PARALLELIZATION OF THE H.261 VIDEO CODING ALGORITHM ON THE IBM SP2® MULTIPROCESSOR SYSTEM

N. H. C. Yung and K. K. Leung
Department of Electrical & Electronic Engineering
The University of Hong Kong, Pokfulam Road, HONG KONG
Email: nyung@eee.hku.hk

In this paper, the parallelization of the H.261 video coding algorithm on the IBM SP2 multiprocessor system is described. Based on domain decomposition as a framework, data partitioning, data dependencies and communication issues are carefully assessed. From these, two parallel algorithms were developed with the first one maximizes on processor utilization and the second one minimizes on communications. Our analysis shows that the first algorithm exhibits poor scalability and high communication overhead; and the second algorithm exhibits good scalability and low communication overhead. A best median speed up of 13.72 or 11 frames/sec was achieved on 24 processors.

1. Introduction

As video sequences contain a large amount of data both spatially and temporally, the ways in which these data are coded determines much of the cost for storage and transmission. Among the varieties of coding methods developed, the setting up of international standards has given a defined direction to encoder implementation which in turns fueled the rapid expansion of applications into multimedia computing, information storage, video-phone, medical imaging and other audiovisual services[1]. These standards are similar in many facets with different applications in mind. For instance, the H.261 is designed for audiovisual services at the rates of p×64 kbits/sec (where $1 \le p \le 30$) over an ISDN line[2]. It employs the discrete cosine transform in conjunction with motion estimation, compensation and uses variable length Huffman codes for channel coding. Along this line, the H.263 evolved from the H.261 to achieve a low bit rate at 24 kbits/s for video-phone over the PSTN. On the other hand, MPEG-1 (Moving Picture Expert Group) is designed for the storage of CIF (Common Intermediate Format) video and its associated audio at 1.5 Mbits/s on digital storage media, which operates random access, flexible frame rate and image size, and has compensation over one or more frames. The newer MPEG-2 standard aims to be used in all the digital transmission of broadcast TV quality video at coded bit rates between 4 and 9 Mbits/s[3-5].

So far, there are several attempts of parallel implementation of these coding standards. For instance, Sijstermans[6] implemented an MPEG-1 encoder using 100 M68020 processors. A measured speedup of 32 for a sequence of NTSC images was achieved on 100 processors, an equivalent of 0.5 frames/s. Akramullah et al[7] achieved real time performance of MPEG-2 coding on a 400-node Intel Paragon® XP/S using purely spatial partitioning, a speedup of 128 on 330 nodes. Adopting a more dedicated approach, Akiyama et al[8] outlined a pipelined structure of digital signal processors for different stages of the coding. Their simulation showed that real time encoding is possible, but no implementation was given. Bouville et al[9] developed a platform based on an array of TMS320C80 processors, and adopted the spatial parallelization approach, with no real results. Further, Agi & Jagannathan[10] implemented an MPEG-1 encoder on a network of workstations and CM5 system, using temporal

(0-7803-4229-1/97/$10.00 1997 IEEE)

parallelization. A speedup of 7.5 over a 12-node cluster of Sun SPARC 2 was achieved, an equivalent of 3 frames/s; and 4.5 frames/s was achieved on a 16 nodes CM5.

The goal of this research is to investigate how best the computing and communication resources can be utilized using spatial parallelization. The H.261 coding standard is chosen in this study because it has a high degree of complexity, data dependency and communication constraints. By considering the data grain size, data dependencies and communication issues, two parallel algorithms were developed on the IBM SP2 multiprocessor system. The first algorithm maximizes on processor utilization and the second one minimizes on communications. Our analysis shows that the first algorithm exhibits poor scalability and high communication overhead. A best median speedup of 10.5 on 23 processors, i.e. 8.36 frames/s, was achieved. For the second algorithm, it exhibits good scalability and low communication overhead. A best median speed up of 13.72 on 24 processors, i.e. 11.08 frames/s, was achieved. Both their performances agree with the theoretical prediction.

The organization of this paper is as follows: Section 2 outlines the H.261 standard and it's computing requirement; Section 3 describes the two parallelized algorithms; Section 4 presents the test conditions and detailed results; this paper is concluded in Section 5.

