Fast and Parallel Video Encoding by Workload Balancing

N. H. C. Yung & K. C. Chu
Department of Electrical & Electronic Engineering
The University of Hong Kong
Chow Yei Ching Building, Pokfulam Road, HONG KONG SAR
Email: {nyung, kcchu}@eee.hku.hk

ABSTRACT

Today's video coding/decoding technology captures a wide area of applications such as video phone/conferencing, interactive TV and many other audio-video services. Ideally, the coding of the video should be fast enough to offer real-time performance (>24 f/s). However, the inherent computing complexity of some of the coding components including motion estimation, discrete cosine transform and variable length entropy coding, means that fast implementation on parallel computing platform is potentially fruitful. Over the years, results have been reported on the implementation of parallel MPEG and H.261 encoders [1-4], where spatial or temporal data parallelism is commonly exploited. Most of these methods decomposed a fixed number of macroblocks (MB) in an arbitrary sense. As the MB's delays are different because of motion content, this approach introduces uneven workload across the processors, causing long critical path and poor utilization of the processors [5]. In this paper, we explore the issue of balancing the MB computing workload across the processors. This includes first, the prediction of the workload based on the previous frame workload, and second, the scheduling of the MB bounded by the locality constraint (Fig. 6). The algorithm was implemented on an IBM SP2, and the results showed that the reduction in the worst case delay is around 19-23%, with both the prediction and scheduling overhead taken into account (Fig. 9b). Because of the critical path reduction, the overall processor utilization was increased, and the overall coding rate improved.

1. INTRODUCTION

The H.261 video codec is mainly designed for audio-visual services such as video phone or video conferencing at the rates of p×64 kbits/second through the ISDN network [6]. Due to the nature of these applications, encoding of the input frames must be fast enough to offer a close to real-time frame rate (>24 f/s) and minimize the number of dropped frames. However, the inherent computing complexity of motion estimation (ME), discrete Cosine transform (DCT) and variable length coding (VLC) [7], means that single processors today are far from offering a satisfactory solution. For this reason, most researchers take one of the two directions: develop a new generation of fast coding technique [8]; or implement the video encoder on parallel computing platforms [1-4]. For parallel implementation, the focus has been on the MPEG standards, with [1] using a functional decomposition approach on a 100 M68020, achieving a utilization rate of 32%. In [2], a data parallel technique was adopted to achieve real-time MPEG-2 coding on 330 nodes of an Intel paragon XP/S, or an equivalent of 38.7% utilization. While [3] implemented the MPEG-1 on a network of workstations, at a utilization of 62.5%. In [4], the H.261 coding algorithm was considered and a number of spatial and temporal parallel algorithms were developed, with a utilization of 57.3% achieved.

Of all these development, it is interesting to note that most groups strive to achieve real-time coding for MPEG-1 or MPEG-2, instead of the H.261 or H.263. Of the four international standards, both the H.261 and H.263 are designed for video phone/conferencing that require real-time encoding, whereas the MPEG-1 and MPEG-2 only need real-time decoding, not encoding. Furthermore, most of these approaches have taken a one-off parallelization approach without dealing with the issues of optimality and generality. In particular, the derivation of the granularity of spatial and temporal parallelization seems to be arbitrarily on fixed data packet size, rather than other possible criteria such as computing delay or memory utilization [5]. As the computing delay of motion

0-7803-4778-1/98 $10.00 © 1998 IEEE

estimation is dependent on the motion in the sequence, a fixed number of macroblock (MB) approach would only mean that the delay for estimating motion in a frame is different for each MB.

For this reason, the research presented here aims to investigating a workload balancing scheme that is based on balancing the computing delays across the number of processors. The success of this balancing scheme hinges on two issues: a viable workload scheduling and the accurately predicted computing delays for the scheduling. Further, the allocation of MB to processor is determined by the predicted computing delays and bounded by the MB locality constraint. In this paper, both the balancing and prediction issues are dealt with, and an algorithm is proposed. It predicts the current MB delays using the actual and predicted MB delays of the last frame; and balances the workload by comparing the remaining mean delay per processor with the allocated value, to ensure that the MB locality constraint is satisfied. By implementing the algorithm on IBM SP2 platform, our results show that the balancing approach is very effective, offering close to theoretical performance if the prediction is accurate. The critical path is only 4.5% longer than the theoretical in the balanced case, whereas the unbalanced case is 35% longer. Even if the prediction is less than ideal, the improvement is still substantial.

