexibility in terms of programming.

2.4.1. Hardware-based approach

The most common approach is to design a dedicated VLSI circuit for video compression [74]. One can have function specific hardware, such as block matching, DCT, VLC and associated inverse operations. Due to the exploitations of the special control and data flow of the algorithm, the processing capability of these approaches can be increased tenfold compared to those of conventional microprocessors [88]. However, function specific approaches provide limited flexibility and cannot be mod-

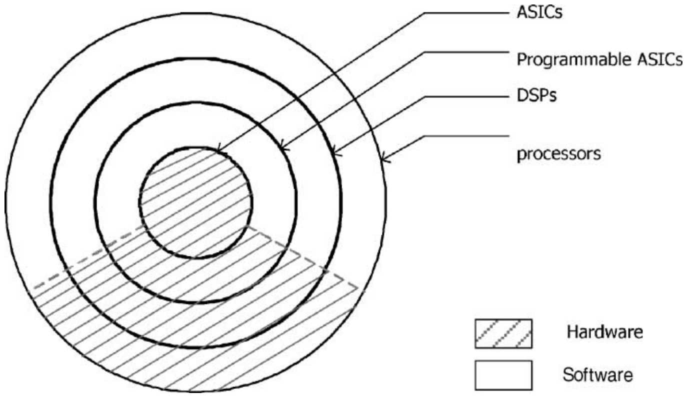

Fig. 4. Various implementation approaches and hardware–software relationship.

ified for later extensions. In addition, the architecture design usually requires a regular control paradigm and data flow of the algorithms that may not be suitable for most content-based techniques. Furthermore, the complexity limitations of the circuit design, such as processing speed, silicon area, throughput and the number of translators, also restrict its implementation potential for growing multimedia applications.

Programmable processors, such as programmable DSP or VSP, provide a more cost-effective alternative [33]. Such an approach can execute different tasks under software control, which can avoid cost intensive hardware redesign. It is flexible in that it allows the implementation of various video compression algorithms without the need for a hardware redesign. In addition, multiple algorithms can be executed on the same hardware and their performance can be optimized as well [60]. Usually, programmable processors require silicon area for control unit and program storage, and dissipate more power than dedicated VLSI solutions. Consequently, their implementation time and cost increase accordingly. Furthermore, they also incur significant costs in software development and system integration.

2.4.2. *Software-based approach*

Software-based approaches are becoming more popular because the performance of general-purpose processors has been increasing rapidly. Further, the rapid evolution of multimedia techniques has dramatically shortened the required time-to-market making it very difficult to come up with a new hardware design for each updated technique. Furthermore, more and more emerging multimedia standards emphasize high-level interactivity, flexibility, and extensibility, posing significant opportunities for software-based solutions.

The major advantage of using the software-based approach is that it allows to incorporate new research ideas and algorithms in the encoding process for achieving a better picture quality at a given bit rate, or alternatively reduce the bit rate for a desired level of picture quality. The inherent modular nature of various video compression algorithms allows experimenting with and hence improving various parts of the encoder independently, including DCT, ME algorithm and rate-controlled coding. The software-based approach is also flexible in that it allows tuning of various parameters for multiple passes for optimization. Added benefits of software-based approach are portability, and flexibility to adapt to the continuing changes in multimedia applications.

While decoding can be easily done in software, encoding is more challenging due to enormous amount of computation required. Encoding for simple video profiles of various standards can now be done on a single processor, but real-time performance for high-quality profiles is still quite difficult. A natural alternative is to utilize the accumulated processing capability of parallel processing to speed up the compression [92]. However, parallelism can be exploited in different ways, ranging from simultaneous instructions execution within a single processor, to distributed networks and massively parallel processors (MPPs), and there is no unique philosophy for the best solution. It is important to recognize that parallel processing alone may not be enough in software-based implementation, rather one needs to optimize all design and implementation phases, including efficient algorithms for DCT, fast ME and other parts of the encoder [4]. In addition, low-level programming primitives that take advantage of the machine architecture must be harnessed to accelerate the computation [84,102]. Finally, several issues should be addressed in software-based parallel processing such as I/O [2,4], memory access [98,99], and achieving better rate control [12,78,83,101].

3. Parallel processing technologies

Limits to sequential computing arise from the traditional processor architecture, processor–memory bandwidth and the physical limitations of circuit design. In recent years, the performances of commodity off-the shelf components, such as processor, memory, disk, and networking technology, have improved tremendously. Free operating systems are available and well supported.

3.1. Parallel architectures

Driven by the intensive computational processing demand over the past two decades, a variety of parallel architectures with opposing philosophies have evolved, such as distributed versus shared memory, SIMD versus MIDM, tightly coupled versus loosely coupled [49]. Since it is beyond the scope of this paper to provide a detailed overview of various parallel architectures, we briefly describe the architectures that can be exploited for video compression.

3.1.1. Symmetric multiprocessors

A symmetric multiprocessor (SMP) system uses commodity microprocessors with on-chip and off-chip caches. The processors are connected to a shared memory through a high-speed snoopy bus or a switch. It is important for the system to be *symmetric*, in that every processor has equal access to the shared memory, the I/O devices, and the operating system services. Most SMPs have at most 64 processors, such as the Sun Ultra Enterprise 10000. The limitation is mainly due to a centralized shared memory and a bus or crossbar system. Examples include the IBM R50, the SGI Power Challenge, and the DEC Alpha server 8400. SMP systems are heavily used in commercial applications, such as databases, on-line transaction systems, and data warehouses. SMP can be very useful for video compression because of their low cost.

3.1.2. Massively parallel processors

The term MPP generally refers to a very large-scale computer system that is built using commodity microprocessors in processing nodes, each having its own memory and connected by a network with high communication bandwidth and low latency. MPPs can be scaled up to hundreds or thousands of processors. The program consists of multiple tasks, each having its private address space. Tasks interact by passing messages. Examples are the Intel Paragon and TFLOP. MPPs are not commercially viable but can be useful for off-line encoding of very large videos, such as a digital library.

3.1.3. Distributed shared-memory machines

The main difference between distributed shared-memory (DSM) machines and SMP is that the memory is physically distributed among different nodes. However, the system hardware and software create an illusion of a single address space to application users. A cache directory is used to support distributed coherent caches. A DSM machine can be also implemented with software extensions on a network of workstations such as the TreadMarks. Examples are the Stanford DASH, and Cray T3D. Due to the ease of programming and scalability, DSMs can be potentially exploited for video compression.

3.1.4. Cluster computing

A cluster is a collection of complete computers (nodes) interconnected by a high-speed network or a local-area network [49]. Typically, each node is a workstation, PC, or SMP. Cluster nodes work collectively as a single computing resource and fill the conventional role of using each node as an independent machine. A cluster computing system is a compromise between a massively parallel processing system and a distributed system. An MPP system node typically cannot serve as a standalone computer; a cluster node usually contains its own disk and a complete operating system, and therefore, also can handle interactive jobs. In a distributed system, nodes can serve only as individual resources while a cluster presents a single system image to the user.

Cluster of workstations [4] or PCs (Beowlf) [85] are growing rapidly as a result of the advances in high-speed communication networks and the economics of replication. Clusters have several advantages such as high availability, rapidly improved performance, cost effective, high network throughput, scalability, and easy integration [1].

Recently, PCs or workstations-based clusters connected together with high-speed networks have emerged as popular platforms, primarily because of the fast speed of processors and low cost of off-the-shelf components. Several standard parallel programming environments, such as MPI, PVM and BSP, are also available that are added as message-passing libraries in standard C or Fortran programs running on top of Windows or Unix operating systems. In addition to traditional scientific and engineering problems, parallel processing is becoming increasingly popular for commercial applications [41].

Clusters of commodity PCs and workstations using off-the-shelf processors and communication platforms such as the Myrinet, Fast Ethernet, and Gigabit Ethernet, are becoming increasingly cost effective. With proper design, clusters can provide scalability by allowing the system to scale up or down, and hence, offering a variable range of computational power. Furthermore, since each computing node in a cluster is equipped with a disk, the cumulative disk space of the whole system can be substantial. These features of cluster computing are highly suitable for implementing software-only encoders needing heavy but low-cost computing power. The inherent problem of inefficient inter-processor communication in cluster computing can be overcome if the video data is partitioned in a coarse-grained manner and thus reducing the amount of communication data.

3.2. Processor architectures

Word-parallelism: At the data word level through SIMD [2] instructions enabling multiple data to be executed simultaneously. Such parallelism takes advantage of the intrinsic parallelism that exists in the computation of data and generally can be applied when there is regular, repetitive computation. In general, digital video and image processing have high data-level parallelism potentials since each pixel, block or MB data of each image or frame of video sequence performs the same coding procedures repetitively, such as DCT/IDCT transform, quantization, ME and compensation. This is a kind of data-parallel processing employed in both general-purpose processors and DSPs. Currently, most media processors use SIMD-based multimedia accelerate instructions to speedup the multimedia processing. Typical examples of such multimedia extensions are Intel's MMX (MultiMedia Extension) technology for extending the $\times 86$ architecture [84]; HPs MAX and MAX-2 accelerate primitives in its 64 bits PA-architectures [64]; Sun's VIS (Visual Instruction Set) in UltraSPARC [102]. SIMD approach has also been adopted in DSPs such as TigerSHARC of Analog Devices [34], AltiVec of Motorola [28], etc.

Instruction parallelism: At the instruction level through very long instruction word (VLIW) or superscalar scheduling to execute a group of independent instructions in parallel. Instruction level parallelism is used to execute several instructions from a single process that are independent from one another and coordinated either at run-time by conventional dynamic superscalar processors or VLIW processors [57]. Major superscalar $\times 86$ microprocessors include Intel Pentium, Pentium Pro and Pentium-2; AMD K5 and K6; and Cyrix 6×86. High-end superscalar microprocessors include PowerPC620/604e, HP PA8000, MIPS R10000/5000 and Digital Alpha 21164. VLIW style has been introduced mainly into latest DSPs (for example, Trimedia TM-1 of Philips, TI C6X DSP and Chromatic Mpact) for multimedia and networking services.

Multi-processors integration on a chip is a form of very coarse parallelism, process-level parallelism, involves complete independent applications running in independent processes controlled by the operating system. The efficient exploitation of multiple processors or core on a chip include various technologies such as real-time scheduling, load balance prediction and detection, memory access methodology, parallel I/O design and inter-processor communications.

[2] The SIMD paradigm here implies processing of multiple data in one instruction; this is not to be confused with the SIMD paradigm at the multiprocessor level.

3.3. Software for parallel processing

There are currently four models of parallel programming that are widely used on real parallel computers: implicit, data parallel, message-passing, and shared variable [49].

A parallelizing compiler on vector multiprocessors, VLIW and superscalar processors typically exploits implicit parallelism, usually expressed by a user in the program through compiler directives or language constructs (e.g., HPF for data-parallel programming). The compiler performs dependence analysis on the source code and uses a suite of program transformation to convert the sequential code into native parallel code. In the case of video compression, loops on blocks of video data may benefit from implicit parallelism. However, the efficiency of even the manually optimized codes is not high and improvement can be achieved by combining user direction and run-time parallelization. Therefore, only part of the video encoding program may be parallelized.