2. Overview of the H.261 Video Encoder

The H.261 encoder is a hybrid of inter-picture prediction to remove temporal redundancy, and transform coding of the remaining signal to reduce spatial redundancy of the video. The functional architecture of the coding algorithm is depicted in Fig. 1, where the major components are the motion estimation (ME)/compensation (MC), discrete Cosine transform (DCT) and variable length entropy coding (VLC). In the H.261, *macroblock* (MB) of size 16×16 is the basic unit for ME, where the last decoded frame is used to estimate the motion vectors of the current frame. Evaluation of the similarity between two MB's requires 2×16×16 integer operations. Searching of the motion vector is limited to an area within 15 pixel offset from the position of the MB. In the worst case, each MB requires 31×31 times of evaluation similarity in order to determine the most similar MB inside the area. For an CIF frame with 12 Groups of Blocks (GOB) and 33 MB's per GOB, the computing requirement for motion estimation of one frame alone is approximately 194 million operations. The computing requirement for motion compensation is much smaller than the estimation.

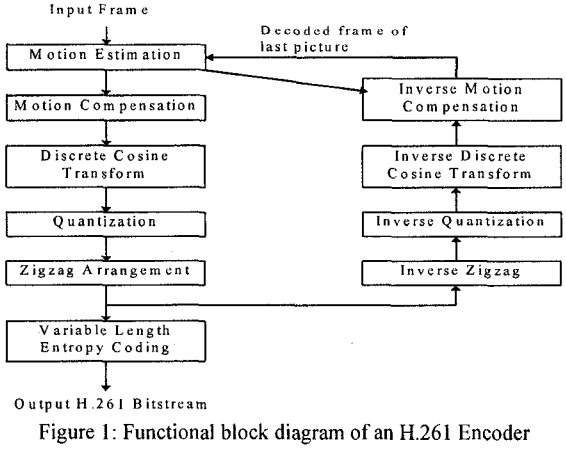

Figure 1: Functional block diagram of an H.261 Encoder

After motion compensation, DCT is performed on each 8×8 block to obtain the transformed coefficients. The computing requirement for this is $33\times12\times4\times3\times8^4$ where $3\times(8)^4$ is the number of operations for computing the DCT directly, i.e. approximately 19.5 million operations. The transformed coefficients of the DCT are then quantized to clamp most of the values to zero. The quantized DCT coefficients are then arranged into a zigzag pattern for the run-length coding. As the quantization and zigzag arrangement are simple operations, the computing requirement for these two functions is small. On the other hand, the VLC requires a number of comparisons and table lookups which the coding of each 8×8 block depends on the number of zeros preceding a non-zero coefficient, and the speed of assembling these. It is assumed to be similar to that of the DCT.

Considering only the major components, the computing requirement for coding one frame is roughly ~250 MOPS. For a 266 MFLOPS POWER2, the expected frame time would be around 1 second. This is a conservative estimation as it has excluded the overheads due to input/output, buffering and programming.

3. Parallelization of the H.261 Algorithm

In this research, the domain decomposition method is chosen, in which the input frame data is partitioned into a number of units and are mapped to the processors for computation, while the computations performed by each processor are identical. As the H.261 coding algorithm uses techniques based on reducing the spatial and temporal redundancy of an image sequence, there are naturally data dependencies between the MB's, GOB's and frames. Within an image frame, the organization of the MB's and the distribution of the blocks to the processors becomes a non-trivial task.

3.1 Data Partitioning, Dependency and Communication Issues

Theoretically, the unit of data partitioning can be as small as a pixel, although such fine grain partitioning introduces a huge amount of communications during ME/MC and other processes. As an MB is the basic unit used for ME/MC, it is natural to consider an MB as the smallest unit. However, if a unit is larger than a MB, then some of the parallelism would be lost because these MB's can be processed in parallel. In general, if the MB's are evenly partitioned, for a frame containing m MB's and system having n processors, each processor is allocated $\lceil m/n \rceil$ MB's.