2. CRITICAL PATH ISSUE

Typically, the computation of a problem can be considered as comprising a sequential part S and a parallel part P [5]. The sequential part is the computation that can only be carried out by one processor while the parallel part is the rest of computation that can be performed by more than one processor in parallel. In a single processor implementation, its critical path (or longest computing delay path) is defined as the sum of delays for calculating S and P. For parallel implementation, its critical path is defined slightly differently as now P can be calculated in parallel. Assume there are n processors available for the computation, and the overhead for achieving parallel computation is minimal, the critical path for parallel calculation is given by the sum of S and P/n as illustrated in Fig.1.

If we define the speedup, S_n, to be the ratio between the computing delay using one processor and the computing delay using n processor, then Amdahl's law [5] can be modified to

$$S_n = \frac{t_S + t_P}{t_S + t_P/n + t_o(n)}, \qquad (1)$$

where $t_O(n)$ denotes the parallelization overhead. Ideally, for n processors, S_n equals to n. This is only possible when t_S & $t_O(n)$ are zero, i.e., there is no sequential component in the problem or parallelization overhead. In practice, for $n \geq 2$, S_n increases, and for large n, S_n becomes almost constant.

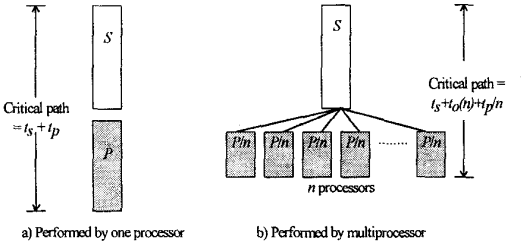

Fig. 1: Critical paths

In the case of implementing a video encoder on a multiprocessing system using domain decomposition, the problem still can be viewed as having S and P, where S is given by the reading of frame data from the camera buffer and writing of the coded bit stream; and P is given by the computation of ME, DCT and etc. of each MB done in parallel. The parallelization overhead is given by the distribution of MB among the processors, and the collection of VLC and decoded MB from them. Considering MB is the smallest unit of decomposition and they are distributed evenly among processors, for N MB in a frame, $\lceil N/n \rceil$ MB are distributed to each processor as depicted in Fig. 2.

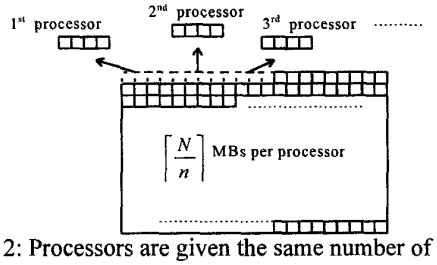

Fig. 2: Processors are given the same number of MB

In theory, if each MB incurs an identical computing delay, then the delay through each parallel computing path is the same and the critical path is given by $t_S+t_O(n)+\lceil N/n \rceil t_{MB}$, where t_{MB} is the delay for coding one MB. However, as motion estimation depends on the motion content in the sequence and VLC depends on the previous processes, the computing delay for an MB is not likely to be constant. Fig. 3 depicts a more realistic case of having the critical path determined by the longest delay in the parallel computation, given by Eqt. (2).

$$\text{Critical path} = t_s + t_o(n) + \max_{\text{all i}}(t_p(i)) \quad (2)$$

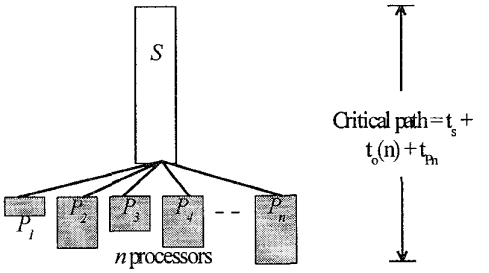

Fig. 3: Realistic critical path of parallel encoding

From Eqt. (2), the uneven parallel delays means that some processors will complete their computation earlier than the others. As a result of these processors staying idle, waiting for the rest to finish, the utilization of the implementation is reduced, which is normally defined as the ratio between the speedup factor and n. To alleviate this problem, it is possible to balance the computation across the processors according to the workload per MB rather than on a fixed number of MB per processor. However, the computing delays must be somehow known or estimated before they can be scheduled to each processor on an equal computing delays basis. These two issues are explored in the following sections.