Data-parallelism, using both SIMD and single program multiple data (SPMD) programming paradigms, can be very effective for video compression because there is ample opportunity for exploiting data-parallelism due to the characteristics of video and image processing, such as common integer operations, predicted memory-access patterns, and fixed data precision, etc. While SIMD paradigm at the multiprocessor level is no longer very popular, exploiting parallelism through the processing of multiple data words for the same instruction is become wide spread. Under the SPMD paradigm, the data is partitioned into smaller pieces that are assigned to different processors. A single program is written for all processors that asynchronously execute the program on their local piece of data. Communication of data and synchronization is done through message-passing. Several standard open tools for message passing are now available for different parallel architectures. MPI [104] and PVM [37] for message passing that extend Fortran or C programs with library subroutines.

Shared-memory programming model using shared variables and locking schemes is attractive due to its ease of programming but is not very highly portable. Cost-effective software-based video compression solutions can be designed using platforms such as SMPs or multiple-processor PCs. While there is no widely adapted standard for shared-memory model, several tools and libraries such as X3H5 and POSIX *Pthreads* multithreading library are available [49]. A thread is a piece of code within an application that runs concurrently with the application's other threads, sharing an address space with them, along with access to the application's variables, file handles, device contexts, objects, and other resources. Threads are different from processes, which typically do not share resources or an address space and communicate only through the mechanisms provided by the operating system for inter-process communication, such as pipes and queues. Threads often use simpler and less resource-intensive forms of communication like semaphores, mutexes, and events. In using multithreading to implement parallelism, the overhead caused by thread creation and thread synchronization can counteract the benefits of parallelism. Because threads require synchronization mechanisms to guard against race conditions in shared data, the volume of processing data in each thread can be a major factor in determining whether a process is suitable for multithread processing. Multithreading is also supported by Java Virtual Machine to enable concurrent execution of the multiple portion of a program.

4. Parallel processing approaches for video compression

In this section, we describe various parallel processing techniques that have been employed for different algorithms or to build complete compression systems.

4.1. Parallel video coding algorithms on VLSI

Various coding techniques are widely deployed for video compression targeting different bandwidth and quality. Some compression algorithms, such as DCT/IDCT, ME and entropy coding, are shared by most video coding standards due to their excellent performance and reasonable complexity. The high computational complexity of some algorithms is an impediment for real-time implementation in many applications. Fortunately, many of these algorithms are structured in such a way that a small number of basic arithmetic operations are repeatedly applied to a large body of data. In addition, the operations of the algorithm are very regular with a predefined sequence of operations and data access. Therefore, these algorithms can be mapped onto specific VLSI architectures or DSP/VSPs with extensive use of parallel processing such as pipelining and multi-processor techniques [87].

From VLSI design perspective, compressed video is an application that places special demands on the technology in terms of parallelism, scalability and programmability [38]. Parallel processing has been applied to VQ [32,112,113], DCT/IDCT [21,55,97], wavelet transform [72,89], VLC [14], and most compute intensive tasks, such as ME [15,18,19,30,35,43,48,51,68,86,90,110,111,114–116]. Further details are provided below.

4.1.1. Vector quantization

VQ is a powerful technique for very low bit rate image/video compression. The computation complexity of VQ is high (for example, the encoding complexity for K input vectors of dimension L, and a codebook of size N is $O(KLN)$ [1]). In VQ, each block of symbols is uniquely represented by a subinterval within 0 and 1. Its coding scheme is a two-stage process: The first stage is to determine the subinterval the block represents. The second stage is to find the binary representation (with minimum number of bits) of a value that lies within that subinterval.

Several processor architectures are proposed to implement VQ for real-time video compression. The processor designed in [32] can handle 2048 template vectors by a single chip using massively parallel operation and a new search algorithm. A pipeline VQ algorithm proposed in [112] makes efficient use of the parallel capabilities of the SIMD machine. A memory-based parallel processor was proposed in [113] for VQ, which accelerates the neighborhood search by computing all distances between the input and reference vectors in codebook in all processors simultaneously.

4.1.2. Parallel discrete cosine transform

The DCT is an orthogonal transform with superior energy compaction property and near optimal performance. 2D DCT has been widely accepted as the most effective technique in image and video compression. The DCT function reduces spatial redundancy in video and image data. DCT provides the basis for compression on the 8×8 pixels block by decomposing pixels value into a weighted sum of spatial frequencies, and IDCT is the inverse of it. Compression is achieved by quantization. Using 8×8 DCT/IDCT blocks, the required real-time processing speed for CIF at 30 f/s is approximately 73 M multiplications per second. The 8×8 2D DCT and IDCT used in MPEG compression are defined in the following equations.

Forward 8×8 2D DCT:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\frac{(2x+1)\pi u}{16}\cos\frac{(2y+1)\pi v}{16} \tag{1}$$

Inverse 8×8 2D DCT (IDCT):

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7} \alpha(u)\alpha(v)C(u,v)\cos\frac{(2x+1)\pi u}{16}\cos\frac{(2y+1)\pi v}{16} \tag{2}$$

where $\alpha(k) = 1$, if and only if $k \neq 0$, otherwise,

$$\alpha(k) = \frac{1}{2\sqrt{2}}$$

The DCT coefficients are then scanned in a zig-zag scan fashion to convert the 2D signal into 1D bitstream. The coefficients are then quantized using a quantization matrix (see Fig. 5), followed by run-length entropy coded by using variable-length code (VLC) tables.

The chip design proposed in [97] uses a bit-serial/bit-parallel architecture and distributed arithmetic to implement a 16×16 DCT. The chip contains 32 processing

8	4	2	1	0	0	0	0
4	2	1	0	0	0	0	0
2	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

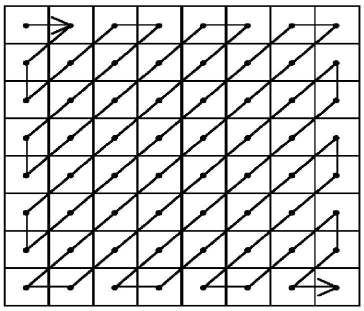

Fig. 5. DCT quantization and zig-zag scanning.

elements (PEs) working in parallel for performing a 16×16 matrix transposition. The chip designed for real-time processing performs an equivalent of a half billion multiplications and accumulations per second.

Most of the 2D DCT architectures are implemented by the row-column decomposition methods. In [21], an architecture combining pipelining and frame-recursive architecture is proposed in which the 2D DCT is implemented by 1D DCT lattice arrays, such that the total number of multipliers required for an $N \times N$ 2D DCT is $8N$. The design is shown to be very suitable for VLSI implementation for bit-rate systems. Another VLSI architecture for DCT is proposed in [55] that uses row-column decomposition to implement a 2D transform. The architecture also uses distributed arithmetic combined with bit-serial and bit-parallel structures to implement the required vector inner product concurrently.

4.1.3. Parallel wavelet

The discrete wavelet transform (DWT) is an efficient tool for image and video compression. Various dedicated VLSI ASIC solutions have been proposed based on either SIMD architecture or pipeline realization. An efficient approach using general-purpose parallel computers without inter-processor communications was proposed in [72]. The basic idea is to use a matrix-vector multiplication with intrinsically parallel filter bank algorithm to perform the DWT in standard form. Each filter can be described as a matrix and the filtering as matrix-vector multiplication. The algorithm is tested on AT & T DSP-3 multiprocessor system (16–128 processors), and speedup was found to be higher than traditional parallel methods that use input data partitioning. A massively parallel video codec proposed in [89] implements wavelet transform on an array of intelligent pixel processors.

4.1.4. Parallel variable length coding

Several image and video coding standards have adopted VLC due to its high compression efficiency and algorithmic simplicity. A VLC encoder is usually realized by using a table look-up implementation. The concatenation of the variable-length codewords can be done bit-serially, demanding high operation speed at the output and causing the bottleneck of the data-intensive application. A parallel VLC encoder architecture reported in [14] enables the codeword concatenation in one clock cycle. Therefore, the encoder can operate at the same rate as the input sampling data rate.

4.1.5. Motion estimation

ME is the central part of MPEG1/2/4 and the H.261/H.263 video compression standards and has attracted much attention in research and industry for the reason that it is computationally the most demanding algorithm of a video encoder (about 60–80%). In addition, it also has a high impact on the visual quality [58]. Prediction is accomplished by ME that is done in the spatial domain. ME information together with error image (if any) requires much less number of bits to code than that of the original block. Prediction is based on a previously processed frame. In block-based ME, the algorithm compares each block of a frame at time t with the corresponding block in the frame at time $t-1$ and then within a certain range (a window) around that position. Block matching is a widely used ME algorithm in current video systems. Typically ME is performed on a block size of 16×16, or 8×8. The size of search window depends on frame differences, speed of moving objects, resolution, etc. The matching criteria include accuracy and complexity. Let $C(x+i, y+j)$ denote the pixels inside the MB at location (x, y) in the current frame, and let $R(x+i+u, y+j+v)$ denote the pixels inside the MB in the reference frame with motion vector (u, v) defined within the search range, $0 \leqslant i \leqslant 15$, $0 \leqslant j \leqslant 15$ for 16×16 block.

The algorithm compares the pixel values of the two blocks using the sum of absolute difference (SAD), defined as follows:

$$\mathrm{SAD}(u,v) = \sum_{i=0}^{16} \sum_{j=0}^{16} |C(x+i, y+j) - R(x+i+u, y+j+v)| \tag{3}$$

The mean absolute error (MAE) is calculated by dividing SAD by 256. Alternatively, some algorithms use mean square error (MSE), defined as:

$$\mathrm{MSE}(u,v) = \frac{1}{256} \sum_{i=0}^{16} \sum_{j=0}^{16} [C(x+i, y+j) - R(x+i+u, y+j+v)]^2 \tag{4}$$

Finding the best matching MB, with the minimum error, in the reference frame is the core operation in ME. Rather than searching the entire reference frame, the ME algorithm restricts the search range to a $[-p, p]$ region around the (x, y) location of the MB in the current frame, (see Fig. 6). The ME algorithm finds a motion vector by matching the current block with the blocks at displacement (u, v) in the previous frame.

Among typical searching strategies, the full search scheme provides better precision and regular data flow as well as a higher degree of parallelism, a characteristic that is advantageous for VLSI implementation. Some of the well-known techniques are hierarchical approach, subsampling of motion field, reducing number of checking points (fast search algorithm), etc. (see [59] for a detailed survey on ME).

The full search is not optimal but is considered to be a reference since it uses an exhaustive method to find the best match. Fast search algorithms are suboptimal and not robust in general. Further, they require much less computation and are suitable

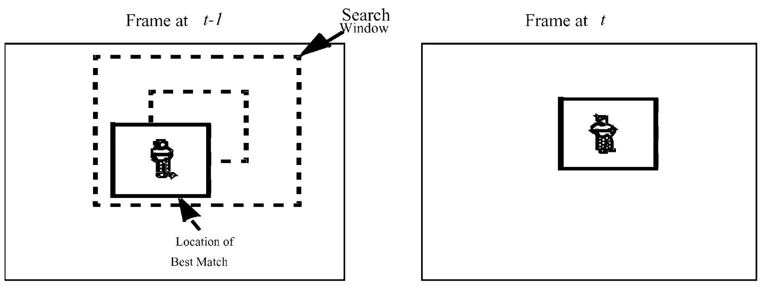

Fig. 6. Block-based ME.

for software implementation [109]. The complexity of real-time processing for the full search ME can be calculated as follows: Let, the block size $= 16 \times 16$, and window size $= 32 \times 32$, then assuming CIF frame at 30 fps, we need

$$\left(256\,\frac{\text{ops}}{\text{search}}\right)\left(289\,\frac{\text{search}}{\text{block}}\right)\left(296\,\frac{\text{block}}{\text{frame}}\right)\left(30\,\frac{\text{frame}}{\text{sec}}\right) = 879\ \text{Mops/s}. \tag{5}$$

For CCIR 601 or HDTV, it would require several or tens of giga operations per second (GOPS)/s. The full search has been used in certain applications with specially designed hardware. Other search algorithms with reduced complexity have been and are still being developed.

