Parallel Approaches to Real-Time MPEG Video Compression

Ke Shen and Edward J. Delp
Purdue University
School of Electrical Engineering
Video and Image Processing Laboratory
Purdue Multimedia Testbed
West Lafayette, Indiana 47907-1285

This work was supported in part by the Advanced Research Projects Agency under contract DABT63-92-C-0022 and by a grant from the AT&T Foundation. The content of the information does not necessarily reflect the position or policy of the United States Government and no official endorsement should be inferred.
Address all correspondence to E. J. Delp, ace@ecn.purdue.edu, http://dynamo.ecn.purdue.edu/∼ace, or +1 765 494 1740.

Parallel Approaches to Real-Time MPEG Video Compression

Corresponding Author:

Professor Edward J. Delp

Purdue University

School of Electrical Engineering

1285 Electrical Engineering Building

West Lafayette, IN 47907-1285

Telephone: +1 765 494-1740

Fax: +1 765 494-0880

E-mail: ace@ecn.purdue.edu

Abstract

In this paper we present several parallel implementations of an MPEG1 video encoder on a multiple instruction streams multiple data streams (MIMD), distributed memory supercomputer. Since the MPEG compression algorithm is frame/block based, video data can be distributed to the processors frame-wise (*temporal parallelism*) or block-wise (*spatial parallelism*) without changing the overall computation complexity of the algorithm. In our approaches, both *spatial* and *temporal* parallelism have been exploited. For a supercomputer, such as the Intel Paragon, which is optimized for scientific computation, it is difficult to maintain a huge data throughput which is required for real-time video compression. While the Paragon has the computation capacity to compress video sequences at a speed faster than real-time, we found that real-time performance cannot be achieved if the Input/Output (I/O) techniques are not designed properly. We present several schemes, corresponding to different types of data parallelism, for managing the I/O operations and regulating the data flow. We show their effect on increasing the overall speed of video compression. Using our algorithm, real-time MPEG compression of CCIR 601 digital video sequences was achieved.

List of Symbols: None

List of Figures

1. Introduction

Video compression plays an important role in many digital video applications, such as digital libraries, video on demand (VOD), and high definition television (HDTV). The main objective of a video compression algorithm is to exploit both the spatial and temporal redundancy of a video sequence such that fewer bits can be used to represent a video sequence at an acceptable visual quality level. An NTSC digital video sequence which conforms to the ITU-R (CCIR) Recommendation 601 (720×486 pixels per luminance frame, 4:2:2 chrominance subsampling, 8 bits per pixel per color) has an uncompressed data rate of 168 mega-bits per second (Mbps) [4, 13, 25]. A typical 2-hour movie would occupy approximately 150 gigabytes of disk space. This data rate would easily overwhelm most communication channels or storage systems. One immediately sees the need for compression. Many compression standards, such as ITU-T (CCITT) Recommendation H.261, ISO standards MPEG1 and MPEG2 [5, 11, 12, 20, 25], have been proposed to address this problem. The MPEG standards have drawn a lot of attention because the US HDTV standard will adopt MPEG2 as its compression algorithm [9].

The time needed to compress a video sequence is an important issue. Many applications, such as live digital television broadcasting, require real-time video compression. Other applications, such as the creation of a digital video library, require very fast or real-time compression. The time needed to compress a video sequence is affected by the computation complexity of the algorithm, the efficiency in the implementation of the algorithm and the speed of the computation facilities.

The computation complexity of the algorithm, along with the compressed data rate, and the visual quality of the decoded video, are the three major factors used to evaluate a video compression algorithm. An ideal algorithm should have a low computation complexity, a low compressed data rate and a high visual quality for the decoded video. However, these three factors usually cannot be achieved simultaneously. For an algorithm that generates compressed video with high quality and low data rate, the computation cost is usually high. It is highly likely that a compressed video sequence with a lower data rate will have a poorer visual quality after being decompressed. Among these three factors, the computation

complexity directly affects the time needed to compress a video sequence.

The MPEG standard defines the syntax of the bit stream of a compressed video sequence, i.e. the decoder is defined and the implementation of the encoder is open to individual designs. Thus, there is not a "standard" algorithm for MPEG video compression. Since block based motion compensated prediction [14, 15, 18] is used in the MPEG standard, the encoding algorithms have to search for the motion vectors, which is a computationally intensive task. To encode a CCIR 601 video sequence in real-time, the amount of computation required for the search of the motion vectors is on the order of 10^9 operations per second (Gflops) if a full search algorithm is used [22, 24]. Although many fast algorithms have been proposed [14, 16, 19], the amount of computation is still overwhelming. Thus the MPEG algorithm is notoriously asymmetric, i.e. the amount of computation required for the encoding algorithm is much higher than that required for the decoding algorithm. Thus, real-time video compression is difficult to achieve and is very expensive especially when the image size is large, or high image quality and low data rates are required. Therefore, parallel processing becomes a natural choice to meet the large computation requirement for real-time video compression.

Parallel video compression algorithms can be implemented using either hardware or software approaches [21]. In hardware approaches special architectures, specifically parallel architectures, can be designed to accelerate the computation [2, 3, 8]. Such a video compression device is feasible for applications such as live TV broadcasting. These devices are very expensive, e.g. a real-time MPEG2 encoder for CCIR 601 video costs in excess of $100,000. In many cases only a single video stream can be processed at a time. Software approaches include the use of parallel supercomputers and networks of workstations [1, 6, 7, 17, 22, 24, 26]. Usually a software implementation offers more flexibility, e.g. parameters can be adjusted easily and multiple video streams can be handled, which is essential for applications such as a video server or a video database. Portability may also be an advantage of a software implementation. A software approach is most suitable for algorithm development, digital library servers, or massive production of compressed digital video.

A video sequence can be viewed as a 3-dimensional (3-D) signal with two dimensions in the spatial domain and one in the temporal domain. Since many video compression

algorithms are frame/block based, video data can be distributed to the processors frame-wise (*temporal parallelism*) or block-wise (*spatial parallelism*) without changing the overall computation complexity of the algorithm significantly. In spatial parallelism [1, 6, 17, 24, 26], each frame of a video sequence is divided into several parts containing an integer number of blocks and each part is encoded by a processing element (PE). The whole video sequence is processed in a frame-by-frame fashion, which results in a minimum delay. However, there is an upper limit on the number of PEs that can be used because of the limited spatial resolution of a video sequence [1]. Also a massive spatial parallel algorithm usually needs to tolerate a relatively high communication overhead. In temporal parallelism, different PEs are assigned to process different video frames of a video sequence [7, 22]. There is no upper limit on how many PEs can be used since a typical 2 hour motion picture contains more than 200,000 frames. A problem with this approach is that even though the overall performance (throughput) can be faster than real-time the compression of a single frame may be slower than real-time, which results in a constant delay.

A variation of the above approaches, which combines the temporal and spatial parallelism, has been proposed [23]. In this paper, we describe the spatial-temporal parallel approach in detail. New experimental results and an analysis of the execution time of each task module of the algorithm are presented. A comparison of this method with other parallel approaches are also given.

The rest of the paper is organized as follows. In section 2 and section 3, we overview the MPEG standard and the Intel Paragon, respectively. In section 4, we discuss the problem and the difficulties in achieving real-time performance. In section 5, our parallel approaches are described. The experimental results are presented in section 6. Discussion and conclusions are given in section 7 and section 8.

2. An Overview of MPEG1

MPEG1 is an international standard for digital video compression developed by the Moving Picture Experts Group [11, 20, 25]. MPEG1 divides the compressed video frames into three types: I pictures, P pictures and B pictures (Figure 1). An I picture, or *intracoded picture*,

is encoded without reference to other frames, exploiting only the spatial correlation within the frame. A P picture (*predictive coded picture*) is encoded using motion compensated prediction from a past I picture or P picture. A *bidirectionally-predictive coded picture*, or B picture, requires both past and future reference frames (I pictures or P pictures) for motion compensation. Since I and P pictures are used as reference pictures, it is necessary to maintain a relatively high video quality for these pictures. On the contrary B pictures which are never used as reference pictures can tolerate a relatively poorer video quality. Thus the quantization steps for I and P pictures are set to be smaller, which correspond to higher compressed data rates, than those for B pictures. Also, since the motion compensated prediction is used for P and B pictures, the data needed to be coded in P and B pictures have lower variances, which correspond to lower compressed data rates, than I pictures. Therefore, usually I pictures have the highest data rate and the lowest motion artifacts, while B pictures have the lowest data rate and the highest motion artifacts. The typical data rate of an I picture is 1 bit per pixel while that of a P picture is 0.5 bits per pixel and for a B picture, 0.25 bits per pixel. Obviously the use of P pictures and B pictures yields a much lower data rate. However, it takes much longer time to compress a P or B picture because the encoding algorithm need to search for the motion vectors.

An MPEG1 encoder needs to specify which frames of the input video sequence are to be compressed as an I, P, or B picture. This pattern of I, P and B pictures may (or may not) be repeated throughout the video sequence. An example of a frame pattern is IBBPBBPBBIBBP$\cdots$, where I, P, and B indicate an I picture, a P picture and a B picture, respectively. In an MPEG encoded sequence, several consecutive frames are combined to form a structure known as a group of pictures (GOP). While a GOP can contain one or more I pictures, the first picture in a GOP must be an I picture.

The image color space used in the MPEG1 is the YC_rC_b space, in which Y represents the luminance component and C_r and C_b represent the chrominance components, or the color difference components [11, 13, 20]. For an original video frame, each spatial position (pixel) is represented by a Y component, a C_r component and a C_b component. Thus, a video frame consists of 3 images of the same size, each of which corresponds to the Y, the C_r or the C_b component. This is known as the 4:4:4 format [20]. In MPEG1, the

chrominance components are subsampled with respect to the luminance component by half in both vertical and horizontal directions, known as the 4:2:0 format. A video frame consists of non-overlapping blocks of pixels, or macro-blocks, each of which contains 16×16 pixels of the luminance image, 8×8 pixels of the C_r chrominance image and 8×8 pixels of the C_b chrominance image. An 8×8 pixel array is known as a block. Hence a macro-block contains 4 luminance blocks and 2 chrominance blocks (one C_r block and one C_b block). A video frame is divided into slices, each of which consists of a series of an arbitrary number of macro-blocks in raster-scan order from left to right and top to bottom in a frame. Slices in a frame are non-overlapping and covering all the macro-blocks.