3. UNBALANCED PARALLEL ENCODING

For estimating and balancing the workload in an encoder, a parallel algorithm was selected from [4] based on a multiple-master-multiple-slave (SMM) configuration. In this algorithm, each frame data is spatially parallelized across the slaves, and the communication tasks of MB distribution and collection are parallelized by multiple masters. The algorithm was theoretically modeled on a completely load balanced scenario by considering the delays through different stages of the coding and the overall critical path. It was also implemented on a 32-processor IBM SP2 supercomputer using a fixed number of MB approach (unbalanced), where the actual delays through each stage and the overall speedup were measured [4]. The mean results are depicted in Fig. 4, where the measured frame rates (dotted line) are obtained from coding the "table tennis" NTSC sequence. The discrepancy between the two curves is expected with the modeled mean frame rate higher because of the balanced load assumption.

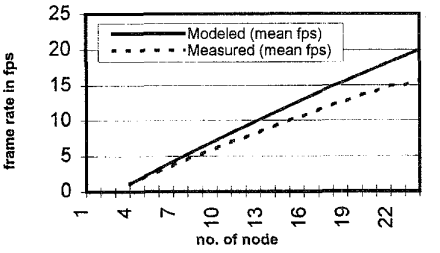

Fig. 4: Modeled and measured frame rates

To illustrate the point of unbalanced workload, Fig. 5 depicts the computing delay per processor for the 20th frame of the video. According to the measured data, the average delay is 37 ms. With the longest delay being 50.1 ms, meaning that it is 35% higher in delay compared with the theoretical case.

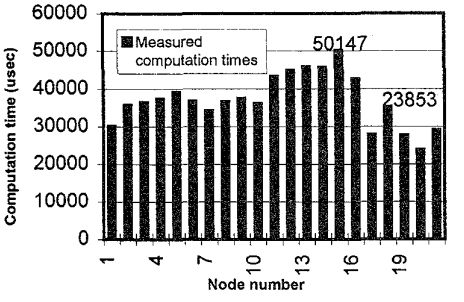

Fig. 5: Measured delays (20th frame of "table-tennis")

4. LOAD BALANCED ENCODING

As can be seen in Section 3, an unbalanced workload across the processors can significantly reduce the performance of the encoder. On the other hand, to achieve a balanced workload scenario, two issues as discussed in Section 2 are involved. First, the scheduling of the workload is to be optimized on computing delay; and second, the computing delay of each MB must be known or estimated. These two issues will be dealt with in this section, with the first one first.

Workload Scheduling

Let us assume that the computing delay, t_k, of the k^{th} MB is somehow known for k=1, ...N. The total computing delay of the frame is thus $\sum_{k=1}^{N} t_k$. From this, the balanced computing delay per processor is given by

$$t_{balanced} = \frac{1}{n}\sum_{k=1}^{N} t_k . \quad (3)$$

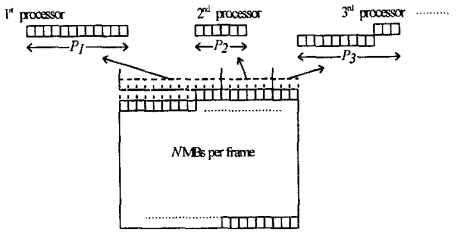

Fig. 6: Allocation of MB according to computing delays

The concept of the proposed algorithm is built upon the continuous comparison between the computing delay per current allocation versus the computing delay remaining. The aim of the comparison is to ensure the current allocation converges to $t_{balanced}$. In reality, due to the MB locality constraint, the allocation must involve consecutive MB as depicted in Fig. 6, and therefore the eventual allocation can only be closed to $t_{balanced}$, but not exactly the same.

The balancing process begins with assigning two indices: startIndex[j] and endIndex[j] to represent the starting and ending location of MB assigned to the j^{th} processor. From this, the computing delay of the j^{th} processor is given by the sum of delays between the two indices. Assuming the MB allocation starts from the 1st processor to the j^{th} processor, where there are n-j processors still to be allocated, the remaining computing delay per processor is given by Eqt. (4) and depicted in Fig.7. This should be similar to t_j in the balanced case.

$$t_{remain} = \frac{1}{n-j}\sum_{i=endIndex[j]+1}^{N} t_i \quad , \quad \text{for } j \neq n \qquad (4)$$