A carefully designed ME chip (or module) implies that it will meet application requirements such as the desired performance level, proper throughput, and able to match with I/O bandwidth. It should also be cost effective, and occupy the minimum chip size and utilize minimum set of computational resources.

The full search can find the best matched motion vectors, but it requires high computational power. Therefore, VLSI-based implementation is critical for its processing speedup. A VLSI architecture for implementing a full-search block-matching algorithm is proposed in [48]. Based on a systolic array processor and shift register arrays with programmable length, it allows serial data inputs to save the pin counts but performs parallel processing. The parallelism and pipelining nature are fully exploited with the 2D systolic array that reduces the computational complexity.

A linear array architecture for a full-search block matching algorithm is proposed in [114] that uses a parallel algorithm based on the idea of partial result accumulation. Combining a serial data input with registers for a line of search window pixels and operating on them in parallel, the partial results of the candidate block distortions are obtained for all horizontal search positions. A real-time encoding rate is reported.

Most VLSI realizations of full search ME are based on systolic arrays, laid out for a specific set of parameter values. Hence these approaches do not offer flexibility. In contrast, programmable DSP can offer a large flexibility, however, since a DSP is conceived for a large variety of image/video processing algorithms, only part of the DSP hardware can be used for common block matching. A flexible VLSI architecture for the full search is proposed in [110]. Flexibility is increased by cascading and/or with parallel operations. Several identical block-matching chips are placed in a serial way. Only the first chip of the cascade is directly connected with the external buffers. The data supplied to the first chip is then passed on from chip to chip within the cascaded duration.

A programmable ME processor is introduced in [86], which has ability to work in parallel making it ideal for real time applications like video compression. The architecture implements four 4×16 arrays that can support all target matching size, such as 4×8, 8×8, 8×16 and 16×16 pels block matching. Another programmable architecture is presented in [15] featuring cascadable modules of PEs with simple interconnection. Therefore, flexibility in computing power is available. The architecture is implemented with 32 PEs in one chip that consists of 102K transistors.

A 165 GOPS exhaustive Me processor ME3 has been developed [43]. The ME3 is powerful enough to realize a range of −64/+63 pixels horizontally and −32/+31 pixels vertically with a single chip. A dual-array architecture supports both the joined operation and the parallel operation of the PE-arrays. The parallel operation realizes a wide search range with a single chip configuration. ME3 chips can work concurrently in a cascade fashion to support a multi-chip configuration, so that the exhaustive search range can be easily expanded to meet the requirements of high-resolution encoder. A 64-processors architecture is reported in [90] that consists of four clusters of 16 processors each, with each cluster working in a pipelined fashion. The ME algorithm combines full search block matching with sparse search.

The existing fast ME approaches can be identified into three groups: (1) fast search by reduction of motion vector candidates; (2) fast block-matching distortion computation; and (3) motion field subsampling [109]. However, the fast algorithms are often designed to merely reduce arithmetic operations without considering their overall performance in VLSI parallel implementation. A new three-step fast search algorithm based on hardware consideration proposed in [115] uses a 1D systolic array as the basic computing engine (see Fig. 7). A checking-vector-based search strategy processes several adjacent checking points. Compared to other widely used fast ME algorithms, this scheme can be implemented cost-effectively in terms of silicon area, I/O rate and processing speed. The paper also proposes a VLSI design for block ME in [116], which results in more than 60% reduction in power consumption.

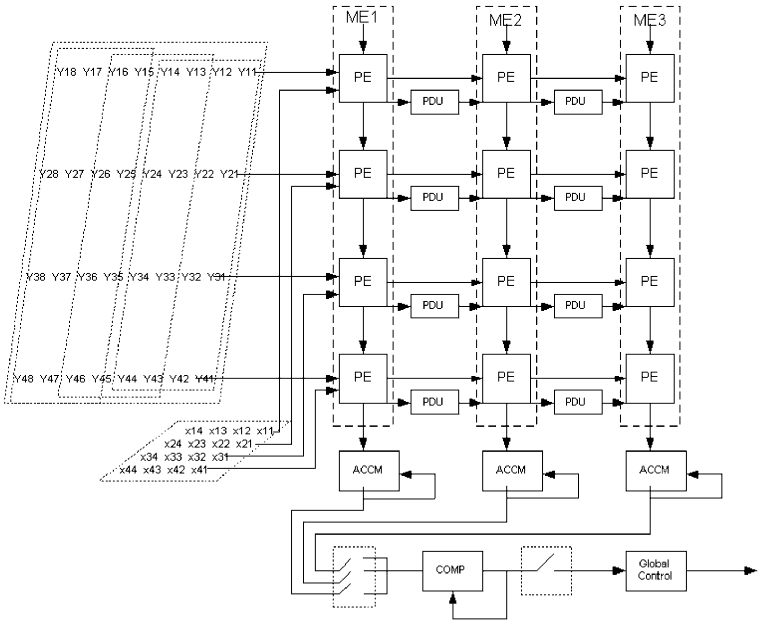
ME1
ME2
ME3
PE
PDU
ACCM
COMP
Global Control
Y18 Y17 Y16 Y15 Y14 Y13 Y12 Y11
Y28 Y27 Y26 Y25 Y24 Y23 Y22 Y21
Y38 Y37 Y36 Y35 Y34 Y33 Y32 Y31
Y48 Y47 Y46 Y45 Y44 Y43 Y42 Y41
x14 x13 x12 x11
x24 x23 x22 x21
x34 x33 x32 x31
x44 x43 x42 x41

Fig. 7. Three 1D systolic array architecture.

This is a useful attribute as power consumption is also one of the major concerns due to the presence of multiple processors.

A flexible architecture for ME and compensation using a 1D pipeline systolic array is reported in [111] to implement a fast ME algorithm with an aim to be incorporated into the digital compression unit of a single-chip video camera. The target application is QCIF resolution at 20 fps, with a source rate of 3.877 Mbps and the compressed bitrate of 56 kbps. Another fast block-matching algorithm called the parallel hierarchical 1D search (PHODS) for ME is proposed in [19]. Instead of finding the 2D motion vector directly, PHODS finds two 1D displacements in parallel on the two axes (say x and y) independently within the search area. A fully pipelined parallel architecture for the 3-step hierarchical search block-matching algorithm is reported in [51]. The paper also proposes techniques for reducing interconnections and external memory accesses. The proposed 3-PE, 9-PE, and 27-PE architectures provide efficient solutions for real-time MEs required by video applications of various data rates, from low bitrate video to HDTV systems. An architecture proposed in [68] is designed for a high-speed motion estimator using the 2D log search algorithm. It consists of five simple PEs where each PE is capable of computing the SAD to exploit the parallelism. For each step in the 2D log search procedure, the 5 SADs of the 5 search points are computed in parallel. The architecture is well suited for encoding MPEG2 video up to the main profile.

Various hardware engines have been for implementing different block-matching algorithms. Some of them utilize array processors, while some others use LSI-based pipelined architectures or ASIC-based VLSI systems. But they can only implement a single algorithm. The advantage of having a programmable ME engine is that the chip can be used in different search algorithms and the custom can upgraded ME algorithms. In [30] a flexible architecture is proposed for implementing widely varying motion-estimation algorithms. To achieve real-time performance, it employs multiple PEs. Three different block-matching algorithms-full search, three step search, and conjugate-direction search-have been mapped onto this architecture to illustrate its programmability.

Besides block matching algorithms, other ME algorithms are also of interest in parallel VLSI architecture design. For example, a pipelined parallel CORDIC architecture is proposed in [18] that estimates motion in the DCT domain instead of the spatial domain. The linear array of processors architecture design described in [35] leads to column parallelism in which one processor is allocated to one or more image rows. The array of processors performs the ME process for a whole column of pels in parallel using a pel-recursive ME.

4.1.6. Rate control

Rate control algorithms try to optimize the bit rate through proper adjustment of quantization and bit allocation. Availability of parallel processing implies that more detailed and complex rate control algorithms can be designed and implemented. Several papers report algorithms for bit allocation and rate control in software-based parallel encoders. The scheme proposed in [101] analyzes future frames in order to determine bit allocation for the current frame. The analysis uses of preliminary

ME, masking factor computation, and coding regions of future frames to model their coding complexity.

In [83], a number of rate control methods for parallel encoder architecture using picture partitioning are proposed, which are suitable for MPEG-2 VIDEO Main Profile @ High Level encoders. Partitioning boundaries are sometimes visible in decoded pictures because of control discontinuity. An adaptive bitrate allocation scheme is proposed to reduce the discontinuity of picture quality around partitioning boundaries. Another rate control scheme for an MPEG-2 HDTV parallel encoding system consisting of multiple encoding units is described in [78]. An effective scene change rate control algorithm for software-based video compression on network of workstations is proposed in [12]. Taking advantage of the available processing power, the system uses two-pass bit allocation and a rate control algorithm for generating high quality output.

4.2. Complete encoders implemented on parallel hardware

Because of the defect density, production of very large area chips is not economical. Therefore, most encoders are implemented with a moderate number of chips [11,70,71]. In order to enable high complexity encoder systems integration into a single chip, some video DSPs (VSP) use pipeline structures supporting multiprocessors-on-a-chip [17,54,73,81,106]. There exist several examples of parallel architectures for dedicated video processors. A multimedia processor based on a SIMD architecture optimized for block-based video processing algorithms, called digital generic processor, consists of 64 pixel processors (SIMD) connected by a programmable network, and a 32 bit RISC controller [73]. The SIMD array has a peak computational power of 1.7 GOPS to perform video compression according to standards such as H.261, H.263, MPEG-1 and MPEG-2. The processor can be programmed in order to run the code corresponding to each specific algorithm. It can easily satisfy real-time decoding of MPEG-2 and H.263. The various speedup factors for MPEG range from 12 to 45. Another programmable parallel processor architecture for real-time video signal processing is reported in [81]. The architecture is composed of several parallel processor clusters and bus-switch units. Each parallel processor cluster can process one frame picture in real time.

A single-chip VSP supporting H.261 coding standard reported in [106] has a maximum performance of 300MFOPS. The VSP is based on a pipeline processing unit structure, in which each of the four data processing units has a three-stage pipeline. Thus, the VSP can execute 12 operations in one machine cycle. Multiple VSPs systems can be used to achieve further speedup using a tightly coupled share-memory system. With 4 VSPs, the frame rate of H.261 can reach 10 fps at 20 MHz and 12.5 fps at 25 MHz. A parallel implementation of H.263 video encoder on quad DSP (TMS320C6201) platform is presented in [118], reporting real-time encoding speed of 30 fps for CIF (352×288) picture. A single-board all-ASIC CMOS implementation of the H.261 video encoder and decoder together with pre- and post-processing modules is reported in [70]. The encoder consists of multiple chips, each carrying out a different function, such as pre-filtering DCT/IDCT, motion vector detection, buffer control and quantization, VLC, forward error correction, etc. It uses 14 ASICs together with 20 MB SRAM.

