Generalized parallelization methodology for video coding

K.K. Leung* & N. H. C. Yung
Department of Electrical & Electronic Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong SAR

ABSTRACT

This paper describes a generalized parallelization methodology for mapping video coding algorithms onto a multiprocessing architecture, through systematic task decomposition, scheduling and performance analysis. It exploits data parallelism inherent in the coding process and performs task scheduling base on task data size and access locality with the aim to hide as much communication overhead as possible. Utilizing Petri-nets and task graphs for representation and analysis, the method enables parallel video frame capturing, buffering and encoding without extra communication overhead. The theoretical speedup analysis indicates that this method offers excellent communication hiding, resulting in system efficiency well above 90%. A H.261 video encoder has been implemented on a TMS320C80 system using this method, and its performance was measured. The theoretical and measured performances are similar in that the measured speedup of the H.261 is 3.67 and 3.76 on four PP for QCIF and 352×240 video, respectively. They correspond to frame rates of 30.7 frame per second (fps) and 9.25 fps, and system efficiency of 91.8% and 94% respectively. As it is, this method is particularly efficient for platforms with small number of parallel processors.

Keywords: Parallel coding, Petri-net, H.261, H.263, speedup, efficiency

1. INTRODUCTION

The proliferation of video applications demands a better telecommunication infrastructure as well as more advanced technology for video storage, coding and manipulations. The emerging technology being applied to services and consumer products such as VCD, DVD, VoD, digital TV and video phone, are offering multimedia communication, information access and entertainment. A key factor for the success of these applications lies in the way video is coded. In the past decade, various video coding standards were introduced in order to focus effort in the development of technology and algorithms. The International Telecommunication Union (ITU) issued the H.261 recommendation in 1990, which is designed to standardize the video codec for audiovisual services at p×64 kbits[1]. Three years later, the MPEG-1 coding standard for moving pictures to be stored in digital storage media was announced by the International Organization for Standardization (ISO) [2]. In 1995, built upon the earlier H.261, the H.263 standard was recommended by ITU to deal with video coding for low bitrate communication[3]. In the same year, the ISO introduced the MPEG-2 standard for generic coding of moving pictures and audio for applications including digital storage, television broadcasting and communication[4]. In 1997, ISO also announced the MPEG-4 standard for integrating the production, distribution and content access paradigms of digital TV, interactive graphic applications and World Wide Web[5], and the MPEG-7 for standardizing descriptions of various types of multimedia information to allow fast and efficient searching of such information[6].

Apart from these coding standards, there are other coding methods[7-9] developed for good coding efficiency and better video quality. No matter how the coding is done, there is an ever increasing demand of computation resources for real-time performance, which is generally accepted that single processor performance is unable to meet such tremendous computation requirement. As a result, the trend is to use parallel systems with multiple CPU, fast cache memory and high performance inter-processor communication networks or buses. In reality, supercomputers are frequently used for experimentation and verification of parallel coding algorithms, in which dedicated switching networks and proprietary hardware are used. On the other hand, advances in low cost multiple digital signal processor (DSP) desktop systems open up opportunities for real-time video coding with small number of processors. However, the exploitation of processing power of these systems determines, to a large extent, the resulting performance. Therefore, reliable and consistent performance stems from a good parallelization methodology with systematic approach.

Existing examples of implementations of the H.261, H.263, MPEG-1 and MPEG-2 on various platforms can be broadly classified into three categories: supercomputer[10-14], network of workstations (NOW)[15-17] and dedicated system incorporating

* Correspondence: Email: kkleung@eee.hku.hk; Tel: 852-2859-2685; Fax: 852-2559-8738

Part of the IS&T/SPIE Conference on Visual Communications and Image Processing '99 • San Jose, California • January 1999
SPIE Vol. 3653 • 0277-786X/98/$10.00

DSP[18-21]. In terms of parallelization technique, some used data parallel approach with spatial[11,21] , temporal[12,17] or both[10,13]. Only a few implementations used functional parallelism on dedicated hardware[20]. From these examples, it is found that on supercomputer, real-time performance is often achieved using a large number of processors and the system efficiency (speedup/nodes) ranges from 32%[10] to 40%[11]. On NOW platforms, implementations are found with higher efficiency (62%) with smaller number of processors (12-16), but the actual frame rate is usually low (3-4.5 fps)[17]. On dedicated DSP systems, ignoring the simulation cases, an efficiency of 81% was reported in an implementation of the H.263, at the frame rate of 4.26 fps[18]. As not many can have exclusive access to supercomputer for the purpose of video coding, the focus should be placed on platforms such as NOW or parallel DSP. Between the two platforms, parallel DSP seems to offer higher efficiency and reasonable frame rate. As the four coding standards share a common algorithmic framework, it would then be logical to develop a methodology that exploits the parallelism and the common framework of coding, aiming at implementing it on a small number of parallel DSP.

In this paper, we present a parallelization methodology based on systematic task decomposition, scheduling and performance analysis. The method exploits data parallelism inherent in the coding process and performs task scheduling according to task data size and access locality with the aim to hide as much communication overhead as possible. Using Petri-nets and task graphs for representation and analysis, the method enables parallel video capturing, buffering and processing with extremely low communication overhead. The theoretical speedup analysis indicates that this method offers excellent communication hiding, resulting in system efficiency well above 90%. The practical implementation of a H.261 video encoder on a TMS320C80 system shows that the method has measured performance figures very similar to the theoretical prediction. The only difference observed between the theoretical and measured data is the program control overhead that has not been accounted for in the theoretical model. Even with this, the measured speedup of the H.261 is 3.67 and 3.76 on four PP for QCIF and 352×240 video, respectively, which correspond to frame rates of 30.7 frame per second (fps) and 9.25 fps, and system efficiency of 91.8% and 94% respectively[22].

This paper is organized as follows. Section 2 describes the parallelization method and the estimation of computation and communication delays. Section 3 demonstrates how the method is applied to implementing the H.261 encoder on the TMS320C80. Section 4 presents the measurement conditions as well as the detailed results and discussions. This paper is concluded in Section 5.