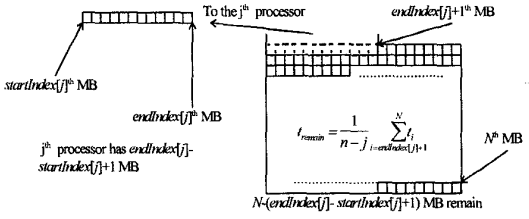

Fig. 7: Load balancing the Macroblocks across the processors

Now considering the algorithm in detailed, for the 1st processor, startIndex[1] is 1 and endIndex[1] is $1+\lceil N/n \rceil$, giving $t_1 = \sum_{i=1}^{1+\lceil N/n \rceil} t_i$ and $t_{remain} = \frac{1}{n-1}\sum_{i=\lceil N/n \rceil+2}^{N} t_i$, initially. With this, the workload in the 1st processor is not balanced. To make it balanced, t_1 and treamin are compared such that if t_1 is larger than tremain then the last MB in the 1st processor is reallocated back to the MB pool; else the next MB from the pool is allocated to the 1st processor. The comparison continues until t_1 and t_{remain} are closest to each other. There is no restriction as to whether t_1 should be slightly larger or vice versa. The very same sequence of steps repeats for the remaining processors with startIndex[j] = endIndex[j-1]+1 and endIndex[j] = startIndex[j] + $\lceil N/n \rceil$.

Computing Delay per MB

On this issue, to predict the computing delay per MB in frame i, $\hat{t}_{k,i}$, a delta prediction with error feedback factor formula is used which is given by

$$\hat{t}_{k,i} = \left| \hat{t}_{k,i-1} + a \cdot (t_{k,i-1} - \hat{t}_{k,i-1}) \right|, \qquad (5)$$

where $t_{k,i-1}$ and $\hat{t}_{k,i-1}$ are the measured and predicted computing delays of the k^{th} MB in frame i-1, respectively; and a is the feedback scaling factor. As both $t_{k,i-1}$ and $\hat{t}_{k,i-1}$ are known after coding frame i-1, by choosing an appropriately, $\hat{t}_{k,i}$ of frame i can be predicted. The measured and predicted delays of frame i is used for predicting the delays of frame i+1.

5. SIMULATION RESULTS

The load balancing algorithm described in Section 4 was simulated using 39 frames of the table-tennis sequences with 330 MB per frame (NTSC). Based on the same parallel algorithm presented in Section 3, the simulation was implemented on 24 processors of the IBM SP2 installed at the University of Hong Kong. The simulation was conducted under two sets of conditions. First, measured computing delays of frame i are used for load balancing of frame i. This enables us to study the behavior and performance of the load balancing aspect of the algorithm. Second, predicted computing delays (Eqt. (5)) are used for load balancing, allowing us to investigate the behavior of the prediction aspect and the effect of prediction error.

Case 1: Measured frame *i* delays for frame *i* balancing

The resulting frame rate per second including the overhead spent on performing the balancing are depicted in Fig.8, as the dotted line. The solid line with circle markers represents the modeled case where the plain solid line represents the measured unbalanced case. From this figure, a number of points can be observed. First, the frame rate of the balanced case is closed to the modeled (theoretical) frame rate. The difference is due to the overhead required for performing the load balancing, and

the imbalance due to the constraints of MB locality. However, the closeness of these two curves shows the effectiveness of the balancing algorithm, and it also implies that the modeled frame rate is an accurate theoretical upper bound. For small n, the difference between the two is minimal. Second, the frame rate improvement from the unbalanced case is substantial, particularly for large *n*. Such improvement can be explained in Fig.9(a), where the computing delays per processor of the 20th frame are plotted. In this case, the worst case delay is 38.7 ms (critical path), and the best case delay is 35.1 ms. This is just 4.5% higher than the ideal case as compared with 35% in the unbalanced case.