A multimedia video processor (MVP), called UWGSP5 [54], performs real-time MPEG video and audio encoding at SIF (360×288) resolution, 30 fps. The processor contains multiple DSP cores that are capable of maintaining a very high rate of operation, and especially optimal to perform DCT, FFT and other video processing routines. The Grand Alliance HDTV encoder developed by Motorola Broadband Communication Sector (formal General Instrument) and AT & T is reported in [71]. This encoder hardware is partitioned into 23 circuit boards supporting different functions respectively. The entire video frame is spatially partitioned into 6 panels. Each panel encoding processor formats compressed data to the MPEG "slice" layer. It can handle high peak 62.2 Mpixels/sec video processing. In addition, the world's first all-digital HDTV system, the DigiCipher® MPEG-2 system developed by Motorola BCS, also implemented spatial parallelism on multi-processor platform for achieving real-time encoding performance. A VLSI chip, named MOVIE [17], is designed to facilitate the development of software-only solutions for real-time video processing applications. This chip can be seen as a building block for SIMD processing, and its architecture has been designed so as to facilitate high-level language programming. Parallelism levels in the MOVIE architecture is exploited using a subarray of PEs and one I/O processor. Two arrays of MOVIE are shown to be sufficient to implement a fully programmable MPEG-2 real-time encoder.

A parallel design using pipelining for compression/decompression of video data is described in [11]. The encoder tasks are divided into ME and error encoder, achieving a throughput of 30 fps with frame size of 352×272 pixels. VLIS architectures for simplified versions of MPEG-4 have started to emerge [39,100].

4.3. Software-based encoders

A considerable amount of work has been reported for software-only encoders, using computer systems ranging from massive parallel systems, such as Intel Paragon, Cray T3D and IBM SP2 [2,5,80,94,108], to cluster of computers [4,6,79]. A wide variety of schemes for parallel encoding on using both DSPs and general-purpose processors, exists based on temporal, spatial, data-parallel, and control-parallel strategies [3,13,22,27,29,66,67,80,91,103]. Cluster of workstations, due to their cost-effectiveness are more useful for flexible software-only video encoding [4,6,79, 107].

4.3.1. Spatial parallelism

Spatial parallelism allocates different parts of a single frame to different PEs. The granularity of the workload may be slices or MBs (see Fig. 8). Processors concurrently encode data sets and then combine the results to generate the final video bitstream [5]. Due to frame dependencies, processors may need to reference other processors' encoding results or previous frame's encoding results for MC. To deal with this problem, one can send overlapped parts of frames to each processor, but it causes overhead in the I/O and scheduling. Due to the limited spatial resolution

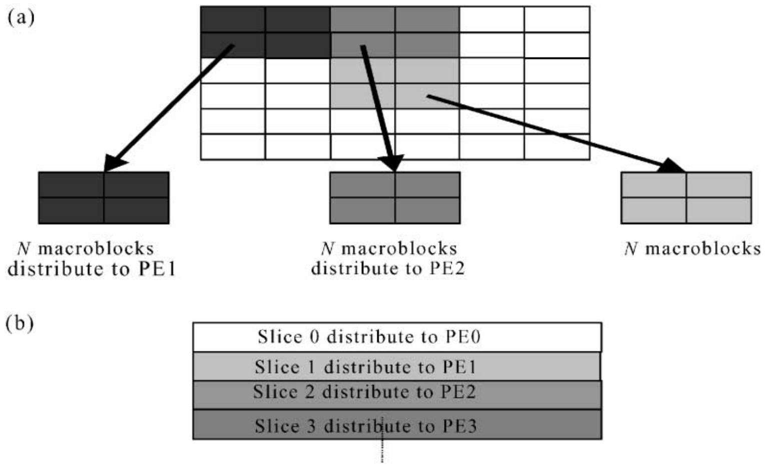
(a)
N macroblocks distribute to PE1
N macroblocks distribute to PE2
N macroblocks
(b)
Slice 0 distribute to PE0
Slice 1 distribute to PE1
Slice 2 distribute to PE2
Slice 3 distribute to PE3

Fig. 8. Spatial parallelism using MB and slice distribution: part of the picture is decomposed and distributed at MB (a) and slice (b) level.

of a video frame, only a limited number of processors can be used [40,94]. Another disadvantage is that the encoding system needs to tolerate a relatively large communication overhead.

A fine-grained software-only MPEG-2 video encoder on various parallel and distributed platforms and its performance is described in [6]. The platforms include an Intel Paragon XP/S and an Intel iPSC/860 hypercube parallel computer as well as various networked clusters of workstations. The encoder exploits parallelism by distributing each frame across the processors. The frame data may be divided among processors up to the point when the search window can be made available to the corresponding processor. The partitioning scheme accomplishes this by distributing one or more 16×16 block to each processor, and then either by sending the required boundary data to corresponding processors to form their local frame (see Fig. 9), that is search window, or equivalently, storing the redundant data at the local memory of each processor, which is necessary to form the search window to be used for determination of motion vectors for the next frame.

A 2D frame data may be mapped onto a 2D processors array. Let P and Q to be the height and the width of the frame respectively, and let p be the total number of processors to be used, with p_h to be the number of processors in the horizontal dimension and p_v to be the number of processors in the vertical dimension of the 2D grid. Thus, $p = p_h \times p_v$. If the search window size is the size of the MBs in a particular processor $\pm W$ in both dimensions, with overlapped (redundant) data distribution, given p_h and p_v, then the local frame size in each processor is given by

$$X_{\text{local}} = \left\lceil \frac{Q}{P_h} + 2W \right\rceil \left\lceil \frac{P}{P_v} + 2W \right\rceil \tag{6}$$

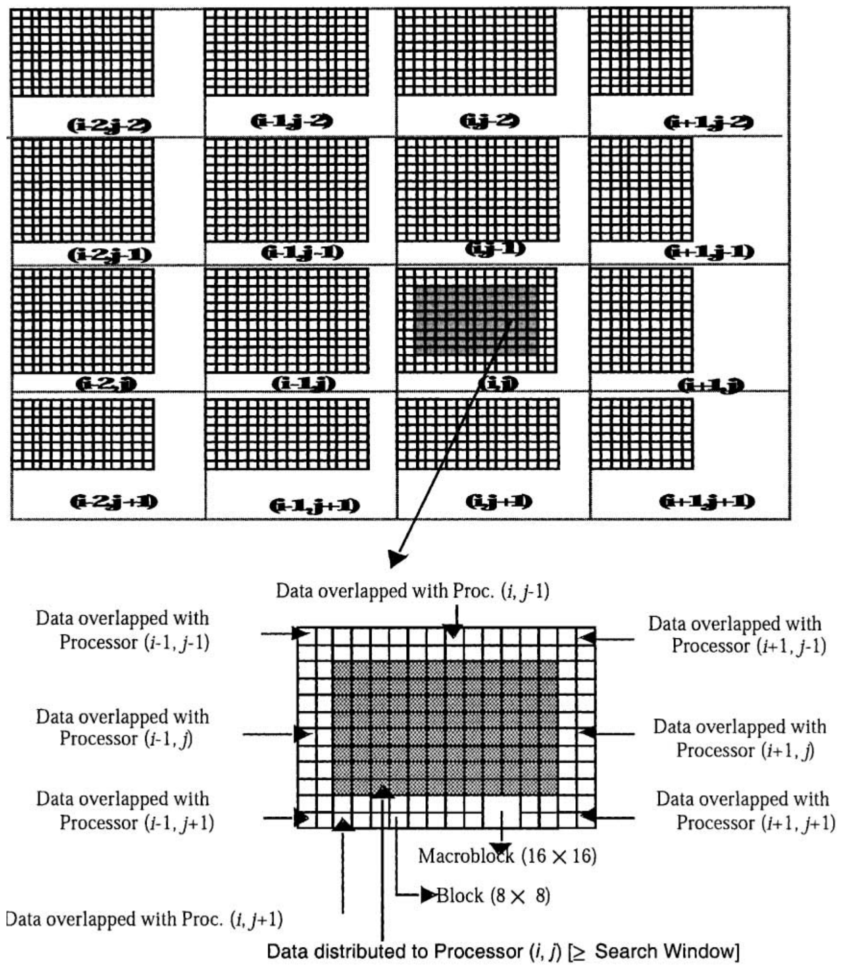
(i-2,j-2)
(i-1,j-2)
(i,j-2)
(i+1,j-2)
(i-2,j-1)
(i-1,j-1)
(i,j-1)
(i+1,j-1)
(i-2,j)
(i-1,j)
(i,j)
(i+1,j)
(i-2,j+1)
(i-1,j+1)
(i,j+1)
(i+1,j+1)
Data overlapped with Proc. (i, j-1)
Data overlapped with Processor (i-1, j-1)
Data overlapped with Processor (i+1, j-1)
Data overlapped with Processor (i-1, j)
Data overlapped with Processor (i+1, j)
Data overlapped with Processor (i-1, j+1)
Data overlapped with Processor (i+1, j+1)
Macroblock (16 × 16)
Block (8 × 8)
Data overlapped with Proc. (i, j+1)
Data distributed to Processor (i, j) [≥ Search Window]

Fig. 9. Data distribution to processor (i, j) and its overlapping region with neighboring processors.

Overlapped data distribution method allocating some redundant source data for ME circumvents the excessive amount of inter-processor communication. A MB is the finest grain size assigned to a processor, providing an upper bound on the maximum number of processors. Let the MB size be $w \times h$ (typically 16×16) for which motion vectors are to be determined.

$$p_{\mathrm{h,max}} = \left\lfloor \frac{Q}{w} \right\rfloor \quad \text{and} \quad p_{\mathrm{v,max}} = \left\lfloor \frac{P}{h} \right\rfloor. \tag{7}$$

Hence, for $w = h = 16$, the maximum number of processors is

$$p_{\max} = \left\lfloor \frac{Q}{16} \right\rfloor \times \left\lfloor \frac{P}{16} \right\rfloor \Rightarrow p \leqslant \left\lfloor \frac{Q}{16} \right\rfloor \times \left\lfloor \frac{P}{16} \right\rfloor \tag{8}$$

For example, if $Q = 360$, $P = 240$, then, $p_{\mathrm{h,max}} = 22$, $p_{\mathrm{v,max}} = 15$ and consequently $p \leqslant 330$.

Another version of this parallel MPEG-2 video encoder is implemented on various clusters of workstations connected via ATM switch and Ethernet [7].

Similar approaches are reported for other platforms and video standards. In a visualization application on massively parallel computer CRAY T3D [80], the whole image is divided into parts of equal size and each processor processes its own data partition. Depending upon the complexity of the compression algorithm, 8–12 images are compressed, decompressed and visualized per second. An H.261 encoder is implemented using a data-parallel approach on the IBM SP2 with 24 processors [108]. Using a load-balancing scheme across the processors, the encoder predicts the workload based on the previous frame workload, and schedules MB bounded by the locality constraint. The results show 19–23% reduction in the worst-case delay with both the prediction and scheduling overhead taken into account.