2. PARALLELIZATION METHODOLOGY

2.1 Speedup and efficiency

The ideal case of parallelization is the embarrassingly parallel algorithm in which a problem can be decomposed into any number of tasks with no data dependency between them. When these tasks are mapped onto N_P number of processors in equal partition of workload, all the processors start and complete their work simultaneously without any idling and data communication overhead. The resulting speedup is N_P since the execution time is reduced by N_P times. If we define system efficiency as the ratio between speedup and N_P, such parallelization is said to have 100% system efficiency. In reality, there always exists certain sequential component of the problem that cannot be decomposed into parallel tasks. Moreover, some of the parallel tasks may have data dependency among them that introduces inter-processor communication and synchronization. Furthermore, task decomposition and scheduling create programming overhead that can be substantial too. Adding these up, the true performance of a parallel implementation depends on how well the parallel component can be exploited

2.2 Parallelisation issues in video coding algorithms

Existing video coding standards rely on the reduction of temporal and spatial data redundancy existing among digital video data. Temporal data redundancy corresponds to data correlation between pixels across different frames and it is reduced by motion compensation between frames. In a MB, the residue pixels, after subtracting by the predicted pixels from the reference frames, are usually smaller in magnitude than the original pixels. Data compression is achieved by coding the residue data and the motion vectors. Some standards allow more than one motion vector per MB by distinguishing motion vectors between forward and backward directions and between odd and even picture field references.

To form the predicted MB, motion estimation is usually performed. Assume that the current and reference frame data are supplied from a global data server (DS) such as a video capturing processor, motion estimation is a process of searching for the closest MB from the reference frames within the search area. The search area is a bounded and enlarged area in a spatial

position offset from the position of the currently coded MB. Different standards have different search area sizes. As long as the data of the MB and the search area data are available, motion estimation can be performed on the MB. Theoretically, there is no specific ordering of the MB's in a picture regarding motion estimation, it can be parallelized over all the MB's in the picture if so desired.

After motion compensation, a frame of residue pixels is obtained. It is divided into a number of 8×8pixel blocks as the basic unit for transform coding, quantization, zigzag traversal and variable length coding. As long as the block of pixels or motion compensation residue is available, these coding steps can be carried out. Similar to motion estimation, there is no specific ordering of the blocks in a frame regarding these functions too. Therefore, in theory, the main body in the coding of a frame can be parallelized spatially with granularity of MB's as long as the required MB data and reference frame data is available.

Apart from the main body, there is a final step in coding a frame, i.e. the generation of the output bit stream. This step includes the construction of the bit stream structure by concatenation of headers of various levels and the coding results of transformed coefficients. These headers often use differential coding in such a way that the previous or neighboring headers have to be referenced. For this reason, the bit stream construction is inherently sequential. In the following discussion, we consider the bit stream construction for a frame to be a single non-divisible task called header-VLC.

2.3 Task decomposition

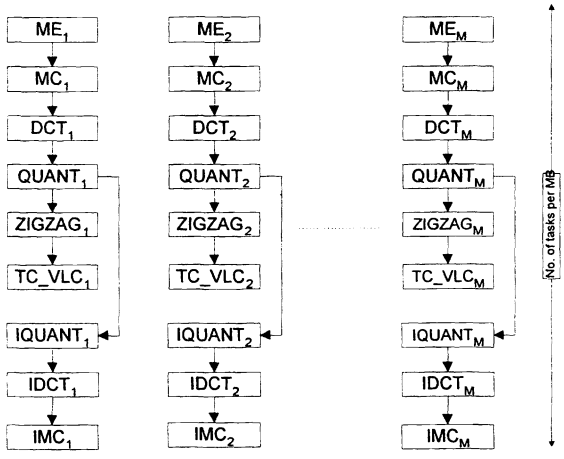

Fig. 1. Task graph for video coding

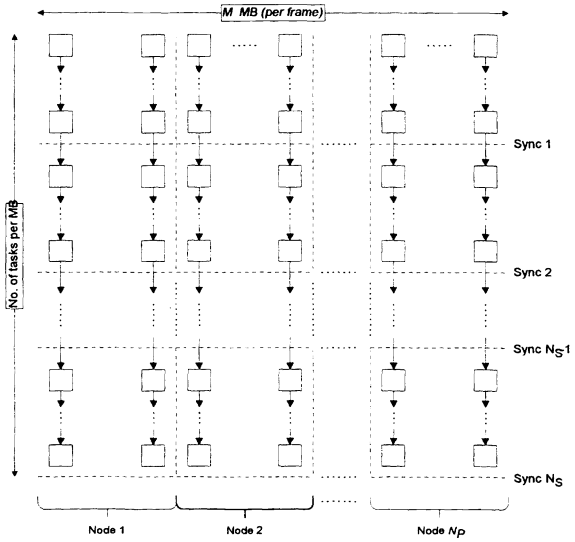

Fig. 2. A task allocation scheme

In this paper, a task is defined as a set of data together with the function that operates on the data. The data includes the variable space of the input, output and working area of the function. For example, a task doing motion estimation for a MB has the input MB and the search area as the input, the motion vector and other error statistics as output. The function is to find the MB in the search area having the smallest sum-absolute-error with the input MB. The formation of tasks follows 3 constraints. First, the data size must be upper bounded by the processor cache size. Second, the task computation time should be larger than the communication time for the task data. Otherwise, it would not be cost-effective to have the task executed in a remote processor. Third, the function should possess certain generalized meaning. With a clear definition of the interface and function for the task, then it can probably be re-used in a class of similar applications. In general, if these 3 criteria cannot be satisfied simultaneously, the first two take precedence over the third. Fig. 1 depicts the tasks with common video coding functions. It consists of 9 tasks per MB in each column. The *arc* between two tasks represents the precedence relation between them. For a frame containing M MB's, there are $9 \times M$ tasks per frame, where there is no precedence constraint between tasks of different MB's.

2.4 Task memory access and allocation

The mapping of tasks to the processors has two constraints. The first is the inter-task precedence relation, which is mandatory for correctness of the parallel implementation. When tasks having precedence relation are mapped onto different

processors, there is synchronization created between the processors to ensure the precedence constraint. Since synchronization introduces idling when some processors are ready earlier than the others, it should be reduced for the sake of efficiency. The second constraint concerns the way of hiding communication delay. If each pair of successive tasks executed by a processor can be accommodated in the cache memory of it, then it allows the use of triple buffering scheme to hide the communication overhead. By this scheme, when a task is executed, the output data from the previous task is saved and the next task input data is loaded. Once the computation of the current task is completed, the processor can switch to the next task and start it as soon as input data is ready. Fig. 2 depicts the task allocation scheme in which the M MB's are decomposed into N_P subsets with each subset, containing M/N_P MB's, being allocated to a processor. The horizontal dashed lines represent synchronization points. Between two synchronization points, there is no inter-processor interaction and each processor executes a sequence of tasks on its own.