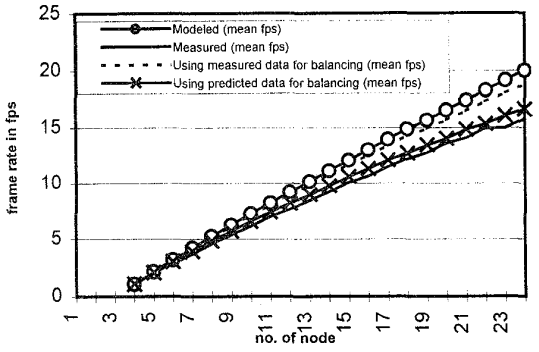

Fig. 8: Frame rates

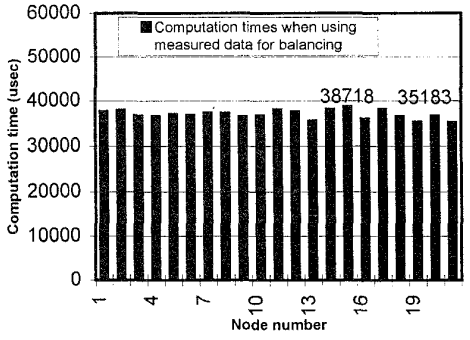

Fig. 9 (a): Balanced computation delays: measured delays

Case 2: Predicted frame *i* delays for frame *i* balancing

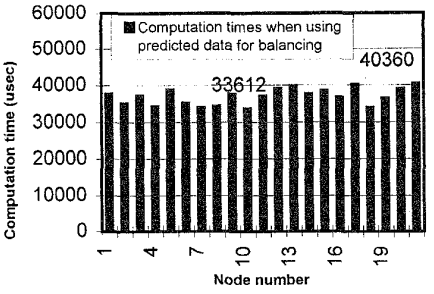

Fig. 9 (b): Balanced computation delays: predicted delays

When the computing delays are predicted using Eqt. (5), the resulting frame rate per second including the overhead spent on performing the balancing are depicted in Fig.8, as the solid line with × markers. In this case, the improvement due to balanced load is not as large as Case 1, not because of the balancing of workload, but due to the inaccuracy in prediction. As the allocation of MB to the processors is based on the predicted delays, any errors in the prediction is reflected in the overall frame rate. As a result, the improvement is about 1 fps rather than 3 fps in previous case. This can be further explained in Fig.9(b), where the computing delays per processor of the 20th frame is again plotted. It can be seen that the critical path is now 40.3 ms, or almost 11% higher than the ideal case. Comparing this with the unbalanced case, the balanced critical path is only 80% of the original, which is a significant improvement.

6. CONCLUSIONS

For future real-time video phone and conferencing applications, multiprocessing implementation seems inevitable. The very high computing resource offered by most multiprocessing platforms makes it all the more important to design fast and efficient parallel algorithms in achieving the ultimate. In this paper, through the use of a data parallel H.261 coding algorithm, we can demonstrate the positive effect of having the computing workload balanced. The improvement in overall coding performance can be substantial if the critical path of the computation is reduced. The price to pay is the overhead required by the balancing, and there may not be an optimally balanced case under the MB locality constraint. Furthermore, the accuracy of delay prediction is vital to how much improvement can be achieved. Future research in this area will be focused on reducing the prediction error.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Computer Centre of The University of Hong Kong, whom provides the multi-processing environment of IBM SP2 Supercomputer for simulations of the project.

REFERENCES

[1] F. Sijstermans & J. Van der Meer, "CD-I Full-motion Video Encoding on a Parallel Computer", Communications of the ACM, Vol.34, No.4, 1991, pp.81-91.

[2] M. Akramullah, I. Ahmad, Ming L. Liou, "A Portable and Scalable MPEG-2 Video Encoder on Parallel and Distributed Computing Systems", SPIE Proc. On Visual Communications and Image Processsing, 1996, pp.973-984.

[3] I. Agi & R. Jagannathan, "A Portable Fault-tolerant Parallel Software MPEG-1 Encoder", Multimedia Tools and Applications, 2, pp. 183-197, 1996.

[4] N. H. C. Yung & K. K. Leung, "Parallelization fo the H.261 video coding algorithm on the IBM SP2 multiprocessor system", to be presented in the IEEE ICA^{3}PP, 1997.

[5] Kai Hwang and Zhiwei Xu, "Scalable Parallel Computers for Real-Time Signal Processing", IEEE signal processing magazine pp. 50-66, July 1996.

[6] ITU-T Draft H.261. Line Transmission of Non-telephone Signals. Video Codec for Audiovisual Services at p×64 kbits.

[7] A. Murat Tekalp, "Digital video processing", Prentice Hall PTR, Prentice-Hall, Inc., 1995.

[8] A. C. Huang & J. L. Wu, "New Generation of Real-time Software-based Video Codec: Popular Video Coder II (PVC-II)", IEEE Transactions on Consumer Electronics, Vol. 42, No. 4, Nov. 1996, pp. 963-973.