A parallel implementation of H.263 video encoder is presented [56] for video conferencing applications, allowing encoding of any of the five standard H.263 picture formats in real-time. The proposed approach uses data parallel implementation by partitioning the image in horizontal or vertical slices along the MB boundaries. With the prototype system using four ADSP-21062 DSPs, a real-time encoding is achieved with QCIF sized picture. Another parallel H.263 video encoder [23], exploiting spatial parallelism, is modelled using a multi-threaded program. The encoder subdivides a frame into equal parts (as far as physically possible). The parallel video coding is implemented on a PC using 2, 3 and 4 parallel threads, each with an additional thread for relocating the output bitstream. A software-based H.263 video encoder is implemented using a cluster of workstations [4]. Again the granularity is a marcoblock. The experimental results indicate an encoding rate of 30 fps for QCIF using 12 workstations.

An algorithm for automatic data partitioning and scheduling algorithm for video processing on multiprocessors is proposed in [63]. It proposes a compiler time processor assignment and data partitioning schemes that optimize the average run time performance of the task chains. The experiments carried out on CS-2 computing system using 4, 16 and 25 processors (PEs), shows a high performance gain.

Multiprocessor DSPs are also attractive since they offer performance typical of parallel machines together with limited cost. The paper provides performance analysis and software design issues according to different data partitioning models, such as overlap partitioning and non-overlap partitioning. The scheme proposed in [26] uses the idea of crossbar multiprocessor DSPs for data parallel local algorithms. Test results on TMS320C80 MVP show high speedup.

4.3.2. Temporal parallelism

In temporal parallelism, the allocation scheme assigns a portion of a video containing a number of video frames that need to be compressed as well as the necessary reference frames to each processor. One important question is how to schedule frames to processors with minimum load unbalancing and scheduling overhead. For example, if a processor encodes $B_4B_5P_6$, then it requires the previous reference

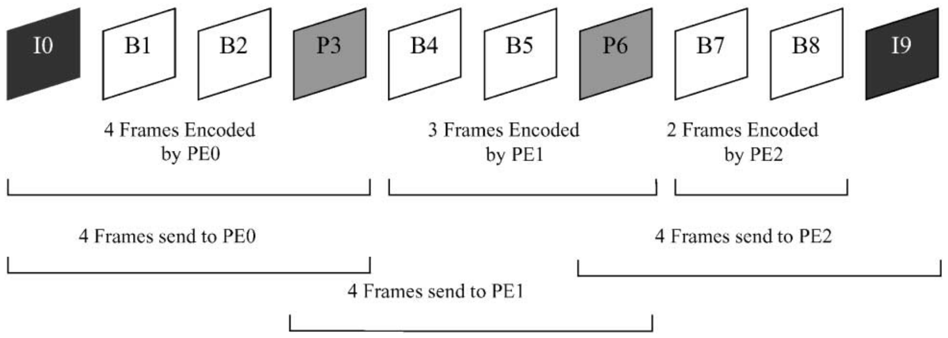

Fig. 10. Temporal parallelism with frame distribution including penalty frames.

frame, P_3, as shown in Fig. 10. Here P_3, P_6 and I_9 are penalty frames and are not encoded. If an original frame is used as a reference frame, this previous reference frame is read at least twice, by the processor that needs it as reference frame and by the other processor that encodes this P frame. A frame that is needed by another processor but is not encoded is a penalty frame. A similar but worse scenario is that if reconstructed frames are used as reference frames. Then the second process must wait for the reconstructed P frame from the first process, the third must wait for the reconstructed P frame from the second process, and so on. This lowers the degree of temporal parallelism. Making each processor encode frames starting with an I frame can solve this problem. When each process encodes GOP with pattern like $I_0B_1B_2P_3B_4B_5P_6$ (closed GOP size equal to seven for this case), in which no frame uses another GOPs frame as reference, there will be no penalty frames.

Temporal parallelism does not limit the number of processors but may cause delay in the encoding time due to rapid I/O traffic [2,93]. In addition, synchronization may be required after each round of frame encoding, which may add extra waiting times at various processors [94]. An administrative mechanism is needed to avoid this situation. One possible solution to solve this problem is to improve the overlap of frame distribution and encoding process [82,92]. Various issues involved in I/O for a parallel MPEG-2 video encoder on a cluster of workstations are discussed in [2]. Four scheduling schemes are proposed to remove I/O bottleneck.

Another scheme using temporal parallelism for MPEG-2 video encoder on the Intel Paragon parallel computer is reported in [2]. This scheme uses GOP as the granularity of parallelism. This work shows that the entire system including the computation, and I/O (reading of uncompressed data and writing of compressed data) has to work in a balanced fashion in order to achieve the final high throughput. The proposed scheme partitions multiprocessor system to groups of compute nodes, and schedules the computing nodes, I/O nodes, and disks in a highly balanced fashion. The experimental results show a frame rate of about 71 fps for SIF sequence on Intel Paragon using 126 processors. A parallelizing scheme for MPEG-1 video encoding algorithm on Ethernet-connected workstations reported in [79] uses the slice-level, frame-level. Experiments on thirty workstations shows that the MPEG-1 Video encoding time can be reduced in proportional to the number of workstations used in encoding computations although there is a saturation point in the speedup graphs.

Another possibility is to use spatial-temporal parallelism that is a combination of spatial and temporal approaches [92]. The scheme evenly divides the PEs into groups, each of which contains n PEs and compresses one video section at a time. Frames in the video section are spatially divided into m parts and processed by n PEs. The compressed data from each PE is sent to one of the PEs in that group, which assembles and writes the cumulative compressed data to a disk.

4.3.3. Shared-memory implementations

Parallel encoders have also been implemented on shared-memory parallel machines. A shared memory system, IBM Power Visualization System (PVS) is used for a parallel implementation of a MPEG-2 compliant, a constant bit rate video encoder [101]. Maximum PVS has 32 Intel i860 processors and 1 GB shared memory. The proposed scheme defines an I partition unit which includes all pictures between two I frames. Each processor then compresses one IPU at a time and parallelism at the IPU level is explored.

In another approach [27], a simple shared-memory parallel algorithm for MPEG-1 video encoding based on the parallel execution of different coarse-grained tasks (read pictures, write coded frames, I, P, and B frames coding) is proposed and implemented on the Sun enterprise 100000 containing 32 processors. The GOP and frame encoding tasks are mapped to the parallel system using a multithreaded methodology. Two synchronization strategies on encoding and writing operations are implemented.

4.3.4. Pipelining

Another possibility is to use pipelining. A real-time ITU-T H.263 codec using a multi-processor DSP system based on TMS320C80 MVP is reported in [103], which consists of 4 parallel 32-bit fixed-point DSP. The basic H.263 codec is estimated to be 40–50% computationally more demanding than H.261 at the same image size, frame rate and other conditions. For inter-frame encoding, the ME and DCT-related modules are carried out in parallel with two fixed pointed DSPs. Encoding tasks are carried out in parallel using task-partitioning pipelining method. The ME for a MB is carried out in PP0 and the rest of the encoding tasks of the same MB are then carried on PP1. The MVPs 64-bit parallel instructions are explored. The system works approximately with 29 fps for the decoder and 7.5 fps for the encoder with QCIF format. In [29], the H.261 coding algorithm is mapped onto a generalized image coder parallel-pipeline model using general purpose parallel processor systems, achieving a maximum speedup of about 11 on 16 processors.

An MPEG-2 video codec is implemented on a specialized DSP, the TMS320C80 MVP [13]. The proposed approach models the MPEG-2 codec as an 8×8 pixel block-based pipeline. These blocks flow through the four parallel processors of MVP. The pipeline consists of different codec performing ME, DCT, quantization, etc. The approach truncates the pipeline chain into 4 parts of equal delay, each part being executed sequentially on one dedicated parallel processor. The obtained codec is able to encode CIF-format 4:2:0 colour intra- and P-pictures in real-time (25 Hz) with one motion vector per 16/spl times/16 MB.

4.3.5. Object-based parallelization

As opposed to frame-based structure in previous video standards like MPEG-1, MPEG-2, H.261, and H.263, the structure of MPEG-4 is object based. An MPEG-4 scene may consist of a single video to multiple videos (rectangular or arbitrarily shaped) as well as images, text, graphics, audio, and animation. In addition, a scene is described by BIFS (binary format for scenes) language. Thus, each object in the scene has to be encoded individually including the scene description, and then multiplexed to form a single system bitstream. A parallel implementation of the MPEG-4 encoder has to take into account the dynamic nature of MPEG-4 video in which multiple objects may arrive or leave the scene. Furthermore, each object may need a different encoding efficiency and scalability level. Orchestrating various tasks of the encoder and distributing and dividing objects dynamically into pieces for concurrent execution requires efficient scheduling and load balancing schemes.

Several results showing object-based parallelism are reported. Parallelism is exploited to implement ME for MPEG-4. In a VLSI implementation [58] of the flexible high throughput ME architecture supporting fixed and variable block size full search ME, arbitrarily shaped video objects are supported. It is shown the additional area costs for the support of block-matching with arbitrary shaped objects are very small. Parallelization of an iterative partial quarter ME algorithm is shown with less than 1 dB PSNR loss. A 2D mesh-based object ME architecture that generates a content-based video object is reported in [9]. The proposed approach decomposes the mesh into independent triangular patches that can be processed in parallel. The three steps ME algorithm is used to simplify the ME of the mesh nodes. The proposed architecture is suitable for very low bit rate online applications and it can be used as a building block for an MPEG-4 codec.

Wavelet transform, which has inherent parallelism, is a potential candidate algorithm for MPEG-4. In a high speed VLSI architecture [16] of the DWT for MPEG-4, the input data are separated between even and odd, and the two data streams are inputted in parallel. This causes faster DWT operation than other architectures. In the proposed architecture the N-point DWT is computed in $N/2$ cycles with 100% hardware utilization.

In a parallel MPEG-4 system encoder on a cluster of workstations, [46] three scheduling algorithms are used for different situations. The encoder is scalable in that the encoding speed can be adjusted depending upon the number of available processors. A model generator first generates an analysis of the scene to determine object dependencies and then a scheduler uses this information to assign video compression tasks to a group of workstations. The scheduler also assigns the audio, image and text encoders to a workstation (see Fig. 11). The experiment results indicate that a real-time encoding rate can be achieved for the sequences with multiple media objects. Another version of a parallel MPEG-4 video encoder on a cluster of workstations with load balancing is described in [47]. A shape adaptive data partitioning

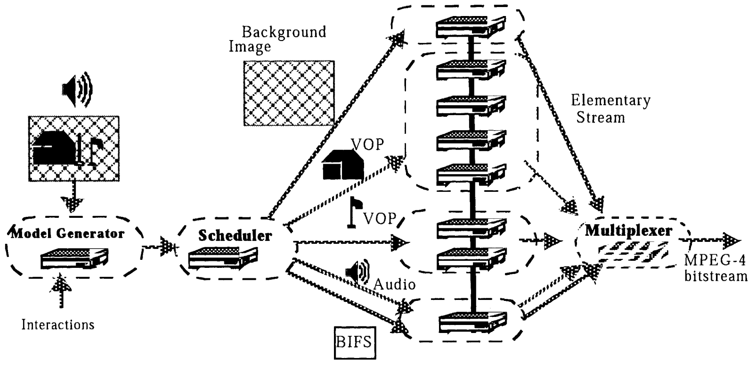

Fig. 11. An object-based MPEG-4 system encoder using a cluster of workstations.

method divides the arbitrary shaped video objects so that each workstation can complete the video encoding at near same time.

Another parallel software implementation of the MPEG-4 video encoder using multithreaded technique is reported in [42]. A scheduling policy is proposed to guarantee via a buffer synchronization a significant speed up which under some special circumstances reach an optimized load balancing solution. The proposed scheme is tested using the hardware resources of Unix multiprocessor hardware platform.