2.5 Communication hiding

Let N_F be the number of frames to be coded as a unit. For H.261, N_F equals one. For H.263, it can be one or two depending on whether PB-frame option is used or not. For MPEG-1/2, N_F is the number of B-frames between successive I- or P-frames, plus the trailing I- or P-frame. The state transition of capturing, buffering and processing of each set of N_F frames can be represented by the Petri-net in Fig. 3.

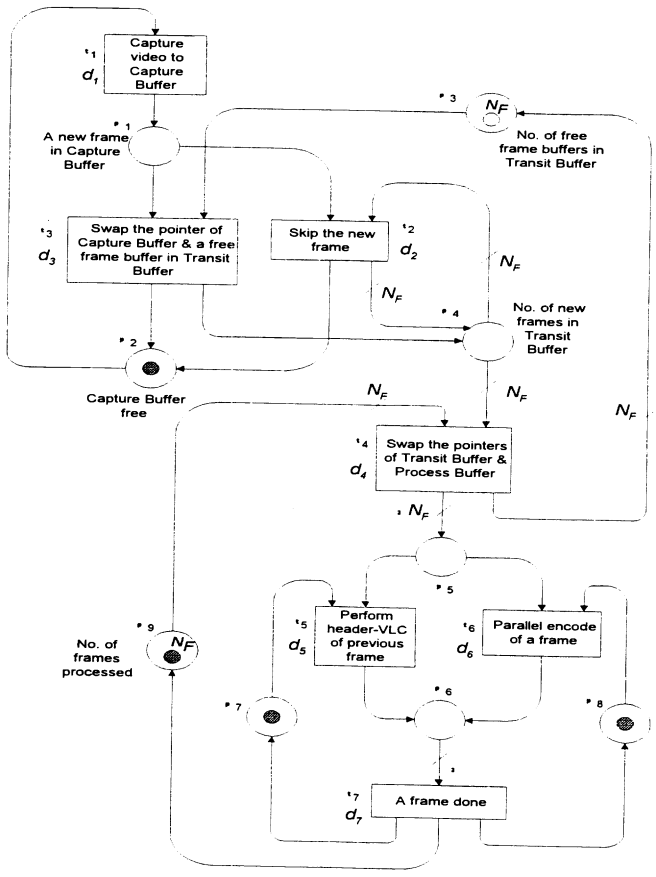

Fig. 3. Petri-net representation of the hiding scheme

Under this convention[23], a *place* (p_i) is represented by a circle and a *transition* (t_i) by a rectangular box. A transition may have a delay (d_i) associated with it; otherwise, it is represented by a *bar*. The *arcs* between places and transitions carry a weight of unity unless specified. When all the inlet places of a transition contain the specified number of tokens, then the transition is enables for firing. Upon firing, the input tokens are consumed while new tokens are generated at the outlet

places accordingly. The set of tokens in the net form a *marking*. Fig. 3 shows the initial marking with frame capturing enabled.

To handle the captured frames for processing, the scheme relies on three sets of buffers. The first, called *Capture Buffer* (CB), has a size of a frame that holds the new frame being captured. The second is the *Transit Buffer* (TB) with a structure containing N_F frame buffers. It is used to hold the N_F recently captured frames. The third buffer is called *Process Buffer* (PB) in which the N_F frames being processed are stored. Once a new frame is captured in CB, the pointers of CB and a free frame of TB are swapped. In this way, the next captured frame is then placed into the free buffer pointed to by CB, while recently captured frames are accumulated in TB. After N_F new frames are accumulated in TB, their pointers are swapped with that of PB for processing. Throughout the whole buffering scheme, there is no pixel data copying. The delay for pointer swapping is considered insignificant as compared with the coding time.

When a set of N_F frames are swapped into PB, there are $2\times N_F$ tokens in p_5, thus, enabling t_5 and t_6 to fire N_F times. These two transitions correspond to the Header-VLC task (t_5) and the parallel encoding (t_6) of a frame. After the encoding and Header-VLC are done to the N_F frames, it fills p_9 with N_F tokens and allows the next set of N_F frames to come in. The delay in the output bit stream is $2\times N_F$ frame. More detailed analysis shows that new frame is skipped (t_2) only if the processing time is longer than the frame capturing time. The resulting frame rate is upper bounded by the capture rate and is determined either by d_5 or d_6 depending on how well t_6 is parallelized. Fig. 4 depicts the expansion of t_6 into rows of parallel transitions. The parallel transitions in each row represent the parallel processing in the N_P processors. As they proceed, there are N_S synchronization points in time. S(i,j) represents a sequence of tasks performed in the j^{th} processor before the i^{th} synchronization point. Down one more level, the task sequence S(i,j) is executed according to the Petri-net shown in Fig. 5. $N_T(i,j)$ is the number of tasks in sequence S(i,j). Before a task is executed, its input data is loaded into the processor cache from a global data server. After execution, the result is saved back to the server. The loading and saving are overlapped in time with the computation.

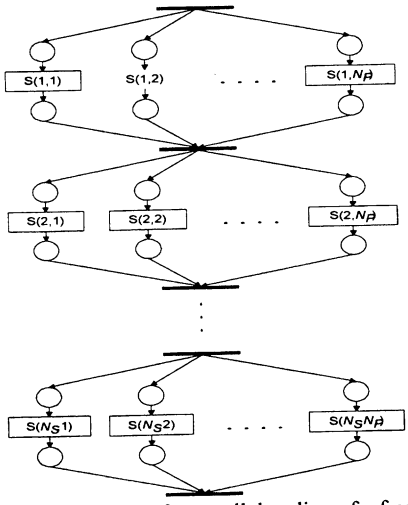

Fig. 4. Petri-net for parallel coding of a frame

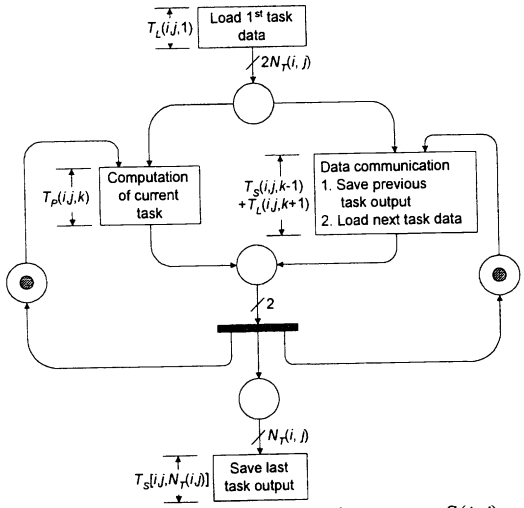

Fig. 5. Petri-net of task sequence S(i, j)