In the compression of I pictures, each block of the picture is discrete cosine transformed (DCT) and the DCT coefficients are quantized. Different quantizer step sizes, set by the quantizer scale, can be used to control the data rate. Differential pulse coding modulation (DPCM), run length coding and variable length coding (VLC) are then performed on the quantized DCT coefficients. In the compression of P pictures and B pictures, motion compensated prediction is used. Motion associated to a macro-block in a P or B picture is represented by a two-dimensional vector, known as motion vector, which specifies where to retrieve the macro-block from the reference frame. Integer and half pixel displacements are allowed in motion vectors . The motion vector associated to a macro-block is obtained by matching the macro-block with pixel arrays in the reference frame within the spatial search range. The displacement between the macro-block and the best matched pixel array in the reference frame is used as the motion vector. Many different strategies can be used to search for motion vectors, including full search and logarithmic search [11, 14, 18]. If full search, or exhaustive search, is used, all macro-block sized pixel arrays within the chosen spatial search range need to be evaluated, which makes the computation very intensive. Other methods, such as logarithmic search [14], use strategies to control the search process so that not all of the possible displacements are evaluated and the amount of computation is reduced. The motion vectors are then encoded using DPCM and VLC. A predicted picture is formed by replicating the pixels from the reference frame at new locations according to the motion vectors. The difference picture (error picture) between the current picture and the predicted picture is encoded using the same DCT-based techniques used for the I pictures. If a motion

vector cannot be found according to the matching criterion, the macro-block will be coded as an intra-coded macro-block (similar to the macro-blocks in an I picture).

3. An Overview of the Intel Paragon

While our approach is very general and could be extended to any MIMD distributed memory architecture, in our experiments, we used the Intel Paragon as the implementation platform. The Intel Paragon XP/S supercomputer is a typical distributed memory, multiple instruction streams multiple data streams (MIMD) parallel supercomputer [10]. It uses i860 XP processors as its Numerical Nodes which operate at 50 MHz and have peak floating point performance of 75 Mflops double-precision or 100 Mflops single-precision. Each Numerical Node is attached to a mesh routing chips (MRC). The MRCs are interconnected via a network with a two-dimensional mesh topology. Since these nodes do not share memory, they must exchange messages through the network to share information. The mesh is connected to the Parallel File System (PFS), which distributes file blocks to all available disks using algorithms for reading and writing that allow several PEs to use the file systems simultaneously. The number of Numerical Nodes and the number and size of PFSs in a Paragon computer vary relative to the configuration. The Paragon computers we used for our experiments are the system located at the Concurrent Supercomputing Consortium which has 512 Numerical Nodes and the one located at Purdue University which has 140 Numeric Nodes. The total computation capacities exceed 38 Gflops and 10 Gflops, respectively. Both systems have multiple PFSs.

4. Problem Description

Several approaches have been proposed to map the MPEG algorithm on to the Intel Paragon [1, 22, 23]. The main objective of the work has been to develop parallel implementations that provide real-time throughput. In our experiments, we found that a single node on the Paragon can compress a CCIR 601 video sequence at a speed of 0.09 frames per second with 95.4% of the time used for computation. Since the parallel algorithm does not reduce

the overall amount of computation required, to achieve real-time performance 320 nodes are needed if 100% of the time is used for computation. Thus it is critical to minimize the time spent on tasks other than computation, such as I/O, communications and data distribution/assemblage, in the implementation. For a supercomputer, such as the Intel Paragon, which is optimized for scientific computation, the problem can be difficult when a huge data throughput needs to be maintained. Recent work has shown that the performance of the I/O system can greatly affect the performance of the whole algorithm in parallel image/video processing where the data set is huge [1, 6, 22]. For the 512-node Paragon used in our experiments, the goal of our work can be reiterated as follows: develop a parallel algorithm that, on average, less than $1 - 320/512 = 37.5\%$ of the time is used for I/O, communications and data distribution/assemblage.

A video sequence can be divided into parallel data streams spatially, temporally, or spatial-temporally. In [1], strict spatial parallelism was used and real-time performance on the compression of CIF (352×288 pixels per luminance frame) video sequences, which is a quarter of the image size of CCIR 601 sequences [20], was achieved. A maximum compression speed of 31.14 frames per second was obtained using 330 processors on an Intel Paragon. However, in their computation of compression speed (execution time), the time needed for the I/O operations was not taken into account. When I/O operations are included, the algorithm will not obtain real-time performance.

In [22], we used temporal parallelism to achieve faster than real time compression of CIF video sequences using the Intel Touchstone Delta and Intel Paragon. The implementation was based on a greatly modified version of the software MPEG encoder developed at the University of California at Berkeley [7]. We assumed that the video sequences are stored on the PFS of the parallel computer. The algorithm reads a video stream from the PFS, compresses the sequence and then writes the compressed data to the PFS. The end-to-end execution time (including I/O time) is used to compute the compression speed. I/O contention was found to be the main bottleneck because a large number of PEs needed to read different frames from the PFS simultaneously, which resulted in I/O congestion. An I/O management algorithm was used to minimize the I/O contention. Frame rates of over 41 frames per second were achieved using 100 processors on the Touchstone Delta or 144

processors on the Paragon. However, for CCIR 601 images (720×486 pixels per luminance frame) the compression speed is less than 30 frames per second using the temporal parallel algorithm. The reason is that a CCIR 601 frame is almost 4 times the size of a CIF sized frame, which increases both the computation and the I/O time.

In [23], a spatial-temporal parallelism was introduced. The number of PEs that need to perform I/O operations at the same time is reduced because more than one of the PEs share the same I/O operations. The I/O contention is also reduced.

The efficiency of these parallel algorithms is determined by how the algorithm can reduce the overhead, including the I/O operations, communications and data distribution/assemblage. In the following two sections, we compare the performance of the algorithms using temporal parallelism and spatial-temporal parallelism.

5. Parallel Implementation of the Encoder

In this section, we describe several parallel approaches to MPEG video compression. For the purpose of clarity, we describe the temporal parallel algorithm first. The spatial-temporal parallel algorithm is then described in detail.

5.1 Temporal Parallel Algorithm

A block diagram of the data flow in the temporal parallel algorithm is shown in Figure 2. The input video sequence is divided into sections, each of which contains the video frames to be compressed and the necessary reference frames. These sections are stored on the PFS as individual files. This arrangement is used to reduce the number of *read* operations, hence the I/O contention. Each PE is assigned a section of the video sequence at one time. The assigned section is read and then compressed by the PE. The compressed data is saved on the PFS and the PE is then assigned to compress another video section. An I/O queue is used to reduce I/O contention. At any given time only a certain number of PEs can read from the PFS. Other PEs have to wait in the queue and gain access to the I/O on a first-come first-serve basis. The compressed data of each section resides on the PFS as an individual file. A better solution seemed to be the use of a PE to assemble the compressed

data from different PEs into a single data stream before saving the data to the PFS. However, experiments showed that a single PE cannot assemble and save the data fast enough to fulfill the real-time video compression requirement [22].

5.2 Spatial-Temporal Parallel Algorithm

The spatial-temporal parallel algorithm is based on the temporal parallel algorithm. The compression of each video section is conducted by a group of PEs in a spatial parallel mode, i.e. each frame in the video section is spatially divided into pieces that are compressed in parallel by the PEs in the group. More specifically, all of the PEs are evenly divided into groups, each of which contains n PEs. Among the n PEs in each group, m PEs ($m \leq n$) are used for computation (denoted as *computation PEs*). The rest of the PEs are reserved for special purposes. One of the PEs in a group reads a video section and sends the data to those m computation PEs in the same group through message passing. Each frame in the video section is spatially divided into m slices which are processed by the m computation PEs in parallel. Since each PE on the Paragon has sufficient memory, even though a computation PE processes only a part of each frame, the entire video section is sent to all of the computation PEs. Therefore, we do not have to divide a video section into frames or a frame into several parts and send different parts to different PEs using multiple *send* and *receive* operations. Thus, the communication overhead is reduced. According to the number of PEs in a group and the size of the images, each PE is able to determine the portions of data in the entire video section that need to be compressed locally. The compressed data from each computation PE is then sent to one of the PEs in that group, where the compressed data is assembled and written to the disk (denoted as the output PE).

The data flow in each group of PEs along with the I/O and the communication operations of the algorithm are summarized as follows. A single video section is read by one of the PEs in a group using one *read* operation. Then the video section is sent to all the m computation PEs in the group using one *send* operation. The video section is compressed frame by frame, with each computation PE processing only a part of each frame. The compressed data of each frame is then sent to the output PE. Suppose there are i frames in the video section,

then $i \times m$ *send* operations are used by the computation PEs and $i \times m$ *receive* operations are used by the output PE. The output PE assembles the compressed data and writes the compressed data of the entire video section using one *write* operation. Thus, a minimum number of I/O and communications operations are used.

We used two different schemes in our experiments to realize the spatial parallelism.

Scheme 1

A block diagram of the data flow in this scheme is shown in Figure 3. In each group of n PEs, one PE is devoted to I/O operations, which is denoted as the I/O node. The rest of the PEs are used for computation. Thus, $m = n - 1$. The I/O node reads a section of video data from the disk, distributes the data to the computation PEs in the group. Each computation PE compresses a part of each of the frames in the video section and send the compressed data back to the I/O node. The I/O node assembles the compressed data and writes it to the disk. A timing diagram of the task modules for each PE in a group is shown in Figure 4. In the diagram we assume that $n = 4$ and there are 3 frames in each video section. To minimize the idle time of the computation PEs, the I/O node reads the next video section from the disk before the I/O node starts collecting compressed data for the current section, i.e. while the computation PEs are compressing the current section. For the same reason, it distributes the video data for the next section to the computation PEs before it writes the compressed data for the current section to the disk.

Scheme 2

A block diagram of the data flow in this scheme is shown in Figure 5. In this scheme, no PEs are devoted to I/O operations. Thus, $m = n$. All the PEs conduct the compression task. Meanwhile one of the PEs is in charge of the data input operations and another one is in charge of the data output operations.

The PE that is in charge of the data input operations reads the video data from the disk and distributes it to other PEs in the group. All the PEs in the group participate in computation, with each PE processing a part of each frame in parallel. The compressed data is sent to the PE that is in charge of the data output operations, where the data is assembled and written to the disk. A timing diagram of the task modules for each PE in a group is shown in Figure 6.

From the timing diagrams, we can see that both Scheme 1 and Scheme 2 have advantages and disadvantages. For Scheme 1, the computation PEs do not have to wait for the video data to be read in. Therefore it is more efficient for the computation PEs. However, since one PE in each group is dedicated to I/O operations, the total number of PEs involved in the compression is reduced. For Scheme 2, each PE is involved in computation. However, when two of the PEs are conducting input and output operations the rest of the PEs in the group have to wait. Hence, for these two schemes there is a trade off between the number of PEs involved in the computation and the amount of idle time for the computation PEs.