4.3.6. Exploiting multithreading

Multithreading can be powerful mechanism to implement video encoder on multiprocessor PCs and shared-memory system which are widely available and are cost effective. It is important to minimize the effect of synchronization overhead by keep the thread creation overhead small and by processing larger data blocks in each thread [50]. In order to improve the encoding speed and to minimize the overhead mentioned before, the number of threads, the amount of data processed by each thread, management of shared data I/O and the concatenation of results mechanism should be determined properly. In addition, threads should be properly scheduled. Without proper scheduling, thread switching and data sharing may cause extra overhead or resource deadlocks.

In terms of data distribution, GOP level temporal parallelism is the coarsest grain data distribution in MPEG-2 encoding. The multithreaded encoder proposed in [61] uses closed GOP (with pattern like IBBPBB1/4P) as the data block unit (see Fig. 12). No penalty frames exist because the frames inside each GOP do not make reference to the previous GOP. The encoder uses a fast RAID disk system for I/O. A raw video input thread reads raw video data into memory, operating concurrently with the encoding process. Double buffering with round-robin scheduling reduces the I/O wait time. A concatenator thread combines the encoded GOPs into a single MPEG-2 stream. As each of the encoded GOPs can be considered as one independent MPEG-2 stream, operations for the concatenator are read as unordered

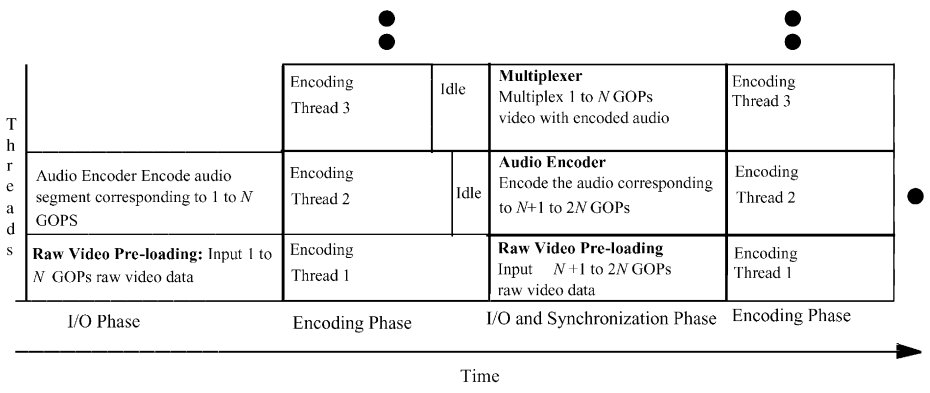
Threads
Encoding Thread 3
Idle
Multiplexer
Multiplex 1 to N GOPs video with encoded audio
Encoding Thread 3
Audio Encoder Encode audio segment corresponding to 1 to N GOPS
Encoding Thread 2
Idle
Audio Encoder
Encode the audio corresponding to N+1 to 2N GOPs
Encoding Thread 2
Raw Video Pre-loading: Input 1 to N GOPs raw video data
Encoding Thread 1
Raw Video Pre-loading
Input N+1 to 2N GOPs raw video data
Encoding Thread 1
I/O Phase
Encoding Phase
I/O and Synchronization Phase
Encoding Phase
Time

Fig. 12. Scheduling of threads in the multithreaded encoder.

encoded GOPs from different multithread encoders, refilling information of each frame inside the GOP and writing the single MPEG-2 stream in the right order. At the same time, the concatenator acts as a synchronizer for controlling the start of encoding next batch of GOPs. A separate thread performs audio encoding and multiplexing. The multiplexer also acts as a synchronizer, after each batch of encoding tasks, *N* GOPs from *N* threads are assembled by the multiplexer. Similar to the audio encoder, the multiplexer incurs a very little computation on the overall system.

In order to occupy the CPU time efficiently, multithreaded video encoders work exclusively. After encoding each batch of GOPs, raw video pre-loading thread, audio encoder and multiplexer grab all of the CPU time and work concurrently. Fig. 12 illustrates the scheduling of thread execution. In the very beginning, audio signal corresponding to first *N* GOPs is encoded and first *N* GOPs are streamed to the buffer by the raw video pre-loading thread. Then multithreaded video encoders start encoding video frames. Next, encoded video and audio are multiplexed. At the same time, the next batch of GOPs is streamed to the buffer and the corresponding audio signal is encoded, and so on.

The multithreaded MPEG-2 encoder runs on a multiple processor CPU. The objective is to have a full-scale MPEG-2 encoder Including audio encoding and multiplexing that is a cost-effective and complete practical solution that is not only highly efficient but is also scalable in that it can be used on a single-processor or multiple-processor PC. The encoder uses temporal parallelism with small overhead. The encoder achieves faster than real-time and half of real-time encoding rate for CIF (352×288) and CCIR601 (720×576) video sequences, respectively.

4.3.7. Miscellaneous parallel approaches

A system implementing a real-time H.263 codec using a multi-processor DSP system, TMS320C80 MVP is described in [67]. The intra frame encoding and the frame interpolation tasks are divided equally between 2 parallel processors (PP) using data-partitioning method where even MBs are coded in PP0 while odd MBs are coded in

PP1. For the inter-frame encoding modules such as ME, DCT/IDCT, quantization and bitstream encoding, tasks are carried out in parallel using the task-partitioning pipeline method. The ME is carried out in PP0 and the rest encoding tasks are then carried out on PP1. The results of both spatial and temporal parallelization of H.261 video coding algorithm on IBM SP2 multiprocessor system is also described in [117].

A single-chip 480 MOPS programmable DSP for real-time MPEG/H261 video codec is reported in [98]. The chip employs 4 SIMD processing units. The parallel architecture is suitable for parallel calculation of matrix-vector product, multiplication or addition among matrix elements, and matrix-scalar operations. These operations are frequently used in transform-based video encoding. The DSP can achieve 30 fps for CIP MPEG/H261 encoding.

Using a heterogeneous approach, a parallel processing system using a combination of a DSP and a PC-AT host processor to increase the computational speed of the MPEG compression algorithm is reported in [91]. The algorithm tries to divide its tasks, which are then allocated to the DSP as well as the host processor based on the relative merits of the tasks, and computed concurrently, resulting in a twofold increase in speed. In another heterogeneous approach, a multiprocessor system for high-speed processing of hybrid picture coding algorithms such as H.261, MPEG or digital HDTV is developed [22], using a combination of a highly parallel 32-bit microprocessor, DCT and ME function specific devices. In addition, an adaptive load balancing technique is also proposed that improves the performance of the system. An integrated image compression DSP, named VDSP2, integrates four different types of processors in the architecture that allows them to operate in parallel [3]. The device is capable of both encoding and decoding the MPEG2-based algorithm by changing programs on the same chip. A new dedicated hardware for ME is developed, which consists of two-pixel precision estimation and full and half pixel precision estimation. The codec can process MPEG2 main profile at main level in real-time at broadcast resolutions.

A Petri-net based representation is proposed to perform decomposition and scheduling of H.261 encoder on a TMS320C80 MVP [65]. The speedup of the H.261 is about 3.7 on 4 processors for QCIF and CIF video, which correspond to 30.7 and 9.25 fps. For a comprehensive analysis of major issues and difficulties in real-time DSP implementation of hybrid video coding, see [66]. The paper discusses methodologies for MOPS reduction of ME and DCT, code optimization, task division in parallel processing, and codec adaptation. Performance of real-time H.263 and MPEG-4 video systems using these techniques are demonstrated.

4.4. Exploiting parallelism within a single processor

Data-parallel instructions developed by Intel and SUN Microsystems allow to execute the same instruction on multiple data set simultaneously, hence allowing a single processor to mimic as an SIMD (Single Instructions Steam Multiple Data Stream) machine. These instructions are used as library routines. The algorithm developer and programmer needs to understand how to pack the data into processor registers in order to exploit parallelism. Operations such as ME and DCT/IDCT contain ample data-parallelism and, therefore, are natural candidates to benefit from these instructions.

SUN Microsystems proposed VIS for its Sparc processors. Previously, Intel's MMX instructions allowed SIMD integer operations on its Pentium processors. The recent streaming SIMD extensions (SSE) library expands the capabilities of the Intel Architecture by allowing floating-point operations on a set of eight registers. Furthermore, the SSE set provides new integer instructions as well as cache control instructions to optimize memory access.

Recall that ME computes motion vectors of pixel blocks, typically on a 16 by 16 MB. The motion vector is the relative displacement of the MB from one frame to a reference frame. Eq. (3) can be used to calculate the MAE (i.e., SAD/256). The C code for calculating MAE function is listed in Fig. 13.

The code needs 16×16 iterations for calculating the SAD. One SSE instruction, *psadbw*, can effectively speed up the SAD calculation. A 64-bit register stores eight 8-bit values, which is subtracted from another eight 8-bit values and the SAD is calculated. Block matching is implemented by using psadbw. Two psadbw instructions calculate the SAD between the pixels in one row of the reference and current MB. These two SADs are added to produce a 16-bit (word) result. It takes about 20 assembly instructions for calculating absolute differences of 16 pixels. For SSE, only two *psadbw* instructions are required. Moreover, fewer instructions are executed in SSE version because unrolling the loop four times save on loop overhead.

DCT and IDCT function provide the basis for compression on the 8×8 pixels block by decomposing pixels value into a weighted sum of spatial frequencies, and IDCT is the inverse of it. The DCT for one 8×8 elements block takes 4096 multiplications and 4032 additions. A number of algorithms are proposed for fast calculation of the 1D and 2D DCT that can be suitable for exploiting VIS and SSE. Most algorithms are variants of Lee's Fast DCT algorithm [62], Winograd's FFT [105], and AAN algorithm [10]. VIS and SSE can speed up the calculation of DCT by processing four 16-bit data elements in parallel [8].

```
#define A   BS(x) (((x)<0)?-(x):(x))
char *currentblock; /* pointer to    the macroblock in current
frame.*/
char *referenceblock;  /* pointer to the macroblock in reference
frame.*/
for ( i = 0; i < 16; i++ )
{
referenceblock = &(referenceframe[(y+v+i)*FRAME_WIDTH + u + x]);
currentblock = &(currentframe[(y+i)*FRAME_WIDTH + x]);
for ( j = 0; j < 16; j++)
{
SAD += ABS((referenceblock[j] - currentblock[j]));
}
}
MAE = SAD / 256;
```

Fig. 13. The C code for implementing MAE.

5. Conclusions

While the speeds of individual processors continue to rise at a sustained pace, the computing requirements driven by applications have always out-paced the available technology. This prompts the designers to seek faster and more cost-effective computer systems. Parallel and distributed computing goes one step ahead to provide computing power beyond the technological limitations of a single processor system. Both hardware and software approaches continue to make improvements, each with its own suitability to various applications. Cost effective hardware systems are now widely available. Software-based compression previously inconceivable is now extending from research laboratories and universities to the commercial market. Single chip solution are now widely available, but efficient designs using parallel architectures for multiple chips and pipelining techniques continue to result in fast encoder circuits for advanced applications. Programmable multimedia processors are becoming increasingly popular as they can be flexibly embedded in complex applications. While several simple video compression solution can now be handled by a single processor, efforts to achieve higher picture quality and lowering bit data rates continue to drive the research in compression to develop more complex algorithms that can benefit from the availability of parallel processors. Parallel processing can therefore be very effective for high-quality video applications where tuning of parameters and multiple passes of encoding are needed to generate the best possible compressed video signal.