2.6. Theoretical speedup estimation

From Fig. 3, ignoring the time delays of d_2, d_3, d_4 and d_7 as they involve only swapping of pointers and skipping of frames, the significant delays are d_1 (frame capture), d_5 (Header-VLC) and d_6 (parallel task execution). As they are executed in parallel, the resulting frame coding time is

$$T_f = \max\{d_1, d_5, d_6\}. \tag{1}$$

From the expansion of t_6, and denoting the delay of S(i, j) by $d(i,j)$, we have

$$d_6 = \sum_{i=1}^{N_S} \max_{j\in[1..N_P]} \{d(i,j)\}. \quad (2)$$

To determine $d(i,j)$, let us denote the computation delay, task loading delay and saving delay by $T_P(i,j,k)$, $T_L(i,j,k)$ and $T_S(i,j,k)$ respectively for the k^{th} task in S(i,j). From the expansion of S(i,j), $d(i,j)$ equals to the sum execution delays of $N_T(i,j)$ tasks in S(i,j) plus the leading task loading and trailing result saving delays, which is given by

$$d(i,j) = T_L(i,j,1) + T_S(i,j,N_T(i,j)) + T_P(i,j,1) + T_P(i,j,N_T(i,j)) + \sum_{k=2}^{N_T(i,j)-1} \max[T_P(i,j,k), T_S(i,j,k-1) + T_L(i,j,k+1)]. \quad (3)$$

Therefore, the resulting frame time is expressed as

$$T_f = \max\left\{ d_1, d_5, \sum_{i=1}^{N_S} \max_{j\in[1..N_P]} \left\{ \begin{array}{l} T_L(i,j,1) + T_S(i,j,N_T(i,j)) + T_P(i,j,1) + T_P(i,j,N_T(i,j)) \\ + \sum_{k=2}^{N_T(i,j)-1} \max[T_P(i,j,k), T_S(i,j,k-1) + T_L(i,j,k+1)] \end{array} \right\} \right\}. \quad (4)$$

Given the sequential frame time $T_1 = T_f\big|_{N_p=1}$, the speedup is the ratio between T_1 and T_f.

From the above equations, we observe that there exists two sources of performance degradation. First is the communication overhead in the execution of S(i,j). From Eqt. (3), there is a constant delay due to the initial task loading and the trailing task saving, and there is further overhead if the task communication delay is larger than the task computation delay. The former delay cannot be hidden as such, but the latter communication delay can be hidden if $T_P(i,j,k) \geq T_S(i,j,k-1) + T_L(i,j,k+1)$ which depends on the communication channel bandwidth and the data size. The second source of degradation is the imbalance in workload across the processors. From Eqt. (2), it is the sum of a series of maximum delay, each represents a critical path between two synchronization points, that determines d_6. Different critical paths imply idling in some of the processors.

To further simplify the estimation, we assume the computation delay ($T_P(i,j,k)$) to be a random variable with Gaussian distribution. The mean and variance of the distribution are estimated from the serial execution measured time. The mean and variance of $T_P(i,j,k)$ are used to determine T_1 and T_f when the communication delays are known.

2.7. Estimation of communication delay with contention

The communication delay is estimated by measuring the time to transfer data messages of different sizes. We find that a linear model with a constant channel bandwidth and initial setup delay is applicable for the processor-to-processor communication in IBM SP2[24] and TMS320C80[25].To send a message of size M_C over a channel with bandwidth W and initial setup time T_0, the time taken, T_C', is given by

$$T_C' = \frac{M_C}{W} + T_0. \quad (5)$$

As the frame data is served centrally, it is reasonable to expect a queue of requests pending on the server, which is served one by one. Assume a statistical queuing model[26], let n be the number of requests in the queue, $\lambda(n)$ be the request arrival rate and μ be the request service rate, where both $\lambda(n)$ and μ are random variables with exponential distribution. Further assume that all the processors generate requests at the same rate λ_o and that processor with a pending request in the queue does not generate request until its pending request is served. Then the arrival rate at the queue is proportional to the number of processors that do not have pending request in the queue, or

$$\lambda(n) = (N_P - n) \cdot \lambda_0. \quad (6)$$

Let p_n be the probability of having n requests in the queue. At equilibrium, there is a set of local balance equations by equating the sum of flow of probability flux between adjacent states to zero, which are given as

$$\mu \cdot p_n = \lambda(n-1) \cdot p_{n-1} . \tag{7}$$

Solving this recursive equation for p_n gives

$$p_n = \left[\prod_{i=1}^{n} \frac{\lambda(i-1)}{\mu}\right] p_0 = \left[\left(\frac{\lambda_0}{\mu}\right)^n \frac{N_P!}{(N_P-n)!}\right] p_0 . \tag{8}$$

To calculate p_0, we equate the sum of all probabilities to 1, which gives

$$p_0 = \frac{1}{1+\sum_{n=1}^{\infty}\prod_{i=1}^{n} \frac{\lambda(i-1)}{\mu}} = \frac{1}{\left[\sum_{n=0}^{N_P}\left(\frac{\lambda_0}{\mu}\right)^n \frac{N_P!}{(N_P-n)!}\right]} . \tag{9}$$

By Little's Law[27], the mean delay of a request in the queue is given by

$$T_C = \frac{\bar{n}}{\bar{\mu}} = \frac{\sum_{n=1}^{\infty} n \cdot p_n}{\sum_{n=1}^{\infty} \mu \cdot p_n} . \tag{10}$$

From Eqt. (10), T_C is the message transfer delay under contention. In general, T_C is greater than T_C' especially for large N_P. Fig. 6 depicts the ratio of T_C to T_C' versus N_p at different μ/λ_o. For small μ/λ_o, the ratio T_C/T_C' increases almost linearly and is large. It is because the request rate is larger than the service rate, resulting in long queue length. For large μ/λ_o, the request rate is smaller than the service rate, hence the server is able to keep the queue length short, and T_C/T_C' is small.