Regarding data dependency, it exists between different MB's of the same frame while performing ME/MC, and in the VLC step when the MB address (MBA), motion vector data record (MVD) and MB type (MTYPE) are coded relative to its preceding neighborhood MB within the same GOB. The ME/MC step can be parallelized to some extend but the data referencing in the VLC step is inherently serial. To resolve this problem, the first method is to perform the VLC of the whole frame by a single processor, which is simple. But it has a potential critical path when the number of processors is large. The second method is to re-order the MB's and GOB's according to the GOB and MB hierarchy, and group those with data dependency into segments into one processor. With this, a slave still requires to reference the MBA, MVD and MTYPE fields of the last MB from a preceding slave. To eliminate this, a processor can compute the referenced MB information by itself and force the referenced MB in the preceding processor to adopt these values. The third method is to further divide the VLC step into a header part and a transformed coefficient (TC) part. Since

this data dependency exists only in the MB header, the TC VLC of an MB can be calculated independently.

On input communication requirement, very often, the data input to the encoder is an array of pixels ordered spatially, which the whole frame can be distributed as it is, and let the other processors extract the corresponding MB's for coding. However, the frame data can be ordered according to the GOB and MB structure in which the ordered list of MB's is divided into segments of equal length. The redundant communication in the latter case is minimal.

Moreover, a processor performing motion estimation to it's MB's requires all the decoded MB's of its own and some of those owned by the other processors, and hence, introduces a demand for communication between them. There are two methods to simplify this communication requirement: first, a designated processor can be used to collect all the decoded MB's before it broadcasts them as a whole frame back to all the processors. There is substantial data redundancy with this approach. The second method is to agglomerate the MB's into rows. This method reduces the amount of redundant communications, and allows all the processors to perform exchange operations with their neighbors concurrently.

3.2 Two Parallelized H.261 Algorithms

The SMMS algorithm was developed based on a single-master-multiple-slave configuration where a master is designated for the centralized communication and ordering of the MB's, and the slaves are responsible for the computation. As depicted in Fig. 2, Master#1 reads the input frame data, reorganizes the array of pixels into any array of MB's, then distributes them to the slaves evenly. Master#1 also broadcasts the last decoded frame to the slaves, where all the decoded MB's are collected at the end of coding the last frame. Upon receiving the decoded frame and the MB's from the current frame, the slaves compute the motion vectors and all the subsequent sub-processes in parallel. In this case, the VLC is parallelized by the slaves using the 2nd method described in Sec. 3.1. Finally, Master#1 collects the statistics, VLC results and the decoded MB's from the slaves. The coded bit stream is then sent to the standard output. In this algorithm, most of the computations are carried out in the slaves in parallel, with all the communications being managed by the master. Although the number of MB's distributed to each slave is identical, due to the difference in motion content in each MB, the computing times for the MB, or a group of MB's are different. Normally, MB's that contain high motion content take longer ME time. As the motion content of the video is not known, even distribution of MB's seems to be the most appropriate.

To minimize communications, the MMMS algorithm uses three masters for separately handling the distribution of MB's, parallelization of the VLC and the collection of results and statistics as depicted in Fig. 3. The MB's are ordered by Master#1 in the same way as the input pixel array and evenly divided into segments of rows for distribution. During this time, the slaves exchanges decoded MB's of the last frame according to whether the MB's are the immediate neighbors of the current MB segments assigned to it. Once the two sets of data are ready in the slaves, coding begins. When the coding of the last MB is completed, Master#2 collects the statistics, and in parallel, Master#3 collects the VLC immediate results and concatenates the header before sending the bit stream to the standard output (3rd method in Sec. 3.1). In this algorithm, the serial communication tasks are now parallelized as follows: first, the decoded MB's are exchanged between the slaves, thus saving communication time to the master, which can be performed in parallel to the distribution of the current frame data.

Second, the collection of statistics and VLC results are now handled in parallel by two masters.