Software-based approach requires optimizations at all design and implementation phases, including algorithmic enhancements, efficient implementations of all encoding modules, and taking advantage of certain architectural features of the machine. Since cost is a major factor in Internet-based applications, the availability of networked computers using off-the-shelf components and using portable parallel programming environments, parallel processing for software-only encoders can be exploited in a cost effective manner. In software-based compression system using parallel processing, the entire system including the computation, and I/O (reading of uncompressed data and writing of compressed data) must work in a balanced fashion in order to achieve the final high throughput. In addition, a scalable design is highly suitable for allowing adjustments in compression speed, resolution, and quality. Parallel processing also helps in designing new ME algorithms for more effective motion vector search and rate control algorithms for efficient bit allocation process.

References

[1] I. Ahmad, Gigantic clusters: Where are they and what are they doing, IEEE Concurrency (April–June) (2000) 83–85.

[2] I. Ahmad, S.M. Akramullah, M.L. Liou, M. Kafeel, A scalable off-line MPEG-2 encoder using a multiprocessor machine, Parallel Computing, in press.

[3] T. Akiyama et al., MPEG-2 video codec using image compression DSP, IEEE Transactions on Consumer Electronics 40 (3) (1994) 466–472.

[4] S.M. Akramullah, I. Ahmad, M.L. Liou, A software based H.263 video encoder using network of workstations, in: Proceedings of SPIE, vol. 3166, 1997.

[5] S.M. Akramullah, I. Ahmad, M.L. Liou, A data-parallel approach for real-time MPEG-2 video encoding, Journal of Parallel and Distributed Computing 30 (2) (1995) 129–146.

[6] S.M. Akramullah, I. Ahmad, M.L. Liou, Performance of a software-based MPEG-2 video encoder on parallel and distributed systems, IEEE Transactions on Circuits and Systems for Video Technology 7 (4) (1997) 687–695.

[7] S.M. Akramullah, I. Ahmad, M.L. Liou, Parallel MPEG-2 encoder on ATM and ethernet-connected workstations, in: Proceedings of 4th International Conference on Parallel Computation, Salzburg, Austria, February 1999, pp. 572–574.

[8] S.M. Akramullah, I. Ahmad, M.L. Liou, Optimization of H.263 video encoding using a single processor computer: Performance trade-offs and benchmarking, IEEE Transactions on Circuits and Systems for Video Technology 11 (8) (2001) 901–915.

[9] W. Badawy, M. Bayoumi, A VLSI architecture for 2D mesh-based video object motion estimation, in: International Conference on Consumer Electronics, 2000, pp. 124–125.

[10] Y. Arai, T. Agui, M. Nakajima, A fast DCT-SQ scheme for images, Transactions IEICE (1971) 1095–1097.

[11] S. Bhattacharjee et al., A parallel architecture for video compression, in: Proceedings of Tenth International Conference on VLSI Design, 1997, pp. 247–252.

[12] S. Bozoki, R.L. Lagendijk, Scene adaptive rate control in a distributed parallel MPEG video encoder, in: International Conference on Image Processing, vol. 2, 1997, pp. 780–783.

[13] O. Cantineau, J.D. Legat, Efficient parallelization of an MPEG-2 codec on a TMS320C80 video processor, in: 1998 International Conference on Image Processing, vol. 3, 1998, pp. 977–980.

[14] H.C. Chang et al., A VLSI architecture design of VLC encoder for high data rate video/image coding, in: Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, vol. 4, 1999, pp. 398–401.

[15] S. Chang, J.-H. Hwang, C.-W. Jen, Scalable array architecture design for full search block matching, IEEE Transactions on Circuits and Systems for Video Technology 5 (4) (1995) 332–343.

[16] S.J. Chang, A high speed VLSI architecture of discrete wavelet transform for MPEG-4, IEEE Transactions on Consumer Electronics 43 (3) (1997) 623–627.

[17] F. Charot et al., Toward hardware building blocks for software-only real-time video processing: the movie approach, IEEE Transactions on Circuits and Systems for Video Technology 9 (6) (1999) 882–894.

[18] J. Chen, K.J.R. Liu, A complete pipelined parallel CORDIC architecture for motion estimation, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 45 (5) (1998) 653–660.

[19] L.-G. Chen et al., An efficient parallel motion estimation algorithm for digital image processing, IEEE Transactions on Circuits and Systems for Video Technology 1 (4) (1991) 378–385.

[20] S.-C. Cheng, H.-M. Hang, The impact of encoding algorithms on MPEG VLSI implementation, in: IEEE International Symposium on Circuits and Systems, vol. 4, 1998, pp. 102–105.

[21] C.T. Chiu, K.J.R. Liu, Real-time parallel and fully pipelined two-dimensional DCT lattice structures with application to HDTV systems, IEEE Transactions on Circuits and Systems for Video Technology 2 (1) (1992) 25–37.

[22] S.H. Choi, K.T. Park, High-speed moving picture coding using adaptively load balanced multiprocessor system, Signal Processing: Image Communication 8 (2) (1996) 113–130.

[23] J.P. Cosmas, Y. Paker, A.J. Pearmain, Parallel H.263 video encoder in normal coding mode, Electronics Letters 34 (22) (1998) 2109–2110.

[24] G. Cote, B. Erol, M. Gallant, F. Kossentini, H.263+: Video coding at low bit rates, IEEE Transaction on Circuits and Systems for Video Technology 8 (7) (1998) 849–866.

[25] R.V. Cox, B.G. Haskell, Y. LeCun, B. Shahraray, L. Rabiner, On the applications of multimedia processing to communications, Proceedings of the IEEE 86 (5) (1998) 755–824.

[26] R. Cucchiara, A. Callipo, M. Piccardi, Block processing on multiprocessor DSPs for multimedia applications, in: IEEE Ist Workshop on Multimedia Signal Processing, 1997, pp. 343–439.

[27] D.M. Barbosa et al., Parallelizing MPEG video encoding using multiprocessors, in: Proceedings of XII Brazilian Symposium on Computer Graphics and Image Processing, 1999, pp. 215–222.
[28] K. Diefendorff, P.K. Dubey, R. Hochsprung, H. Scale, AltiVec extension to PowerPC accelerates media processing, IEEE Micro 20 (2) (2000) 85–95.
[29] A.C. Downton, Speed-up trend analysis for H.261 and model-based image coding algorithms using a parallel-pipeline model, Signal Processing: Image Communication 7 (4–6) (1995) 489–502.
[30] S. Dutta, W. Wolf, A flexible parallel architecture adapted to block-matching motion-estimation algorithms, IEEE Transactions on Circuits and Systems for Video Technology 6 (1) (1996) 74–86.
[31] T. Ebrahimi, M. Kunt, Visual data compression for multimedia applications, Proceedings of the IEEE 86 (6) (1998) 1109–1125.
[32] T. Nozawa et al., A parallel vector-quantization processor eliminating redundant calculations for real-time motion picture compression, IEEE Journal of Solid-State Circuits 35 (11) (2000) 1744–1750.
[33] G. Frantz, Digital signal processor trends, IEEE Micro (2000) 52–59.
[34] J. Fridman, Sub-word parallelism in digital signal processing, IEEE Signal Processing Magazine 17 (2) (2000) 27–35.
[35] E.D. Frimout, I.N. Driessen, E.F. Deprettere, Parallel architecture for a pel-recursive motion estimation algorithm, IEEE Transactions on Circuits and Systems for Video Technology 2 (2) (1992) 159–168.
[36] D. Le Gall, MPEG: a video compression standard for multimedia applications, Communications of the ACM 34 (4) (1991) 46–58.
[37] A. Geist et al., PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Networked Parallel Computing, MIT Press, Cambridge, 1994.
[38] D. Gong, Y. He, Three goals in designing VLSI architectures for video compression, in: APCC/OEC'99, vol. 2, 1999, pp. 908–911.
[39] D. Gong, Y. He, Computation complexity analysis and VLSI architectures of shape coding for MPEG-4, in: Proceedings of SPIE, Visual Communications and Image Processing 2000, vol. 4067, 2000, pp. 1459–1470.
[40] K.L. Gong, L.A. Rowe, Parallel MPEG-1 video encoding, in: Picture Coding Symposium, California, September 1994.
[41] E. Hagersten, G. Papadopoulos, Parallel computing in the commercial marketplace—research and innovation at work, Proceedings of the IEEE 87 (3) (1999) 405–411.
[42] A. Hamosfakidis, Y. Paker, J. Cosmas, A study of concurrency in MPEG-4 video encoder, in: IEEE International Conference on Multimedia Computing and Systems, 1998, pp. 204–207.
[43] A. Hanami et al., A 165-GOPS motion estimation processor with adaptive dual-array architecture for high quality video-encoding applications, in: Proceedings of the IEEE 1998 Custom Integrated Circuits Conference, 1998, pp. 169–172.
[44] B.G. Haskell et al., Image and video coding—emerging standards and beyond, IEEE Transaction on Circuits and Systems for Video Technology 8 (7) (1998) 814–837.
[45] B.G. Haskell, A. Puri, A.N. Netravali, Digital Video: An Introduction to MPEG-2, Chapman and Hall, New York, 1997.
[46] Y. He, I. Ahmad, M.L. Liou, Real-time interactive MPEG-4 system encoder using a cluster of workstations, IEEE Transactions on Multimedia 12 (1999) 217–233.
[47] Y. He, I. Ahmad, M.L. Liou, A software-based MPEG-4 video encoder using parallel processing, IEEE Transaction on Circuits and Systems for Video Technology 8 (7) (1998) 909–920.
[48] C.H. Hsieh, T.P. Lin, VLSI architecture for block-matching motion estimation algorithm, IEEE Transactions on Circuits and Systems for Video Technology 2 (2) (1992) 169–175.
[49] K. Hwang, Z. Xu, Scalable Parallel Computing, Technology, Architecture Programming, WCB/McGraw-Hill, 1997.
[50] Intel orporation, Coarse-grain multithreading, Intel Application Note AP-802, Order no: 243636-002.
[51] H.-M. Jong, L. Gee, T.-D. Chiueh, Parallel architectures for 3-step hierarchical search block-matching algorithm, IEEE Transactions on Circuits and Systems for Video Technology 4 (4) (1994) 407–416.