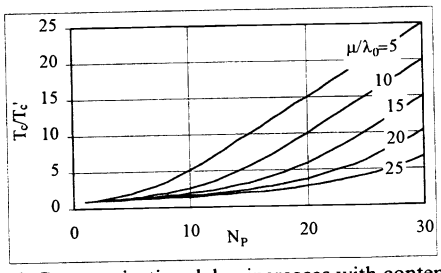

Fig. 6. Communication delay increases with contention

To apply the above to the estimation of communication delay, first $T_S'(i,j,k)$ (or $T_L'(i,j,k)$) is calculated according to the message size and channel characteristics without contention. Then the equations are applied with N_P set to the number of processors, the request generation rate, λ_0, set to the reciprocal of mean task execution time, and the request service rate, μ, assigned the reciprocal value of the service delay $T_S'(i,j,k)$ (or $T_L'(i,j,k)$). During the start of a task sequence just after synchronization, all the processors access the data server almost simultaneously, causing a transient period with exceptional high contention. In this estimation, the initial data loading time $T_L(i,j,1)$ and the final saving delay $T_S(i,j,N_T(i, j))$ are multiplied by N_P to account for this transient period.

2.8. Limitations and applicability

The methodology assumes independent processing between the MB's. Under some situations, this may not hold. For instance, in H.263, when the long vector and Unrestricted Motion Vector options are used simultaneously, the motion estimation for a MB takes place with the search area depending on the motion vector predictor. This predictor is obtained from the motion vectors of three neighboring MB's, implying that their motion estimation tasks have been done beforehand. So, it imposes certain spatial ordering in which motion estimation is done to the MB's.

Concerning bit-rate control, there is no specification from the standards on how it is done. A commonly adopted way is to adjust the quantizer(s) during encoding based on the discrepancy between the number of bits generated in the bit stream and

the target bit budget. In this methodology, this information can be stored in the DS for processor access. The frequency of quantizer adjustment can be in MB, MB row or frame basis. However, for the particular case of H.263, there is a limit of ±2 on the quantizer relative to the left neighboring MB such that the validity of the quantizer is not known, until the neighboring MB quantizer has been determined. This restricts the rate control for the MB's in certain order.

Apart from the above limitations, this method does not impose any restrictions on the list of tasks performed when coding. Different coding standards may be represented by the task allocation scheme shown in Fig. 2, where precedence relationship could be accommodated using appropriate synchronization points. Similarly, the Petri-net representation as depicted in Fig. 3 can be applied to all four standards. The only difference between them would be t_6. Moreover, Fig. 4 and Fig. 5 are equally applicable to all cases disregarding the actual list of tasks performed in individual coding standards. Table 1 lists the possible tasks for each of the four coding standards. There may also be tasks such as coding mode determination, rate control and scalability options, which can be incorporated into the model without any restrictions.

H.261	H.263	MPEG-1	MPEG-2
ME	P-MB MV PREDICTION	P-MB ME	FRAME/FIELD ME
PREDICTION (MC)	P-MB ME	B-MB ME	DUAL-PRIME FRAME/ FIELD
ENC (DCT, QUANT,	B-MB MVD SEARCH	PREDICTION	ME
ZIGZAG)	P-MB PREDICTION	ENC	PREDICTION
TC_VLC	B-MB PREDICTION	TC_VLC	DCT TYPE ESTIMATION
RECONSTRUCTION	ENC	RECONSTRUCTION	ENC
(IQUANT, IDCT, IMC)	P-MB RECONSTRUCTION	HEADER-VLC	TC_VLC
HEADER-VLC	TC_VLC		RECONSTRUCTION
	HEADER-VLC		

Table 1. Possible tasks for the 4 video coding standards

3. IMPLEMENTATION ON THE TMS320C80

3.1. Development board and internal architecture

To verify the methodology, the implementations were carried out on the TMS320C80 Software Development Board[28] (SDB). The board consists of 8MB on-board memory called EXTMEM for storage of program and data, hardware for video frame grabbing, video display, audio and PCI interface to the host PC. Both video capturing and display can be done in real-time (30 fps) at different frame resolution. Inside the TMS320C80, there are four Parallel Processors (PP) for number crunching and one Master Processor (MP) for program control, system management and I/O. All of them access the EXTMEM and other hardware on the board through the Transfer Controller (TC). Data transfer in the form of packet transfer request is submitted to the TC and a queue of requests is serviced with a priority scheme. Data communication is handled separately by the TC, allowing overlapped computation and communication. Program execution by the processors is done via the on-chip cache memory. There is a total of 50KB of on-chip cache memory arranged as 25 2KB blocks. Each processor owns part of the cache for storing the recently used instruction codes and data, although access to the cache belonging to another processor is allowed. In particular, the data cache of the PP is managed by the application programmer. Any data transfer between the PP data cache and EXTMEM is done by explicit packet transfer through the TC.

The MP, 4 PP's, TC and cache blocks are connected by a cross-bar switch inside the TMS320C80. Through the cross-bar, the MP and PP can access its own cache in one cycle and other cache blocks within a couple of cycles. Inter-processor data transfer can be done by coordinated reading and writing of the on-chip cache. If large data transfer is required, it may be done via packet transfer to and from the EXTMEM. For synchronization, the processors can signal each other efficiently with the use of COMM register without burden on the TC or the cross-bar switch. The PP internal architecture[29] is designed with the purpose for enhanced image processing performance. To do that, it executes instructions with high degree of parallelism, delivering up to four 8-bit ALU operations, two 8-bit multiplication operations, and simultaneous transfer of two 32-bit data words between local cache memory and registers in a cycle. Nevertheless, the key for good performance lies in coding with optimized instructions and availability of data in the cache.