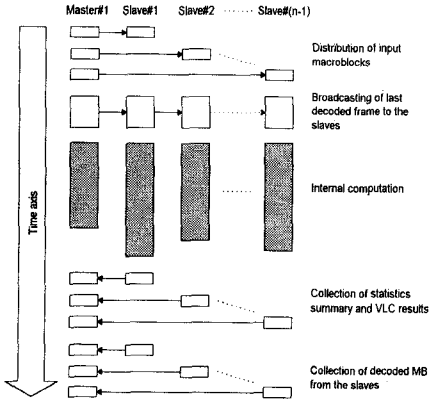

Figure 2: SMMS algorithm

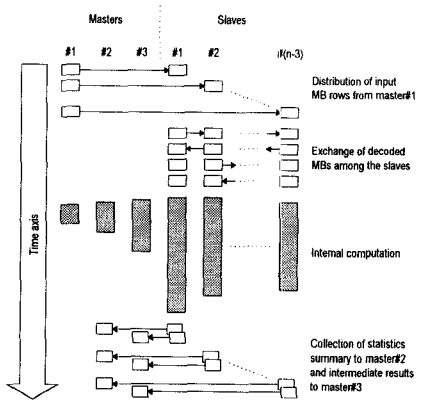

Figure 3: MMMS algorithm

3.4 Performance Prediction

Let n be the number of processors available; $T_{cp}(j)$ be the computation time of processor j, where $j = 1, 2, \ldots, n$; M_i be the size in bytes of a frame; M_o be the average size of a coded frame; M_s be the size of a statistic record; T_w be the asymptotic bandwidth of the communication channel in second per byte; and T_s be the overall startup time of the channel. The frame time of the SMMS algorithm is given by

$$T_{frame} = \left\{(n-1)\cdot\left(\frac{M_i}{n}\cdot T_w + T_s\right)\right\} + \left\{(n-1)\cdot\left[\left(\frac{M_o}{n}\cdot T_w + T_s\right) + \left(M_s\cdot T_w + T_s\right)\right]\right\} + \left\{\max\left\{T_{cp}(j), j = 1,2,\ldots,n\right\}\right\} + \left\{(n-1)\cdot\left(\frac{M_i}{n}\cdot T_w + T_s\right)\right\} + \left\{52\cdot\log(n) + \left[0.029\cdot\log(n)\right]\cdot M_i\right\} + T_{\text{in}} + T_{\text{out}}, \quad (1)$$

where the 1st {} represents the time taken for the master to send M_i/n bytes to the slaves; the 2nd {} represents the time taken for the VLC results and statistics to travel to the master; the 3rd {} represents the computation critical path; the 4th {} represents the time taken to collect the decoded data to the master; the 5th {} represents the time taken to broadcast the decoded frame; T_{in} represents the reading of a frame and MB rearrangement time; and T_{out} represents the writing of the encoded results to an output bit stream.

The frame time of the MMMS algorithm is given by $T_{frame} = \max\{T_1, T_2\} + T_{in} + T_{out}$ where

$$T_1 = \left\{(n-3)\cdot\left(\frac{M_o}{n-3}\cdot T_w + T_s\right) + T_{cp}(3)\right\} \quad (2)$$

$$T_2 = \max\left\{T_{cp}(j), j = 4,5,\ldots,n\right\} + \left\{M_s\cdot T_w + T_s\right\} + \left\{\frac{M_i}{n-3}\cdot T_w + T_s\right\} + \left\{\frac{M_o}{n-3}\cdot T_w + T_s\right\} + \left\{4\cdot\left(\frac{M_i}{R}\cdot T_w + \left\lceil\frac{n}{R}\right\rceil T_s\right)\right\} \quad (3)$$

where R is the number of MB in a column; T_1 represents one of the two critical paths due to the 3rd master, which consists of the time taken to collect the TC VLC, and the time taken to compile the VLC header ($T_{cp}(3)$); and T_2 represents the second critical path due to slave computations, in which the 1st {} represents the computation critical path; the 2nd {} represents the time taken to send the statistics to the 2nd master; the 3rd {} represents the time

taken for the slaves to receive the input MB's; the 4th {} represents the time taken to send the VLC result to the 3rd master; and the 5th {} represents the time taken for a slave to exchange MB's with it's neighbors. T_{frame} of the two algorithms are plotted in Fig. 4 & 6, respectively.