6. Results

The video sequences used in our experiments are several CCIR 601 (704×480 pixels per luminance frame) test sequences. From experimentation, we found that the performance of the MPEG compression algorithm is almost invariant when different data sets are used. Thus, in this paper we only present the results obtained from one MPEG test sequence, known as the *football* sequence. The original sequence has 150 frames, subsampled to the 4:2:0 format in the YUV color space [13, 20]. We extended the sequence to more than 10,000 frames by repeating the original sequence. The video sequence was grouped into sections (GOPs), each of which contained 12 frames to be compressed and a reference frame. A frame pattern of IBBPBBIBBPBB was used. In each of our experiments, 4–6 sections were compressed by each group of PEs. The average speed of a group is defined as the number of frames compressed by the group divided by the overall execution time of the group, including the I/O time, the computation time and the communications time. The

overall speed was obtained by summing the average speeds of all the groups. The motion vector search algorithm used is the logarithmic algorithm and produces integer pixel motion vectors.

The overall performance of the temporal only parallel algorithm in terms of compression speed is shown in Figure 7. We can see that the algorithm has near linear speedup when the number of processors involved are less than 380. When the number of PEs exceeds 380, increasing the number of PEs does not result in any more speedup. The reason is that the I/O utilization is so close to the system's I/O bandwidth limit that the increase in the number of processors only increases I/O contention. This effect can be seen from Figure 8 and Figure 9. In Figure 8, the percentage of time used by a certain module is defined as the ratio between the summation of the time used for this module on all the processors and the summation of the total running time on all the processors. In Figure 9, the virtual number of processors used for a certain module is defined as the percentage of running time used for this module multiplied by the total number of processors. This number indicates how many processors are required for a certain module if 100% of the running time of these processors are devoted to this module.

When the spatial-temporal parallel algorithms were used, the number of computation PEs m was set to 2, 4, 8, 15 and 30. The overall speed of compression of these two schemes are shown in Figure 10 and Figure 16, respectively. The total time is broken down according to 5 different modules—computation, reading the uncompressed data, distributing the uncompressed data among the processors in a group, collecting the compressed the data, and writing the compressed data to the PFS. The percentages of the running time consumed by these modules in both schemes are shown in Figures 11–15 and Figures 17–21, respectively. Since it is difficult to accurately determine the idle time without introducing significant overhead, the idling mode is not timed individually. Instead, it is included in the time consumed by other modules. For instance, the time used for data input includes the time the I/O node spends in the I/O queue, and the time used for collecting the compressed data includes the time the I/O node spends on waiting for the incoming compressed data.

Several sets of the parameters of the spatial-temporal parallel algorithm with faster-than-real-time performance is shown in Table I. A maximum compression speed of 43 frames per

second was achieved when the Scheme 2 was used and $m = 2$.

Our I/O management methods are so effective that the time used for data input is less than 2% in both schemes (Figure 12 and Figure 18). One major factor that makes the I/O operations in the spatial-temporal parallel algorithms so much more efficient than that in the temporal only parallel algorithm is that the spatial parallelism greatly reduce the number of processors conducting data input operations simultaneously. The maximum number of processors that perform data input operations is 256, when $m = 2$ and 512 nodes are used in Scheme 2. The distribution of uncompressed data among processors within a group, instead of I/O operations, consumes a large proportion of the running time, especially when m is large. When $m = 30$, the time used for distributing the uncompressed data can be as much as 50% to 70% of the total running time. Thus, keeping m small is a very important factor to increase the efficiency of the algorithm. The percentage of time used for data output in both schemes is very small (Figure 13 and Figure 19), which is the reason that the I/O queue is not applied to data output.

One may notice that the percentage of the time used for collecting the compressed data is very high when m is small in Scheme 1 (Figure 15), which contradicts our intuition. This is caused by the fact that when m is small, a small number of PEs compress a video section in parallel, resulting in a longer computation time to compress a single section. Thus the I/O nodes have to wait in idle longer for the incoming compressed data. Since this waiting time is included in the time used for collecting data, we get a high percentage data collecting time when m is small. This is also the reason why $m = 2$ is not the most efficient mode in Scheme 1.

7. Discussion

The advantage of the spatial-temporal parallel algorithm is that a smaller number of processors are allowed to perform I/O operations. Hence it effectively reduces the I/O contention. However one may ask why the spatial parallelism is necessary. Another possible arrangement, the *temporal-temporal* parallelism, can be implemented if we divide the video section read by a group of PEs into smaller sections, each of which contains one or more consecutive

frames, and distribute them to different PEs within the group. It should have the same effect on reducing the I/O contention. The problem is that in the temporal-temporal parallelism it is very difficult to maintain the load balance. Since the time required to compress I, P and B frames are different, the time needed to compress a frame (or a section of frames) may be very different from the time needed for another frame (or a section of frames). In spatial-temporal parallelism, all the PEs in a group compress different spatial locations of the same frame. Thus the load balance within a group of PEs in the spatial-temporal parallel algorithm is much better maintained than that in the temporal-temporal algorithm. One should notice that the load balance among different groups is not as critical as that within a group, because if one group of PEs uses less time to compress a section, a new section will be assigned to it immediately—no efficiency is sacrificed here.

While real-time compression of CCIR 601 video sequences can be achieved, there exists a constant delay—the time used to read, compress and write the first video section. The delay depends on the size of the video section and the number of PEs in a group. This makes it unsuitable for some applications, such as live digital TV broadcasting. However, it is still very useful for applications such as the generation of a digital video library, in which the throughput, instead of the delay, is the most important issue.

Suppose each group of PEs has 2 processors and each video section has 12 frames, the total number of frames that are needed to load the entire machine (512 PEs) is about 3000 frames, or 100 seconds of video. Usually a video sequence is much longer than 100 seconds. Hence it is not a problem to keep the Paragon in full operation. In our experiments, all the uncompressed data of a video sequence resides on the PFS before the compression starts. In practical situations, it is not necessary to have all the data ready before the compression. After the first round of loading, as long as the uncompressed data can be provided to the system at a speed no less than the compression speed, the compression process can be sustained without sacrificing the performance.

In our implementation, special arrangements are made for the input and output algorithms. Instead of being saved in a single file, the compressed data is stored in several files. An index list is generated to indicate the order in which the files should be concatenated. This arrangement can be justified as follows. If the storage media is the parallel file system,

the file headers and the index list can be viewed as an overhead on the compressed data, which is less than 0.01% of the total compressed data. Thus compression ratio is sacrificed a little to achieve a huge gain in the compression speed. If the compressed data is to be stored outside the parallel file system directly and these files need to be concatenated while they are being compressed, a dedicated system with a buffer and a multiplexer can easily handle this problem. Also in our implementation, the input data stream is divided into sections of video sequence instead of frames, which can be done when the video sequence is loaded to the disk. In practical situations, the input stream may be fed to the parallel computer directly from a digitizer or a network instead of the parallel file system. In this case a multiplexer at the interface between the parallel computer and the network can be used to group the video stream into appropriate sections.

In the experimentation, our algorithm was implemented on the Intel Paragon. The implementation utilized the Paragon message passing library. Only the basic parallel file system operations (*read* and *write*) and message passing operations (*send* and *receive*) were used. Thus our implementation can be easily exported to other MIMD computers with distributed memory and parallel file systems.

8. Summary

In this paper we presented several parallel approaches to the implementation of an MPEG encoder on a MIMD supercomputer. We designed the I/O algorithm to reduce I/O contention. The data flow is managed by the proper arrangement of the I/O nodes and the computation nodes within a group of PEs. We showed that without changing either the hardware configuration or the operating system of a supercomputer our approach is able to compress CCIR 601 video sequences in real-time. Thus our results can be viewed as a lower bound on the performance of the parallel MPEG compression algorithm on the Paragon. If the I/O bandwidth of the Paragon can be increased, by changing either the hardware configuration or the operating system, the overall performance can be further improved.

A postscript version of this paper and the commented software package are available via anonymous ftp to **skynet.ecn.purdue.edu** in the directory **/pub/dist/delp/p-mpeg**.

9. REFERENCES

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

[2] V. Bhaskaran and K. Konstantinides. *Image and video compression standards algorithms and architectures*. Kluwer Academic Publishers, Massachusetts, 1995.

[3] C-Cube Microsystems. *C-Cube Microsystems Product Catalog*, Spring, 1994.

[4] CCIR. *Recommendation 601-2: Encoding parameters of digital television for studio*. International Consultative Committee for Radio, 1990.

[5] CCITT. *Recommendation H.261: Video codec for audiovisual services at p*64 kbit/s*. The International Telegraph and Telephone Consultative Committee, 1990.

[6] G. W. Cook and E. J. Delp. An investigation of scalable SIMD I/O techniques with application to parallel JPEG compression. *Journal of Parallel and Distributed Computing*, 30(2):111–128, November 1995.

[7] K. L. Gong. *Parallel MPEG-1 video encoding*. Computer Science Division - Department of EECS, University of California at Berkeley, May 1994. Master's Thesis, Issued as Technical Report 811.

[8] R. J. Gove. The MVP: a highly-integrated video compression chip. In *Proceedings of IEEE Data Compression Conference*, pages 215–224, Snowbird, Utah, March 28–31, 1994.

[9] R. Hopkins. Digital terrestrial HDTV for North America: the Grand Alliance HDTV System. *IEEE Transactions on Consumer Electronics*, 40(3):185–198, August 1994.

[10] Intel Supercomputer Systems Division, Intel Corporation. *Intel Paragon XP/S Technical Summary*, January 1994.

[11] ISO/IEC 11172-2. *Coding of moving pictures and associated audio for digital storage media at up ot about 1.5 Mbit/s*. MPEG (Moving Pictures Expert Group), International Organization for Standardisation, 1993. (MPEG1 Video).

[12] ISO/IEC 13818-2. *Generic coding of moving pictures and associated audio information*. MPEG (Moving Pictures Expert Group), International Organization for Standardisation, 1994. (MPEG2 Video).

[13] K. Jack. *Video demystified: a handbook for the digital engineer*. HighText Publications, Inc., California, 1993.

[14] J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe image coding. *IEEE Transactions on Communications*, COM-29(12):1799–1808, December 1981.

[15] D. Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[16] B. Liu and A. Zaccarin. New fast algorithms for the estimation of block motion vectors. *IEEE Transactions on Circuits and Sustems for Video Technology*, 3(2):148–157, April 1993.

[17] A. C. P. Loui, A. T. O. Ogielski, and M. L. Liou. A parallel implementation of the H.261 video coding algorithm. In *Procedure of the IEEE Workshop on Visual Signal Processing and Communications*, pages 80–85, Raleigh, North Carolina, September 2–3, 1992.

[18] H. G. Musmann, P. Pirsch, and H.-J. Grallert. Advances in picture coding. *Proceedings of the IEEE*, 73(4):523–548, April 1985.

[19] A. Puri, H. M. Hang, and D. L. Schilling. An efficient block-matching algorithm for motion compensated coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1063–1066, Dallas, Texas, April 6–9, 1987.

[20] K. R. Rao and J. J. Hwang. *Techniques and standards for image, video and audio coding*. Prentice Hall PTR, New Jersey, 1996.

[21] K. Shen, G. W. Cook, L. H. Jamieson, and E. J. Delp. An overview of parallel processing approaches to image compression. In *Proceedings of the SPIE Conference on Image and Video Compression*, volume 2186, pages 197–208, San Jose, California, February 1994.