3.2. Implementation issues

Due to data access locality, the 9 tasks in Fig. 1 for each MB are grouped into two tasks: ME (motion estimation) and ENC (the other 8 tasks from MC to TCVLC). So, N_S equals to 2 with S(1,j) corresponds to a sequence of ME tasks and S(2,j) represents an ENC task sequence. The MB's in the frame is evenly distributed over the PP's. To start a ME task, it requires an input frame MB plus 9 MB's in the reference frame. Since neighboring MB's overlap their search area by 6 reference

MB's, we save substantial communication delay by executing ME tasks with the MB's arranged in raster scan order. When a ME task is executed, the next task is the right neighboring MB which requires to load 3 more reference MB's plus the one in the input frame. In this way, the minimum cache size required consists of (9+3) reference MB's plus 2 input frame MB's for triple buffering. The data being loaded for each task amounts to approximately 16×16×3+16×16 or 1024 bytes. For ENC task, there is no overlap in data access. It makes no difference to the performance with respect to the MB ordering. Before each ENC task, an input frame MB and a reference frame MB are loaded. After execution, the decoded MB is saved. So, the total data size is (16×16+2×8×8)×3 or 1152 bytes.

Table 2 lists the mapping of the Petri-nets onto the TMS320C80 system. The MP handles video capturing, system functions and Hander-VLC calculation. TC together with the EXTMEM act as a data server for input frame data, reconstructed frame data and output bit stream.

Transitions	Mapped units
t_1	VC (Video Controller)
t_2, t_3, t_4, t_5, t_7	MP
t_6	PP's & TC
$S(i,j)$ for i=1,2,...,N_S	PP(j) & TC
Task loading & saving	TC
Task computation	PP(j)

Table 2. Implementation scheme on the TMS320C80

Fig. 7. A tested QCIF sample

4. RESULTS AND DISCUSSIONS

4.1. Measurement criteria and conditions

Timestamps were used for performance measurement. All the processors reference to a common clock tick generated by the MP TIMER once every 10μsec. At each clock tick, an interrupt is generated to the MP which increments the clock tick count in the cache memory of each PP. As it is, this incurs interrupt overhead to the MP and access contention to the PP cache memory. A finer clock tick may be used but with more MP overhead and higher PP cache memory contention.

4.2. Serial performance

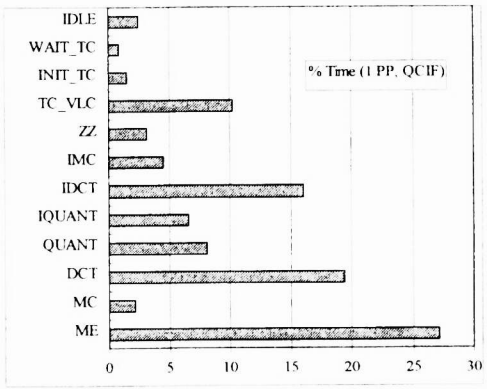

Fig. 8. Serial coding time break down

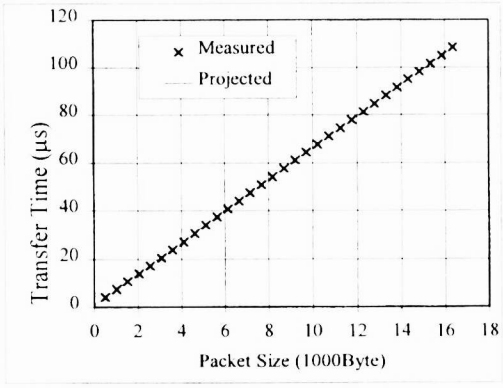

Fig. 9. TC data transfer time

As a baseline reference, a serial encoder was executed using one PP and the average frame time over 50 frames were obtained. A typical frame of the captured video is shown in Fig. 7. The measured frame time is 406.5ms (2.46 fps) for 352×240 video and 119.5ms (8.37 fps) for QCIF (176×240). Fig. 8 depicts the average percentage time breakdown. The most time-consuming part is ME as expected (27.1%), followed by DCT (19.4%), IDCT (16%) and TC_VLC (10.2%). The rest ranges from 8.1(QUANT), 6.5% (IQUANT) to just below 1%. There are also communication overhead times such as INIT_TC (1.5) and WAIT_TC (0.9%) which corresponds, respectively, to the initialization time of packet transfer table and the time to wait for packet transfer to complete in case the data communication time is longer than task computation time. Another overhead, IDLE (2.5%), is neither computation nor communication time. It is the idle time or waiting time for MP initialization before each frame. In this case, as only one PP is used, this 2.5% is mainly due to MP initialization.

The bandwidth of the TC without contention was estimated by measuring the time for transferring messages from EXTMEM to the on-chip cache and vice versa. For different message sizes, a number of measurements were conducted and the average time was taken. As depicted in Fig. 9, W and T_0 are estimated to be 153MB/sec and 0.8μsec respectively.

4.3. Parallel performance

The implementation was executed on up to 4 PP's and measurement was done. Fig. 10 to 12 depicts the frame rate, speedup and efficiency for coding of QCIF video. The predicted and ideal linear speedup performances are also plotted for comparison with the measured result. From Fig. 10, the frame rate rises almost linearly and reaches 30.7 fps at 4 PP's (N_P=4) with a speedup of 3.67. This linear shape is due to successful hiding of communication overhead by computation time. In fact, the mean computation time for ME and ENC tasks are around 300 μs and 800 μs respectively, while the estimated communication delays without contention are 9.5 μs and 13.4 μs for the two tasks respectively. So, there is a high possibility of totally hiding communication delay. We also find that the predicted speedup is slightly better than the measured one. This is due to program control overhead not taken into account in the theoretical performance model. As in Fig. 12, the predicted efficiency is above 90% up to 8 PP's. The measured one is over 90% up to 4 PP's, which is lower than the predicted. But this is still 10% better than any other reported results for small or large N_P.