4. Results and Discussions

4.1 Data Collection Conditions

The IBM SP2® system used for this investigation has a total of 32 processors installed at the University of Hong Kong. Among the 32 processors, 24 can be used exclusively by an application within a limited time window. Each processor consists of a 66.7MHz POWER2® RISC processor with 64 MB main memory and 2 GB disk storage, providing a peak performance of 266 MFLOPS. The measured bandwidth between any two processors is 10 MB/s, much lower than the peak bandwidth, and the measured latency using the message passing library (MPL) for an empty message is ~140 μs.

The software H.261 used is the PRVG-P64 from Portable Research Video Group of Stanford[11]. The original program is a serial program for running on a single computer, at a moderate performance. In deriving the parallel algorithms, all the basic functional blocks of code are left unchanged. It is the macroscopic backbone of the program that is changed for multiprocessing.

The H.261 program was compiled using mpcc -O [filename.c], using single precision integer format throughout. The wall-clock time generated by *gettimeofday()* was used to measure the overall execution duration and individual execution time per stage, where all the processors were synchronized and timed at the start of the execution. Blocking send and receive were used for all the point-to-point communications where fixed startup time and constant channel bandwidth were assumed. The broadcast time is measured from all the processors are ready to receive until all of them have received. The HPS user space communication protocol was used to obtain the best performance from the network.

A video of 39 frames of a table tennis player playing the ball was used for the test. The coded output from the two parallelized algorithms were checked byte-by-byte against the serial program output. The three coded output streams were decoded for visual inspection and comparison.

4.2 SMMS Algorithm Results

Fig. 4 depicts the measured median speedup versus n, together with the linear and the predicted speedup as given by Eqt. (1), where t_s is assumed to be 46 μs and t_w is assumed to be 25 ns/byte. A best median speedup of 10.5 or 8.36 frames/s on 23 processors was achieved. It is observed that first, the predicted speedup is very close to the measured speedup. Second, the speedup for small n is close to the linear speedup. But as n increases, their difference becomes more apparent. At n=15, the measured speedup levels off. This highlights the effect that when the n is more than the algorithm's degree of parallelism, adding more processors does not increase the performance. Also noted in Fig. 5, the computation time decreases from 100% to around 50% when n is increased from 1 to 24. On the other hand, the communication time increases from 0% to 20%; and the idle time increases from 0% to almost 40% in the case of 22 processors. The spike in the idle time is due to the method used to divide the MB's into segments where one of the slaves was assigned a larger than average number of MB's creating a longer critical path.

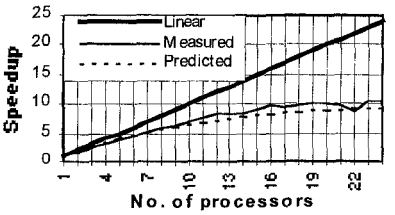

Figure 4: SMMS-median speedup

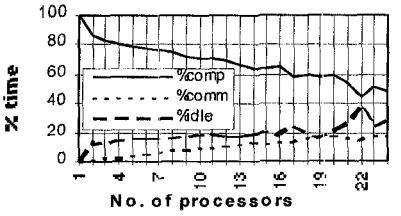

Figure 5: SMMS-percentage component time

4.4 MMMS Algorithm Results

Fig. 6 depicts the measured median speedup versus n for the MMMS algorithm, together with the linear and predicted speedup. It can be observed that first, the measured speedup and the predicted speed generally agree with each other but not as close as the SMMS case. This is probably due to the prediction of the time required for exchanging decoded MB's between slaves is too optimistic. Second, the speedup is very poor for small n, due to the use of multiple masters. Third, the median speedup is close to 13.72 or 11.02 fps on 24 processors. Fourth, the speedup function is reasonably linear and that further speedup looks possible. When we consider Fig. 7, the relationship between the computation, communication and idle times is quite different from the previous algorithm. In the MMMS algorithm, the percentage of computation began at less than 30% for 4 processors and increased to over 60% for 24 processors. Over this range, communication time varied from close to 0% to about 5%, and the idle time varied from over 70% to below 40%. At 20 processors, one of the slaves was assigned a larger than average number of MB's, which causes the critical path length to increase.