[22] K. Shen and 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*, pages 407–418, San Jose, California, February 5–10, 1995.

[23] K. Shen and E. J. Delp. A spatial-temporal parallel approach for real-time mpeg video compression. In *Proceedings of the 25th International Conference on Parallel Processing*, pages II–100–II–107, Bloomingdale, Illinois, August 13–15, 1996.

[24] M. Tan, J. M. Siegel, and H. J. Siegel. Parallel implementation of block-based motion vector estimation for video compression on the MasPar MP-1 and PASM. In *1995 International Conference on Parallel Processing*, pages 21–24, Boca Raton, Florida, August 14–18, 1995.

[25] A. M. Tekalp. *Digital video processing*. Prentice Hall PTR, New Jersey, 1995.

[26] Y. Yu and D. Anastassiou. Software implementation of MPEG-II video encoding using socket programming in LAN. In *Proceedings of SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies*, volume 2187, pages 229–240, San Jose, California, February 7–8, 1994.

Table I: Real-time performance using the spatial-temporal parallel algorithm.

scheme	m	number of nodes	frames/second
1	2	510	31.43
1	4	510	33.59
2	2	510	43.64
2	2	382	33.60
2	4	508	40.04
2	4	380	30.70
2	8	504	32.38

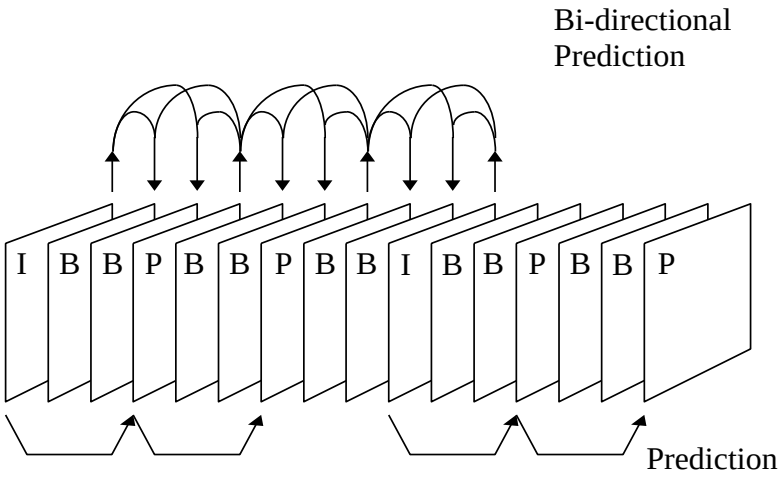

Figure 1: A video sequence with the I picture, P picture or B picture assigned to each frame.

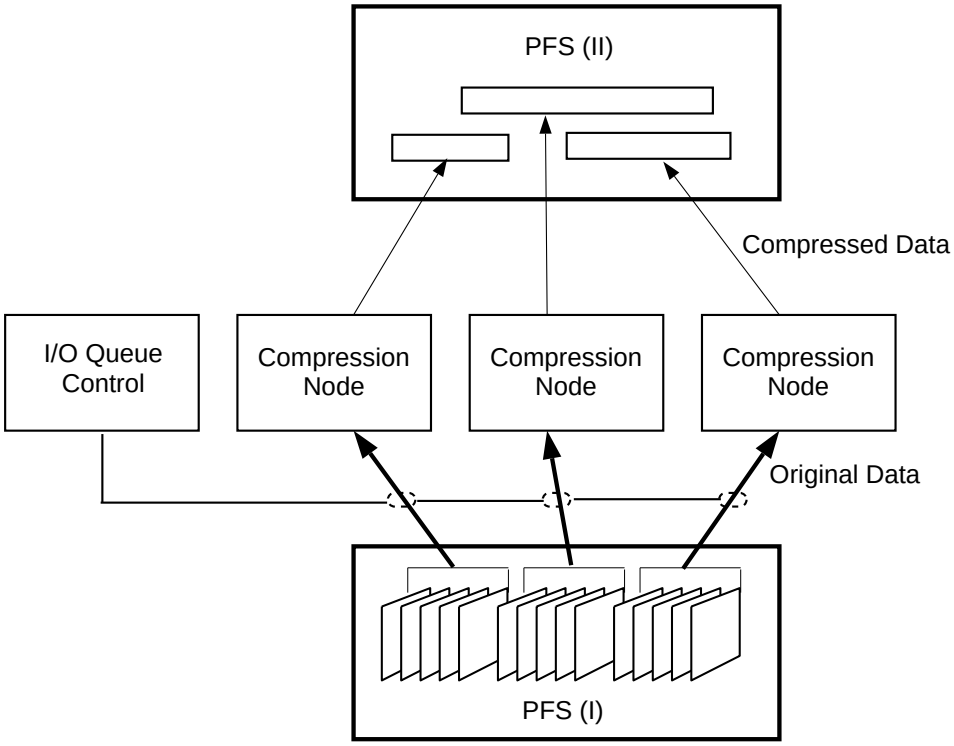

Figure 2: Block diagram of the temporal algorithm.

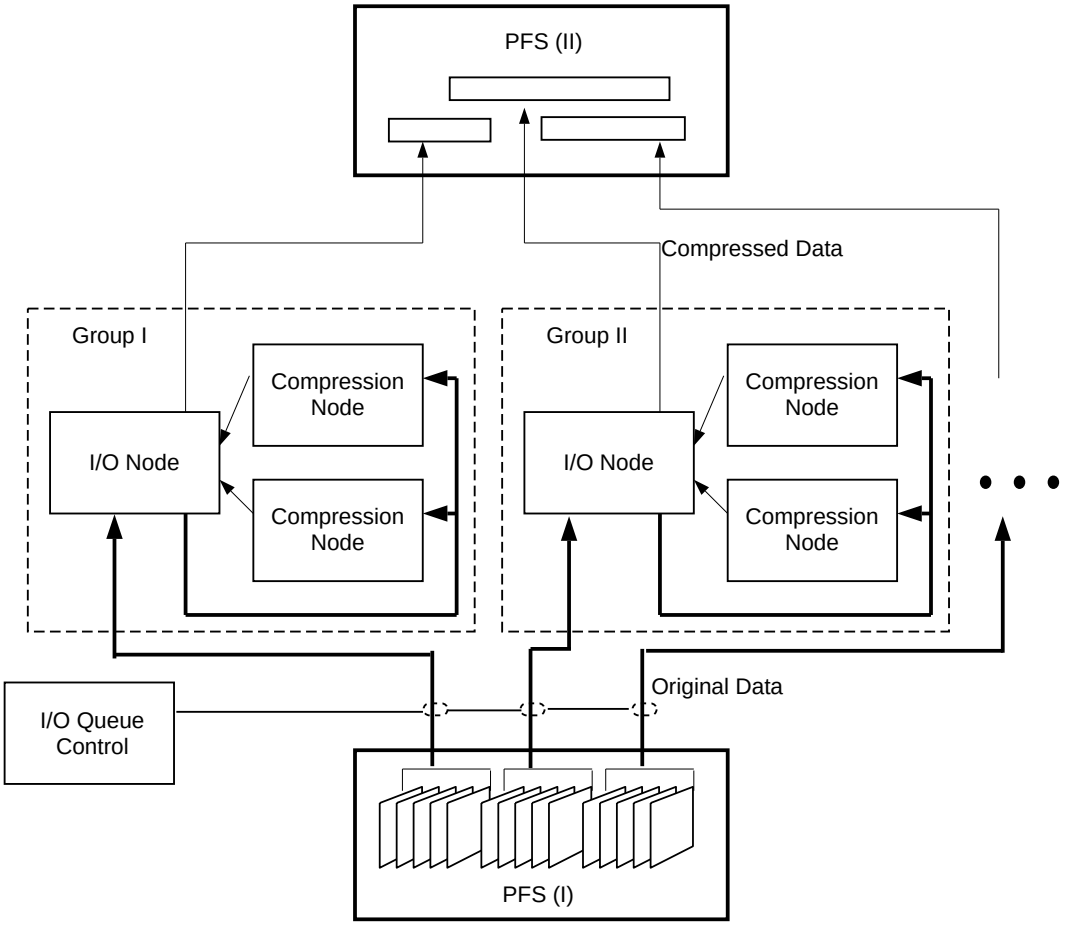

Figure 3: Block diagram of Scheme 1 of the spatial-temporal algorithm.

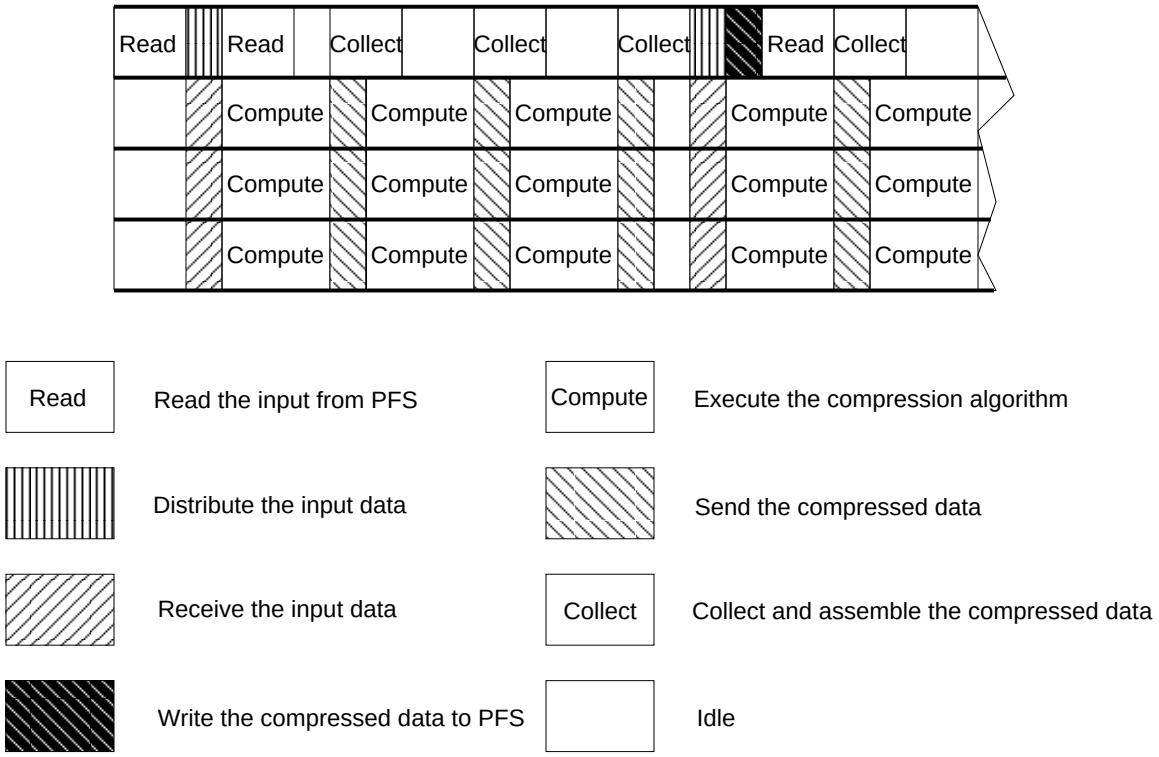

Figure 4: The timing diagram of task modules in Scheme 1. The task modules of a group of 4 PEs are shown. A video section of 3 frames is assumed. Horizontal axis: time, vertical axis: processors.