[52] ITU-T experts group on very low bitrate visual telephony, ITU-T Recommendation H.263: Video coding for low bitrate communication, December 1995.
[53] ITU-T experts group on very bitrate visual telephony, ITU-T Recommendation H.263 version 2: Video coding for low bitrate communication, January 1998.
[54] D. Kim et al., A real-time MPEG encoder using a programmable processor, IEEE Transactions on Consumer Electronics 40 (2) (1994) 161–170.
[55] I.-K. Kim, J.-J. Cha, H.-J. Cho, A design of 2-D DCT/IDCT for real-time video applications, in: 1999 International Conference on VLSI and CAD, 1999, pp. 557–559.
[56] P. Kolinummi et al., Scalable implementation of H.263 video encoder on a parallel DSP system, in: IEEE International Symposium on Circuits and Systems, vol. 1, 2000, pp. 551–554.
[57] K. Konstantinides, VLIW architecture for media processing, IEEE Signal Processing Magazine 152 (1998) 16–19.
[58] P. Kuhn et al., A flexible low-power VLSI architecture for MPEG-4 motion estimation, in: SPIE Conference on Visual Communications and Image Processing '99, vol. 3653, San Jose, CA, USA, January 1999, pp. 883–894.
[59] P. Kuhn, Algorithms, complexity analysis and VLSI architectures for MPEG-4 motion estimation, Kluwer Academic Publishers, 1999.
[60] I. Kuroda, T. Nishitani, Multimedia processors, Proceedings of the IEEE 866 (1998) 1203–1221.
[61] D.-K. Yeung, A scalable MPEG-2 encoding system using the multithread architecture, M.Phil. Thesis, Hong Kong University of Science and Technology, September 2000.
[62] B. Lee, A new algorithm to compute the discrete cosine transform, IEEE Transactions on Signal Processing (1984) 1243–1245.
[63] C. Lee, T. Yang, Y.F. Wang, Partitioning and scheduling for parallel image processing operations, in: 7th IEEE Symposium on Parallel and Distributed Processing, 1995, pp. 86–90.
[64] R.B. Lee, Subword parallelism with MAX-2, IEEE Signal Processing Magazine 17 (2) (2000) 27–35.
[65] K.-K. Leung et al., Parallelization methodology for video coding—an implementation on TMS320C80, IEEE Transaction on Circuits and Systems for Video Technology 10 (8) (2000) 1413–1425.
[66] W. Lin, B. Tye, E. Ong, C. Xiong, T. Miki, S. Hotani, Systematic analysis and methodology of real-time DSP implementation for hybrid video coding, in: 1999 International Conference on Image Processing, vol. 3, 1999, pp. 847–851.
[67] W. Lin et al., Real time H.263 video codec using parallel DSP, in: International Conference on Image Processing, vol. 2, 1997, pp. 586–589.
[68] N. Ling, R. Advani, Architecture of a fast motion estimator for MPEG video coding, in: Second Symposium on Parallel Architectures, Algorithms, and Networks, 1996, pp. 473–479.
[69] M.L. Liou, Overview of the p×64 kbps video coding standard, Communications of the ACM 34 (4) (1991) 59–63.
[70] M.L. Liou, H. Fujiwara, VLSI implementation of a low bit-rate video codec, in: IEEE International Symposium on Circuits and Systems, 1991, pp. 180–183.
[71] J.N. Mailhot, H. Derovanessian, The grand alliance HDTV video encoder, IEEE Transactions on Consumer Electronics 41 (4) (1995) 1014–1019.
[72] F. Marino et al., A parallel implementation of the 2-D discrete wavelet transform without interprocessor communications, IEEE Transactions on Signal Processing 47 (11) (1999) 3179–3184.
[73] M. Mattavelli, S. Brunetton, D. Mlynek, A parallel multimedia processor for macroblock based compression standards, in: International Conference on Image Processing, vol. 2, 1997, pp. 570–573.
[74] D.J. Mlynek, J. Kowalczuk, VLSI for digital television, Proceedings of the IEEE 83 (7) (1995) 1055–1070.
[75] MPEG-l video group, Information technology-coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s: Part 2-Video, ISO/lEC 11172-2, International Standard, 1993.
[76] MPEG-2 video group, Information technology-generic coding of moving pictures and associated audio: Part 2-Video, ISO/lEC 13818-2, International Standard, 1995.

[77] MPEG-4 video group, Generic coding of audio-visual objects: Part 2-Visual, ISO/IEC JTCI/SC29/WG11 N1902, FDIS of ISO/IEC 14496-2, Atlantic City, November 1998.
[78] K. Nakamura, M. Ikeda, T. Yoshitome, T. Ogura, Global rate control scheme for MPEG-2 HDTV parallel encoding system, in: International Conference on Information Technology: Coding and Computing, 2000, pp. 195–200.
[79] J. Nang et al., An effective parallelizing scheme of MPEG-1 video encoding on ethernet-connected workstations, in: Proceedings on Advances in Parallel and Distributed Computing, 1997, pp. 4–11.
[80] H. Nicolas, F. Jordan, Interactive optimization and massively parallel implementations of video compression algorithms, in: International Conference on Image Processing, vol. 1, 1996, pp. 229–232.
[81] T. Nishitani, I. Tamitani, H. Harasaki, Programmable parallel processor for video processing, in: IEEE International Conference on Systems Engineering, 1989, pp. 169–172.
[82] T. Olivares, P. Cuenca, F.J. Quiles, A. Garrido, Parallelization of the MPEG coding algorithm over a multicomputer. A proposal to evaluate its interconnection network, in: 1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM, 10 Years Networking the Pacific Rim, 1987–1997, Part vol. 1, IEEE, New York, NY, USA, 1997, pp.113–116.
[83] S. Nogaki, A study on rate control method for MP@HL encoder with parallel encoder architecture using picture partitioning, in: 1999 International Conference on Image Processing, vol. 4, 1999, pp. 261–265.
[84] A. Peleg, U. Weiser, MMX technology extension to the intel architecture, IEEE Micro 16 (4) (1996) 42–50.
[85] G.F. Pfister, In search of clusters: The Ongoing Battle in Lowly Parallel Computing, second ed., Prentice Hall, Englewood Cliffs, NJ, 1998.
[86] A. Pirson, A Programmable Motion Estimation Processor for Full Search Block Matching, 1995 International Conference on Acoustics, Speech, and Signal Processing, vol. 5, 1995, pp. 3283–3286.
[87] P. Pirsch, H.J. Stolberg, VLSI implementations of image and video multimedia processing systems, IEEE Transactions on Circuits and Systems for Video Technology 8 (7) (1998) 878–891.
[88] P. Pirsch et al., VLSI architectures for video compression—a survey, Proceedings of the IEEE 83 (2) (1995) 1055–1070.
[89] A.M. Rassau, G. Alagoda, K. Eshraghian, Massively parallel wavelet based video codec for an intelligent-pixel multimedia communicator, in: Proceedings of the Fifth International Symposium on Signal Processing and Its Applications, vol. 2, 1999, pp. 793–795.
[90] A. Saha, R. Neogi, Parallel programmable algorithm and architecture for real-time motion estimation of various video applications, IEEE Transactions on Consumer Electronics 41 (4) (1995) 1069–1079.
[91] V. Sankar, S. Srinivasan, S.R. Rangarajan, Efficient implementation of MPEG video codec using dual processors, in: International Conference Digital Signal Processing, vol. 2, 1997, pp. 801–804.
[92] K. Shen, G.W. Cook, L.H. Jamieson, E.J. Delp, An overview of parallel processing approaches to image compression, in: Proceedings of the SPIE Conference on Image and Video Compression, vol. 2186, San Jose, California, February 1994, pp. 197–208.
[93] K. Shen, E.J. Delp, A parallel implementation of an MPEG encoder: Faster than real-time, in: Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies, San Jose, CA, 5–10 February, 1995, pp. 407–418.
[94] K. Shen, E.J. Delp, A spatial-temporal parallel approach for real-time MPEG video compression, in: Proceedings of the 1996 International Conference on Parallel Processing, vol. 2, 1996, pp. 100–107.
[95] T. Sikora, MPEG digital video-coding standards, IEEE Signal Processing Magazine (1997) 82–100.
[96] T. Sikora, L. Chiariglione, MPEG-4 video and its potential for future multimedia services, in: Proceedings of 1997 IEEE International Symposium on Circuits and Systems, vol. 2, 1997, pp. 1468–1471.
[97] M.T. Sun, T.C. Chen, A.M. Gottlieb, VLSI implementation of a 16×16 discrete cosine transform, IEEE Transactions on Circuits and Systems 36 (4) (1989) 610–617.
[98] I. Tamitani, A. Yoshida, Y. Ooi, T. Nishitani, A SIMD DSP for real-time MPEG video encoding and decoding, in: Workshop on VLSI Signal Processing, 1992, pp. 119–128.

[99] J. Tanskanen, J. Niittylahti, Parallel memories in video encoding, in: Data Compression Conference, 1999, p. 552.

[100] L. Thuyen, M. Glesner, Configurable VLSI-architectures for both standard DCT and shape-adaptive DCT in future MPEG-4 circuit implementations, in: The 2000 IEEE International Symposium on Circuits and Systems, vol. 2, 2000, pp. 461–464.

[101] P. Tiwari, E. Viscito, A parallel MPEG-2 video encoder with look-ahead rate control, in: IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 4, 1996, pp. 1994–1997.

[102] M. Tremblay, VIS speeds new media processing, IEEE Micro 164 (1996) 10–20.

[103] B. Tye, DSP implementation of very low bit rate videoconferencing system, in: Proceedings of 1997 International Conference on Information, Communications and Signal Processing, vol. 3, 1997, pp. 1275–1278.

[104] D.W. Walker, J.J. Dongarra, MPI: a standard message passing interface, Supercomputer 12 (1) (1996) 56–68.

[105] S. Winograd, On computing the discrete fourier transform, IBM Research Report, 1976, RC-6291.

[106] H. Yamauchi et al., Architecture and implementation of a highly parallel single-chip video DSP, IEEE Transactions on Circuits and Systems for Video Technology 2 (2) (1992) 207–220.

[107] D.-K. Yeung, A scalable MPEG-2 encoding system using the multithread architecture, M.Phil. Thesis, Hong Kong University of Science and Technology, September 2000.

[108] N.H.C. Yung, K.C. Chu, Fast and parallel video encoding by workload balancing, in: 1998 IEEE International Conference on Systems Man and Cybernetics, vol. 5, 1998, pp. 4642–4647.

[109] B. Zeng, R. Li, M.L. Liou, Optimization of fast block motion estimation algorithms, IEEE Transaction on Circuit and Systems for Video Technology 7 (6) (1997) 833–844.

[110] L. De Vos, M. Schobinger, VLSI architecture for a flexible block matching processor, IEEE Transaction on Circuits and Systems for Video Technology 5 (5) (1995) 417–428.

[111] A.A.J. Roach, A. Moini, VLSI archtietcure for motion estimation on a single-chip video camera, in: Proceedings of SPIE, vol. 4067, 2000, pp. 1441–1450.

[112] K.S. Prashant, V.J. Mathews, A massively parallel algorithm for vector quantization, in: Proceedings of Data Compression Conference, 1995, pp. 499.

[113] K. Kobayashi et al., A functional memory type parallel processor for vector quantization, in: Proceedings of the Asia and South Pacific Design Automation Conference, 1997, pp. 665–666.

[114] S.H. Nam, M.K. Lee, Flexible VLSI architecture of motion estimation for video image compression, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 43 (6) (1996) 467–470.

[115] Z.-L. He, M.L. Liou, Design of fast motion estimation algorithm based on hardware consideration, IEEE Transaction on Circuits and Systems for Video Technology 7 (5) (1997) 819–823.

[116] Z.-L. He, C.-Y. Tsui, K.-K. Chan, M.L. Liou, Low-power VLSI design for motion estimation using adaptive pixel truncation, IEEE Transaction on Circuits and Systems for Video Technology 10 (5) (2000) 669–677.

[117] N.H.C. Yung et al., Spatial and temporal data parallelization of the H.261 video coding algorithm, IEEE Transaction on Circuits and Systems on Video Technology 11 (1) (2001) 91–104.

[118] O. Lehtoranta, T. Hamalainen, J. Saarinen, Parallel implementation of H.263 encoder for CIF-sized images on quad DSP system, in: IEEE International Symposium on Circuits and Systems, ISCAS2001, vol. 2, 2001, pp. 209–212.