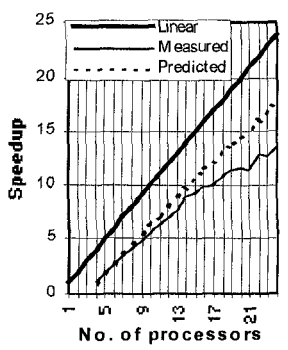

Figure 6: MMMS- speedup

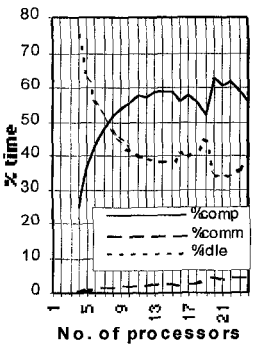

Figure 7: MMMS- component time

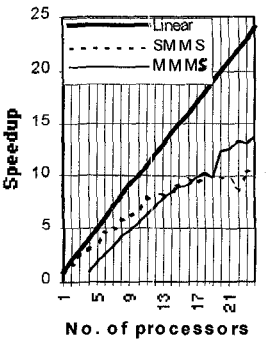

Figure 8: Speedup comparison

4.5 Comparing the Two Algorithms

From Fig. 8, we can observe that the two algorithms behave rather differently. For the SMMS algorithm, it performs well with small n because a high percentage of time is spent on computation; and performs poorly with large n because of a high percentage of time is now spent on communication or idle. Specifically, for 4 processors, the speedup by the SMMS

algorithm is 3.17 times higher than the MMMS algorithm. In contrast, the MMMS algorithm performs much better for large n with a final speedup of 13.72, which is 30% higher than the 10.5 of the SMMS algorithm.

5. Conclusions

We can conclude that first, the use of the domain decomposition approach to parallelizing the H.261 coding algorithm is a viable method as long as the data partitioning and communication issues have been carefully assessed. Second, it is advantageous to partition a frame into MB segments rather than single pixels, i.e., coarse grain parallelization is more suitable on the IBM SP2 and similar machines. Third, the resultant implementation is portable as the sequential software encoder is off-the-shelf and can be readily ported to other platforms. Fourth, the MMMS algorithm scales well with n, but not the SMMS algorithm. Fifth, the communication cost of the MMMS algorithm is low, whereas it is high for the SMMS algorithm. Sixth, frame rate of around 11 fps has been achieved based on a serial software with moderate performance. From the results, real-time performance looks likely with larger n.

References

1. V. Bhaskaran & K. Konstantinides, "Image and Video Compression Standards: algorithms and architectures", *Kluwer Academic Publishers* (1995).
2. "ITU-T recommendation H.261: video codec for audiovisual services at px64 kbits", *International Telecommunication Union* (1990).
3. "ITU-T recommendation H.263: video coding for low bitrate communication", *International Telecommunication Union* (1995).
4. "MPEG-1: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s", *ISO/IEC 11172* (1993).
5. "MPEG-2: Generic coding of moving pictures and associated audio", *ISO/IEC 13818* (1995).
6. F. Sijstermans & J. Van der Meer, "CD-I Full-motion Video Encoding on a Parallel Computer", *Communications of the ACM*, **34**, **4**, 81-91 (1991).
7. M. Akramullah, et al, "A Portable and Scalable MPEG-2 Video Encoder on Parallel and Distributed Computing Systems", *SPIE Proc. On Visual Communications and Image Processsing*, 973-984 (1996).
8. T. Akiyama et al., "MPEG-2 Video Codec using Image Compression DSP", *IEEE Transactions on Consumer Electronics*, **40**, **3**, 466-472 (1994).
9. C. Bouville, et al, "DVFLEX : A Flexible MPEG Real Time Video Codec", Proc. Of *IEEE Int. Conf. On Image Proc.*, Vol. **II**, 829-832 (1996).
10. I. Agi & R. Jagannathan, "A Portable Fault-tolerant Parallel Software MPEG-1 Encoder", *Multimedia Tools and Applications*, **2**, 183-197 (1996).
11. A. C. Hung, "PVRG-P64 Codec 1.1", *Portable Video Research Group (PVRG), Stanford University* (1993).