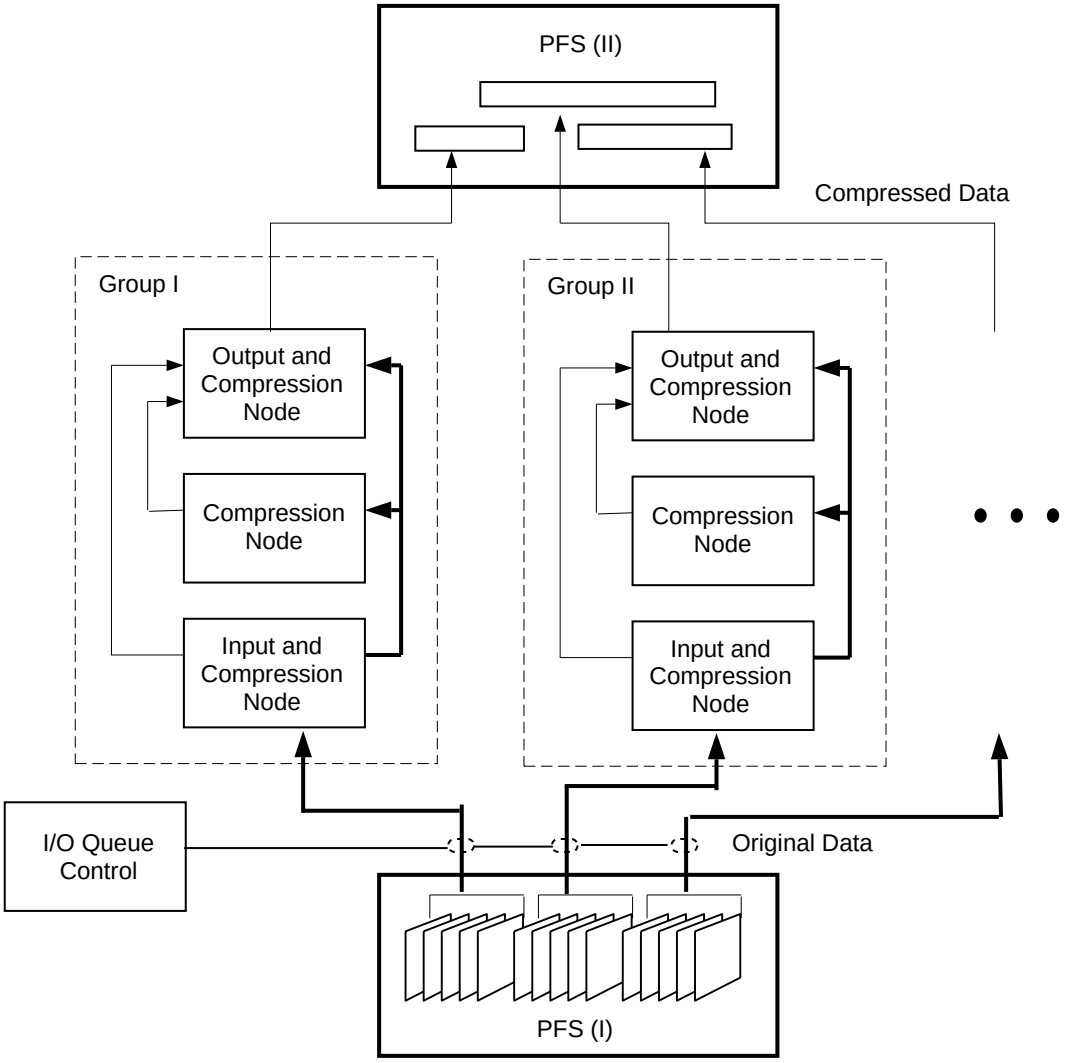

Figure 5: Block diagram of Scheme 2 of the spatial-temporal algorithm.

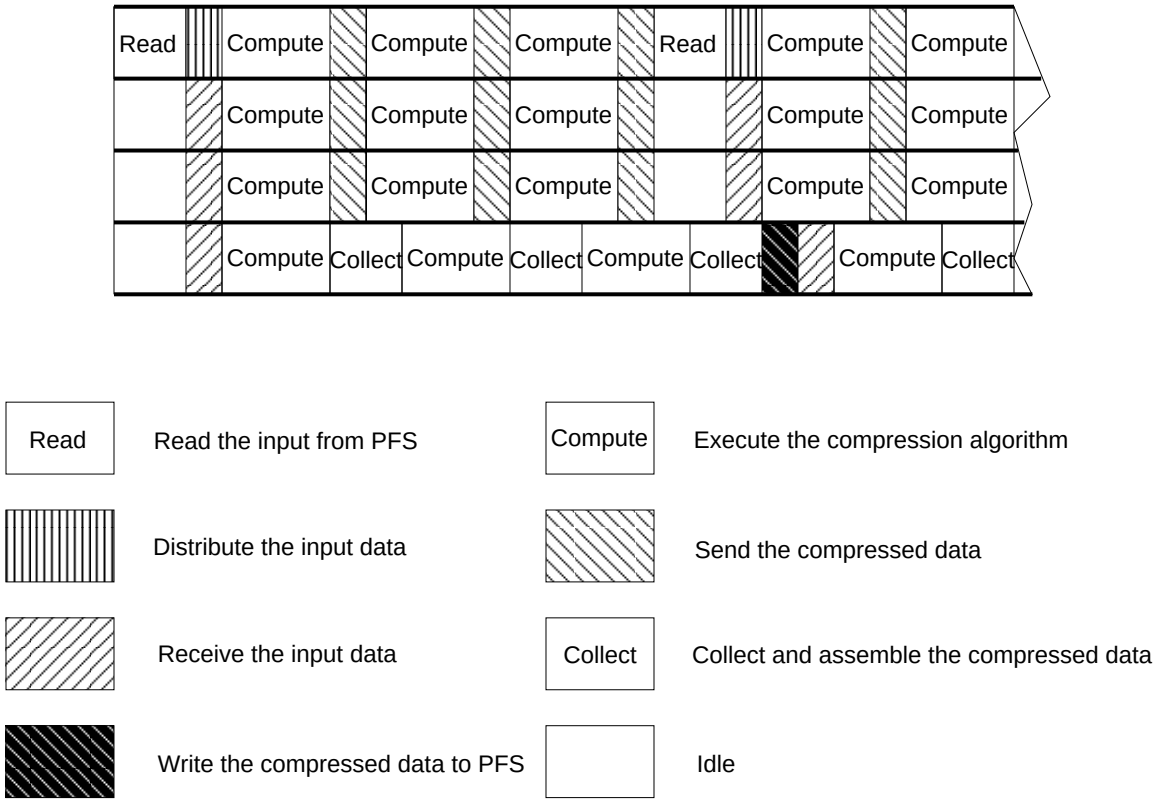

Figure 6: The timing diagram of task modules in Scheme 2. The task modules of a group of 4 PEs are shown. A video section of 3 frames is assumed. Horizontal axis: time, vertical axis: processors.

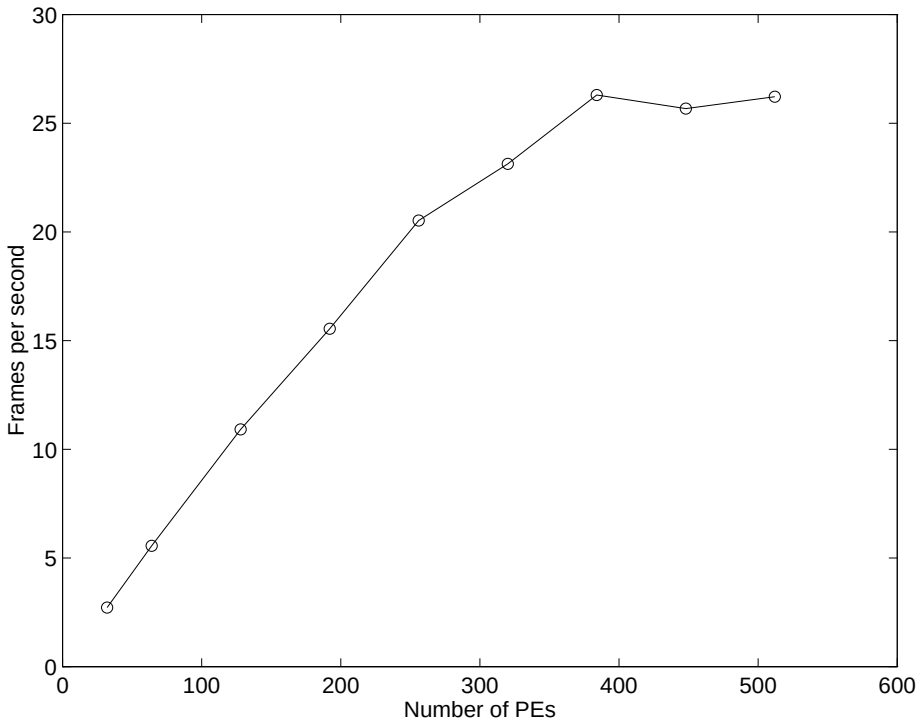

Figure 7: The overall speed for the compression of CCIR 601 video using temporal parallelism.

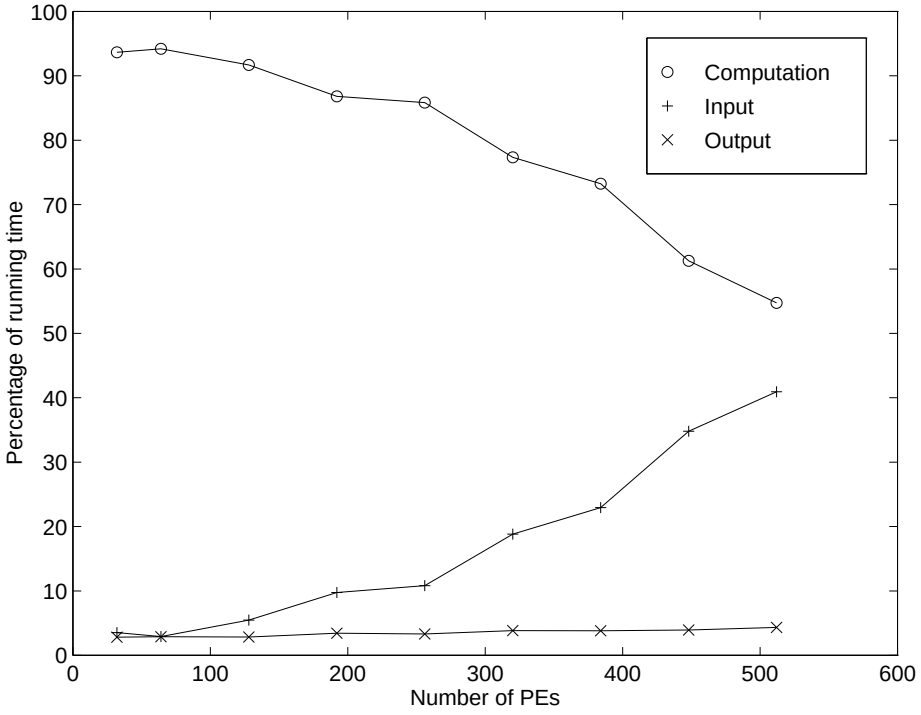

Figure 8: The percentage of running time used by different modules in the temporal parallel algorithm.