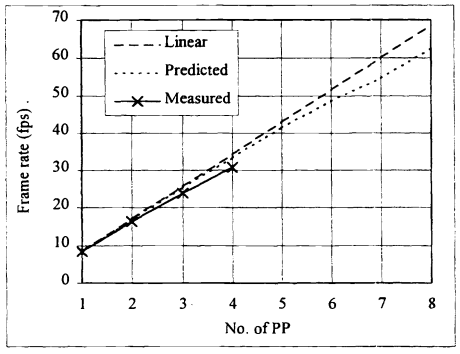

Fig. 10. Frame rate of QCIF

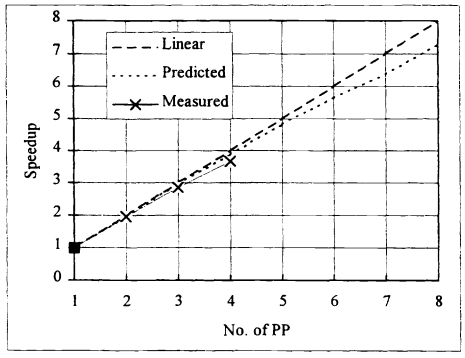

Fig. 11. Speedup of QCIF

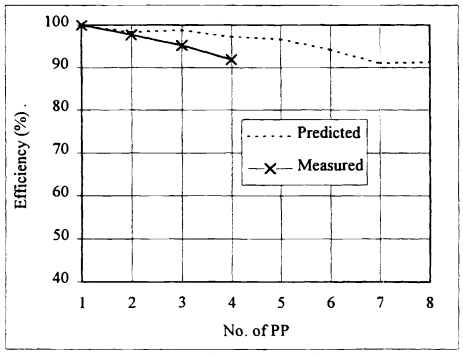

Fig. 12. Efficiency of QCIF

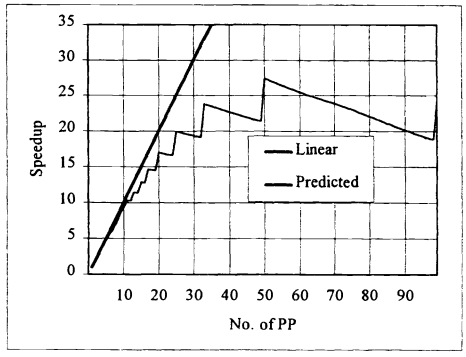

Fig. 13. Projected speedup for QCIF

To extend the prediction for high number of PP's, Fig. 13 depicts the predicted speedup up to 100 PP's. The speedup rises almost linearly up to N_P=10. Beyond that, it becomes stepwise as the small number of MB's, M=99, being integrally divided by N_P. Some processors are allocated one MB more than the others. Thus, there is uneven distribution of workload and idling time upon synchronization. The stepwise speedup may be smoothed and improved if the workload is balanced[30]. The effect is most adverse when N_P approaches M and each sequence of tasks contains only one or two tasks. In such case, the initial and final communication delay can be a significant overhead. A finer granularity of task decomposition is possible to give smoother speedup at large number of processors. However, the communication contention on TC increases with the number of PP's and splitting of task results in more duplicated data communication. From the current result, the TC should allow a finer granularity since the current computation time is far greater than communication delay (30-60 times). For more complicated functions, it may be necessary to split the tasks in order to meet the cache size limitation.

For 352×240, the measured and predicted performance have similar trend to that of QCIF. From Fig. 14 & 15, a frame rate of 9.25 fps (speedup=3.76) was obtained at N_P=4. It is predicted that 30 fps is achievable at around N_P=14, i.e. 4 C80 with the presence of parallelization overhead. The speedup at N_P=4 is better than QCIF due to the larger number of MB's in the 352×240 case. In fact, in both predicted and measured results, 352×240 has better speedup than QCIF. From Fig. 16, the predicted efficiency remains well over 90% up to N_P=8. The measured efficiency however, showed the same tendency as in QCIF in which the parallelization overhead becomes more significant with larger N_P.

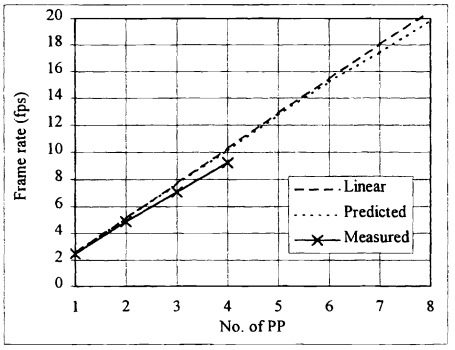

Fig. 14. Frame rate of 352×240

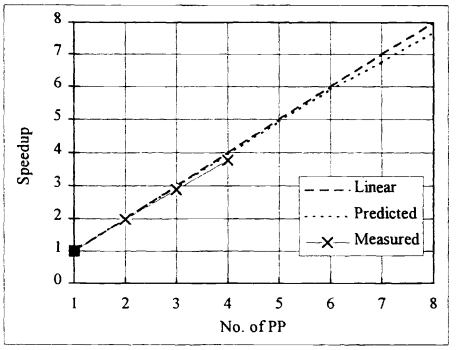

Fig. 15. Speedup of 352×240

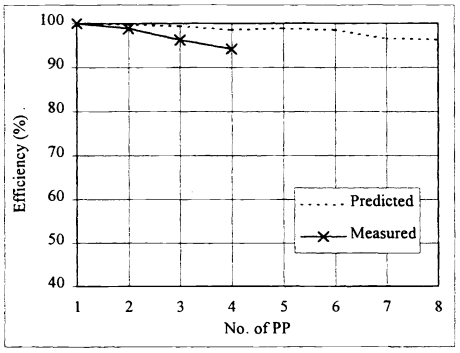

Fig. 16. Percentage efficiency of 352×240

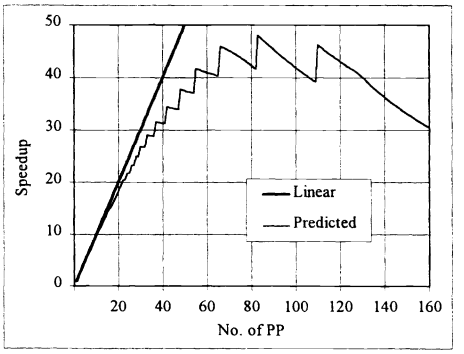

Fig. 17. Projected speedup for 352×240

The extended prediction in Fig. 17 rises almost linearly up to N_P=20 and then becomes stepwise. It attains a maximum of 48, as compared with 27.5 for QCIF. The reasons why this figure is larger than the QCIF case are firstly, for 352×240 video, there are more tasks in the sequence. As a result, the initial task loading and final saving delays in each task sequence constitute a smaller proportion of delay to the overall sequence execution time, giving a higher speedup. Secondly, as there is a synchronization point between the PP's at the end of each task sequence, the expected overall execution time is the expected maximum of the N_P sequence execution time. A large standard deviation in sequence exeuction time implies a large maximum time among N_P sequences and a large proportion of processor idling time.