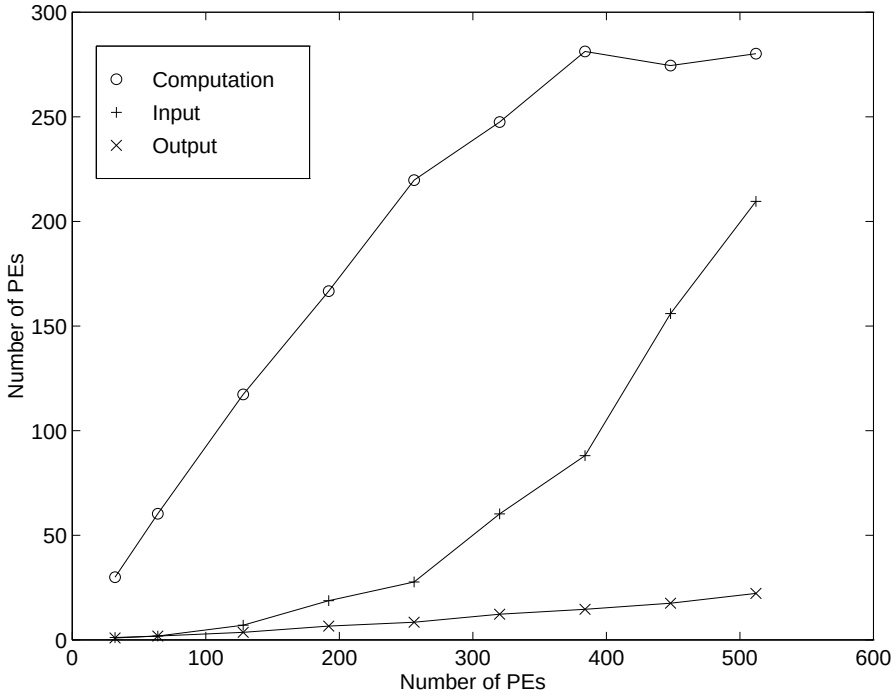

Figure 9: The virtual number of processors used for different modules in the temporal parallel algorithm

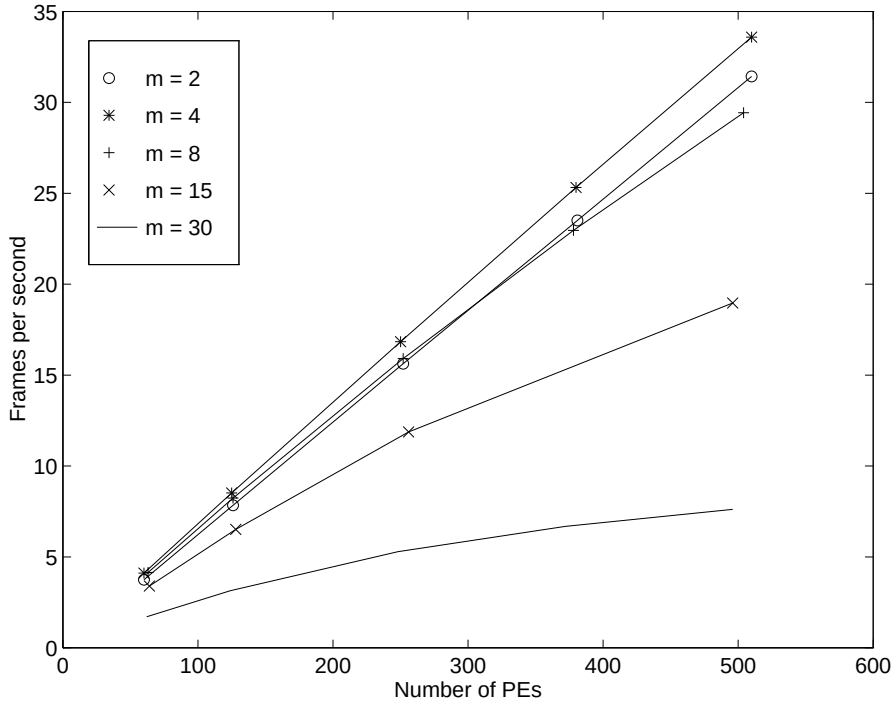

Figure 10: The overall speed for the compression of CCIR 601 video using spatial-temporal parallelism, Scheme 1.

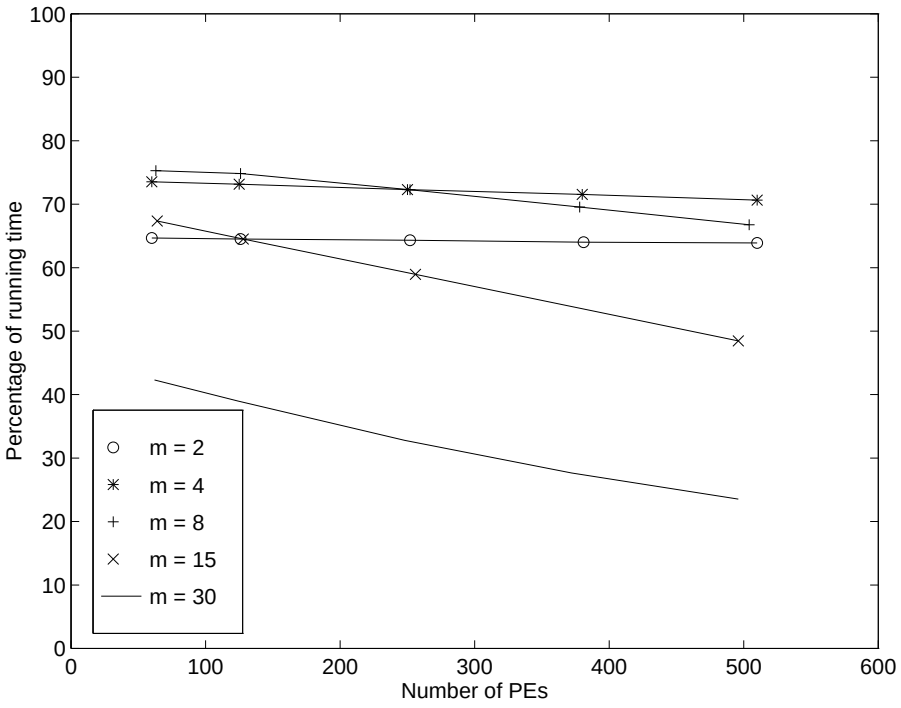

Figure 11: The percentage of time used for computation in the spatial-temporal parallel algorithm, Scheme 1.

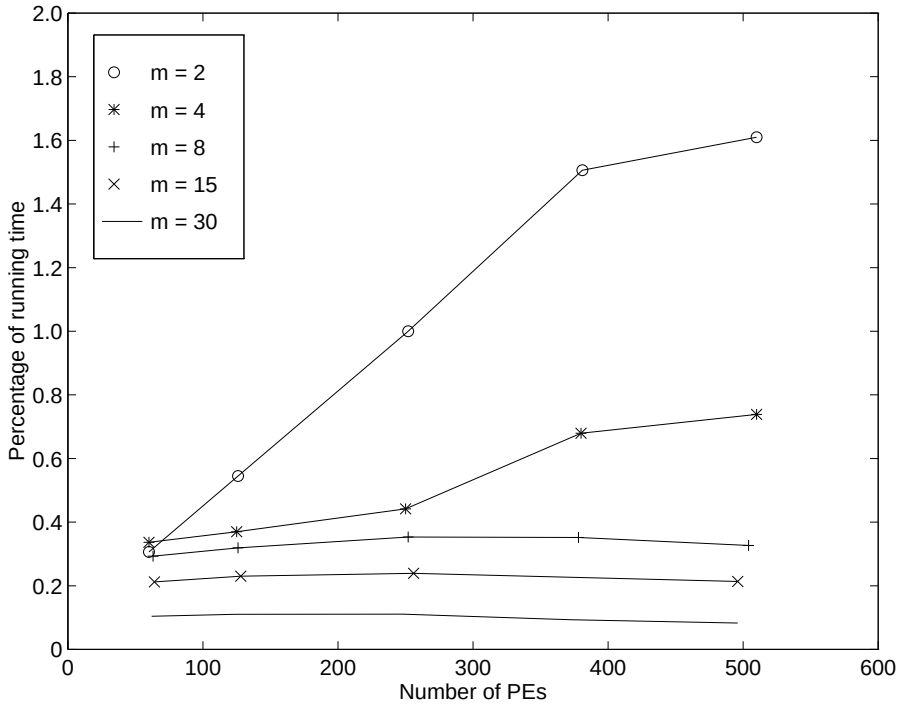

Figure 12: The percentage of time used for data input in the spatial-temporal parallel algorithm, Scheme 1.

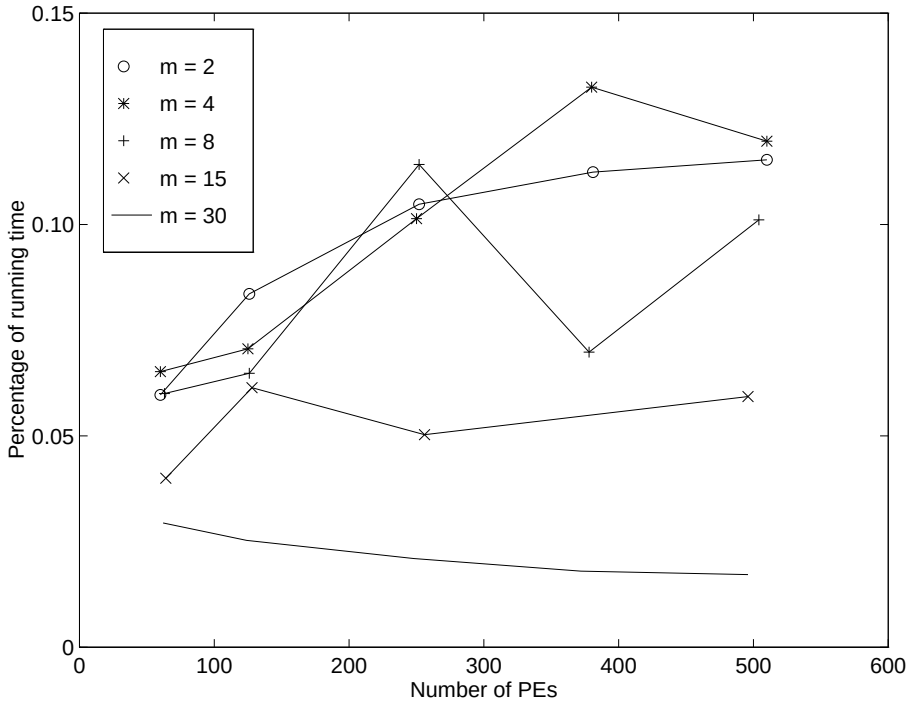

Figure 13: The percentage of time used for data output in the spatial-temporal parallel algorithm, Scheme 1.

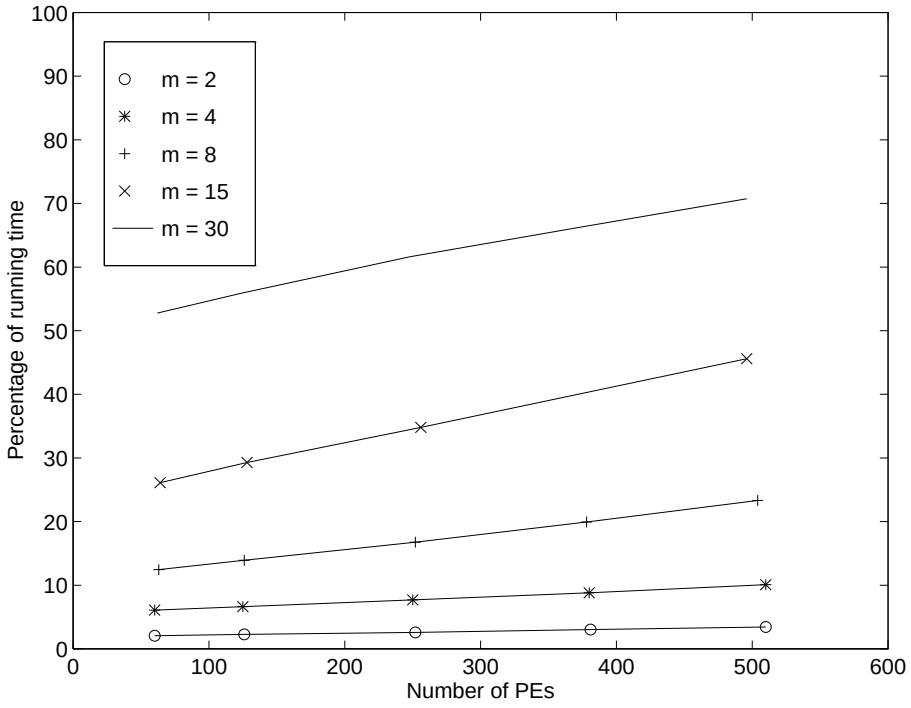

Figure 14: The percentage of time used for distributing the uncompressed data within each group in the spatial-temporal parallel algorithm, Scheme 1.

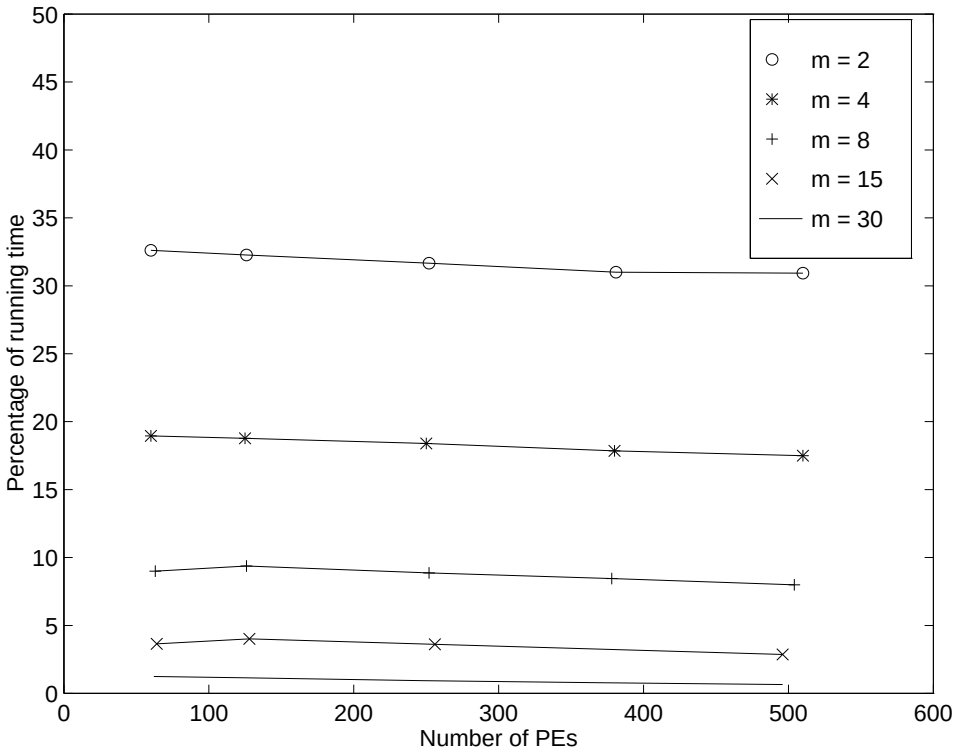

Figure 15: The percentage of time used for collecting the compressed data within each group in the spatial-temporal parallel algorithm, Scheme 1.

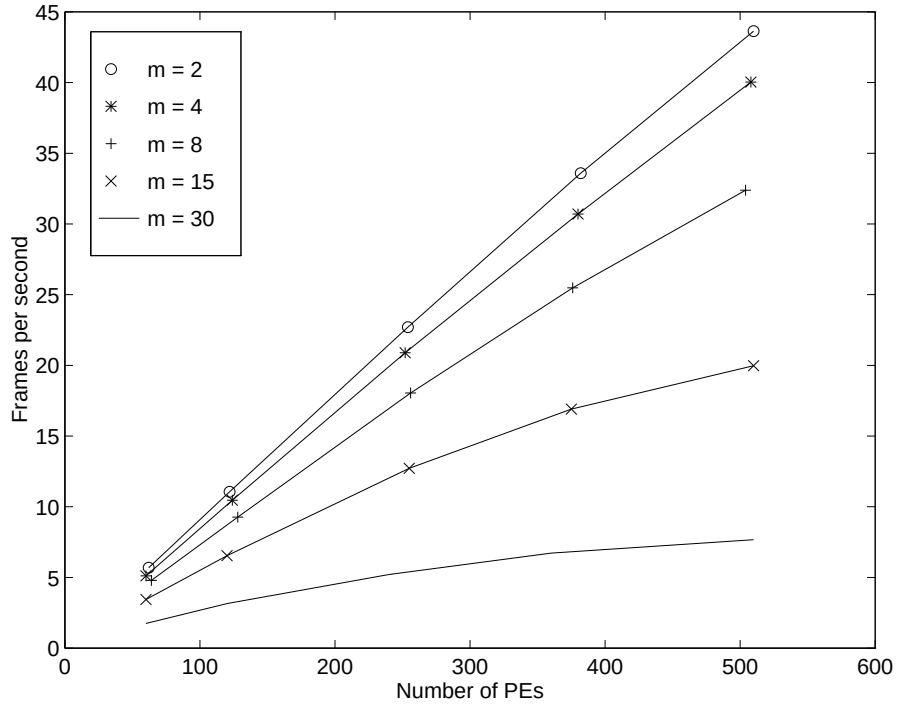

Figure 16: The overall speed of the compression of CCIR 601 video using spatial-temporal parallelism, Scheme 2.

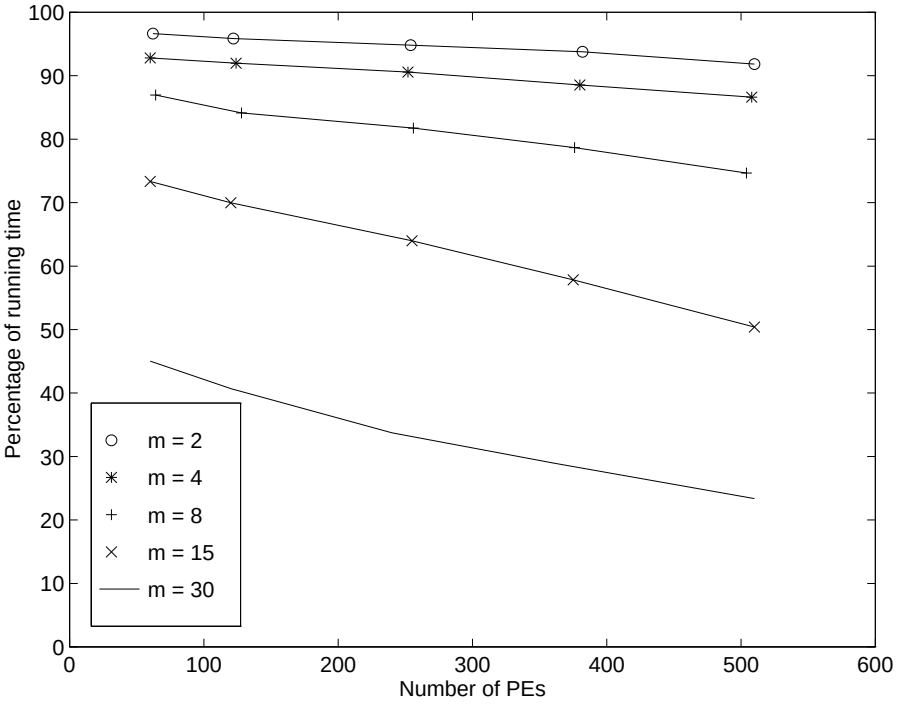

Figure 17: The percentage of time used for computation in the spatial-temporal parallel algorithm, Scheme 2.

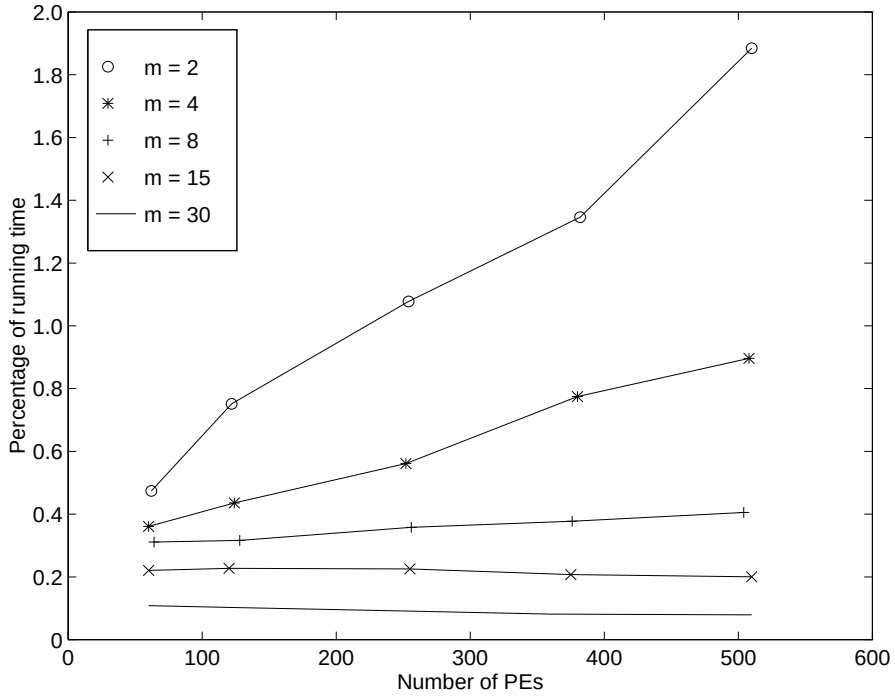

Figure 18: The percentage of time used for data input in the spatial-temporal parallel algorithm, Scheme 2.

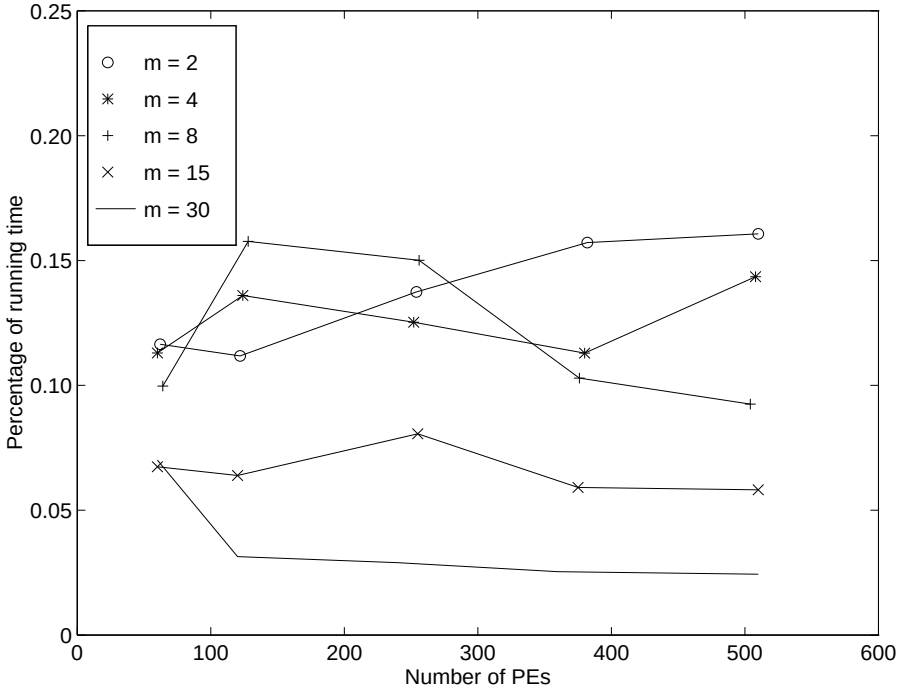

Figure 19: The percentage of time used for data output in the spatial-temporal parallel algorithm, Scheme 2.

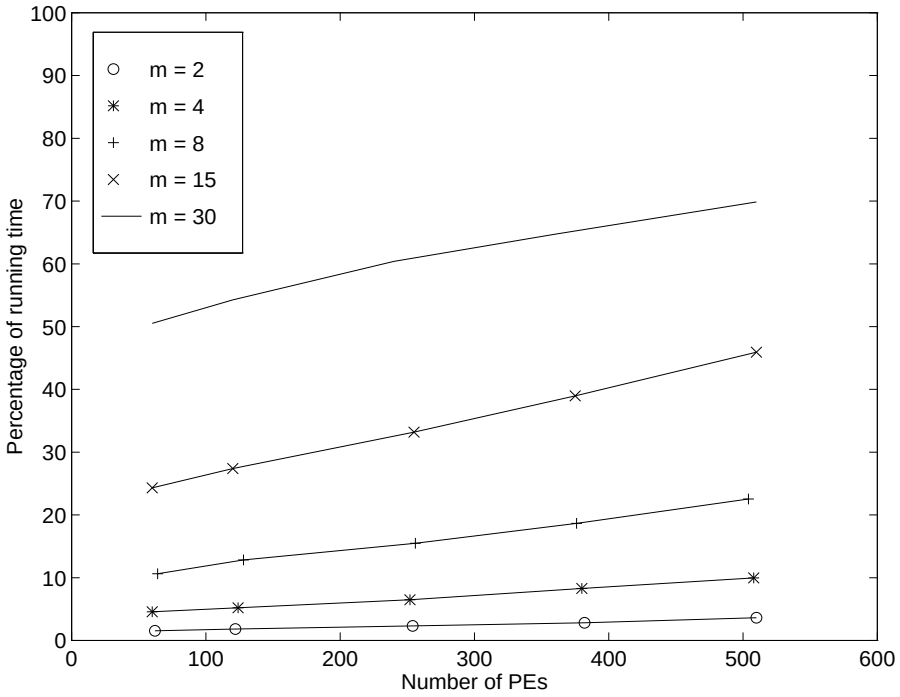

Figure 20: The percentage of time used for distributing the uncompressed data within each group in the spatial-temporal parallel algorithm, Scheme 2.

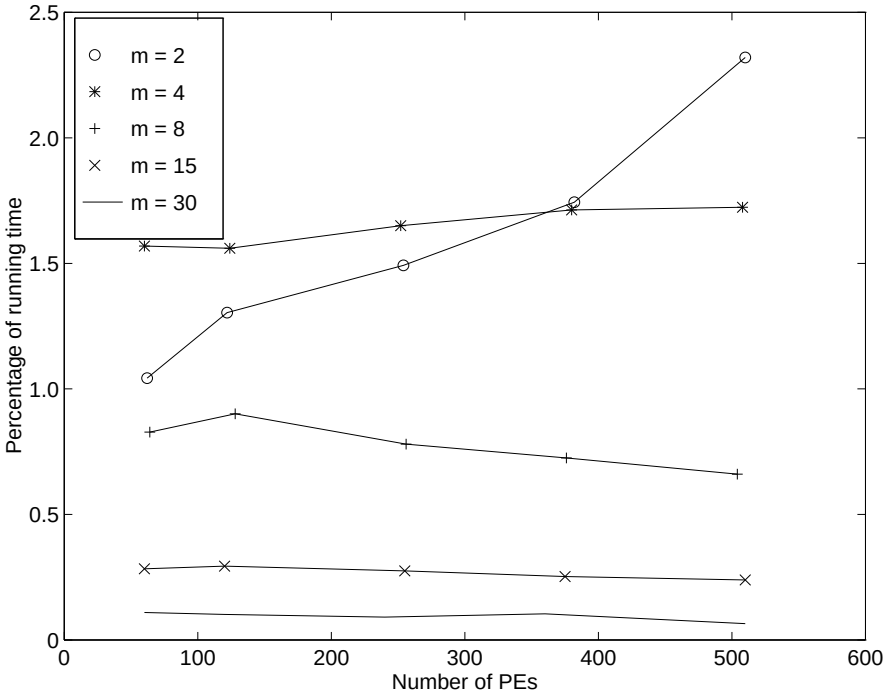

Figure 21: The percentage of time used for collecting the compressed data within each group in the spatial-temporal parallel algorithm, Scheme 2.

Biographies

Ke Shen

Ke Shen was born in Shanghai, China. He received the B.S. in Electronics Engineering degree from Shanghai Jiao-Tong University, China, in 1990. In 1992 he received the M.S. degree in Biomedical Engineering from the University of Texas Southwestern Medical Center at Dallas and the University of Texas at Arlington (jointly). He is currently pursuing a Ph.D. degree from Purdue University, West Lafayette, IN, where he is a research assistant in the School of Electrical Engineering. His research interests include image/video processing and compression, multimedia systems, and parallel processing. He is a member of Eta Kappa Nu.

Edward J. Delp

Edward J. Delp was born in Cincinnati, Ohio. He received the B.S.E.E. (cum laude) and M.S. degrees from the University of Cincinnati, and the Ph.D. degree from Purdue University. From 1980-1984, Dr. Delp was with the Department of Electrical and Computer Engineering at The University of Michigan, Ann Arbor, Michigan. Since August 1984, he has been with the School of Electrical Engineering at Purdue University, West Lafayette, Indiana, where he is a Professor of Electrical Engineering.

His research interests include image and video compression, medical imaging, parallel processing, multimedia systems, ill-posed inverse problems in computational vision, nonlinear filtering using mathematical morphology, communication and information theory.

Dr. Delp has also consulted for various companies and government agencies in the areas of signal and image processing, robot vision, pattern recognition, and secure communications. He has published and presented over 170 papers and has received research funding from NSF, NIH, DARPA, ONR, AFOSR, IBM, AT&T, Rockwell, Kodak, Intel, and GE.

Dr. Delp is a member of Tau Beta Pi, Eta Kappa Nu, Phi Kappa Phi, Sigma Xi, ACM, the Pattern Recognition Society, SPIE, IS&T, and a Fellow of the IEEE. In late 1995 he was elected Vice-Chair of the IMDSP Technical Committee of the IEEE Signal Processing Society. In 1994 he was elected Vice-President for Publications of IS&T. He is also co-editor of the book Digital Cardiac Imaging published by Martinus Nijhoff.

Dr. Delp was the General Co-Chair of the 1997 Visual Communications and Image

Processing Conference (VCIP) held in San Jose. He was Program Chair of the IEEE Signal Processing Society Ninth IMDSP Workshop held in Belize in March 1996. He was General Co-Chairman of the 1993 SPIE/IS&T Symposium on Electronic Imaging.

From 1984-1991 Dr. Delp was a member of the editorial board of the International Journal of Cardiac Imaging. From 1991-1993, he was an Associate Editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence. Since 1992 has been a member of the editorial board of the journal Pattern Recognition. In the fall of 1994, Dr. Delp was appointed Associate Editor of the Journal of Electronic Imaging. In January of 1996, he was appointed Associate Editor of the IEEE Transactions on Image Processing.

In 1990 he received the Honeywell Award and in 1992 the D. D. Ewing Award for excellence in teaching.

In 1990 he received a Fulbright Fellowship to teach and perform research at the Universitat Politecnica de Catalunya in Barcelona, Spain.

Dr. Delp is a registered Professional Engineer.