5. CONCLUSION

In conclusion, a new parallelization methodology for video coding base on systematic task decomposition and scheduling has been successfully developed and implemented. With the aid of Petri-nets and task graphs, performance prediction is achieved. Data and task sizes are considered under the constraint of cache capacity so that substantial number of cache misses can be avoided. Also, with proper task scheduling, it enables sustained overlapping of data communication with computation by executing a sequence of tasks without inter-processor synchronization. Contention in data communication is also considered to give closer prediction of actual performance and enable refinement of implementation. As the only assumption for applicability, it highlights the importance of independence between MB's for processing in parallel. Apart from this requirement, no further restriction is imposed on the coding algorithm. Therefore, it is fair to assume that the methodology is equally applicable to the four standards. In fact, full implementation has been tested on H.261 and the H.263 standard was also implemented based on this method with very similar performance characteristics.

From the measured results, it can be observed that first, the predicted and measured performance are very similar. Second, using one TMS320C80, 30.7 fps and 9.25 fps were achieved for QCIF and 253×240 video respectively, with over 90% efficiency in both cases. This result is favorable compared with the other practical implementations (excluding simulations). Third, the absolute serial performance is due to optimized coding while the almost linear speedup is the result of the parallelization method. Fourth, this parallelization method is particularly suitable for small N_p. For QCIF, the measured speedup is almost linear for $N_p \leq 10$, whereas for 352×240 video, the measure speedup is almost linear for $N_p \leq 20$.

6. ACKNOWLEDGEMENT

The authors would like to express their sincerely gratitude to the financial support of the Texas Instrument Tsukuba Research and Development Center, Japan.

7. REFERENCE

1. "ITU-T recommendation H.261: video codec for audiovisual services at px64 kbits", International Telecommunication Union, 1990.
2. "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.
3. "ITU-T recommendation H.263: video coding for low bitrate communication", International Telecommunication Union, 1995.
4. "MPEG-2: Generic coding of moving pictures and associated audio", ISO/IEC 13818, 1995.
5. "Overview of the MPEG-4 standard", ISO/IEC JTC1/SC29/WG11 N1730, 1997.
6. "MPEG-7: context and objectives (v.5 – Fribourg)", ISO/IEC JTC1/SC29/WG11 N1920, 1997.
7. L. Torres & M. Kunt, *Video Coding: the second generation approach*, Kluwer Academic Publishers, 1996.
8. 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, pp. 963-973, Nov. 1996.
9. K. Li & H. Yuen, "A High Performance Image Compression Technique for Multimedia Applications", IEEE Transactions on Consumer Electronics, Vol. 42, No. 4, pp.239-243, May 1996.
10. F. Sijstermans & J. Van der Meer, "CD-I Full-motion Video Encoding on a Parallel Computer", Communications of the ACM, Vol.34, No.4, pp.81-91, 1991.
11. S. M. Akramullah, I. Ahmad, M. L. Liou, "Performance of Software-Based MPEG-2 Video Encoder on Parallel and Distributed Systems", IEEE Transactions on CSVT, Vol. 7, No. 4, pp. 687-695, Aug 1997.
12. K. Shen, L. A. Rowe, E. J. Delp, "Parallel implementation of an MPEG-1 encoder: faster than real time", SPIE Vol. 2419, pp. 407-418, Feb 1995.
13. K. Shen & E. J. Delp, "A spatial-temporal parallel approach for real-time MPEG video compression", Proceedings. Of the 25th International conference on parallel processing, pp.II100-II107, 1996.
14. N. H. C. Yung & K. K. Leung, "Parallelization of the H.261 video coding algorithm on the IBM SP2 multiprocessor system", Proceedings of the IEEE Int'l Conf. on Algorithm, Architectures for Parallel Processing, pp.571-578, 1997.
15. S. M. Akramullah, I. Ahmad, M. L. Liou, "Software-based H.263 video encoder using a cluster of workstations", SPIE Vol. 3166, pp. 266-273, Jul 1997.
16. Y. Yu, D. Anastassiou, "Software implementation of MPEG-II video encoding using socket programming in LAN", SPIE Vol. 2187, pp. 229-240, Feb 1994.
17. I. Agi & R. Jagannathan, "A Portable Fault-tolerant Parallel Software MPEG-1 Encoder", Multimedia Tools and Applications, 2, pp. 183-197, 1996.
18. H. Mooshofer, A. Hutter, W. Stechele, "Parallelization of a H.263 Encoder for the TMS320C80 MVP", ESIEE, Paris, SPRA339, Texas Instruments, Sept 1996.
19. W. Lee, J. Golston, R. J. Gove, Y. Kim, "Real-time MPEG video codec on a single-chip multiprocessor", SPIE Vol. 2187, pp. 32-43, Feb 1994.
20. T. Akiyama et al., "MPEG-2 Video Codec using Image Compression DSP", IEEE Trans. on Consumer Electronics, Vol.40, No.3, pp.466-472, 1994.
21. C. Bouville, P. Houlier, J. L. Dubois, I. Marchal, B. Thébault, M. Klefstad, "DVFLEX: A Flexible MPEG Real Time Video Codec", Proc. of IEEE Int. Conf. On Image Proc., ICIP'96, Vol. II, pp. 829-832, 1996.
22. K. K. Leung, *Parallelization methodology for video coding – an implementation on the TMS320C80*, Research report, Department of E. & E. Eng., The University of Hong Kong, May 1998.
23. David, Rene, *Petri nets and Grafcet: tools for modeling discrete event systems*, Prentice Hall, 1992.
24. C. B. Stunkel, et al, "The SP2 High-Performance Switch", IBM Systems Journal, Vol. 34, No. 2, pp.185-204, 1995.
25. *TMS320C80 (MVP) Transfer Controller User's Guide*, Texas Instruments, SPRU261, 1995.
26. Thomas G. Robertazzi, *Computer Networks and Systems - Queuing Theory and Performance Evaluation*, Springer-Verlag, 1994.
27. Little, J. D. C. (1961). A proof of the queuing formula L=λW, Operations Research, 9, 383-387.
28. *TMS320C8x Software Development Board Technical Reference*, Texas Instruments, SPRU178, 1997.
29. *TMS320C80 (MVP) Parallel Processor User's Guide*, Texas Instruments, SPRU110A, 1995.
30. N. H. C. Yung & K. C. Chu, "Load balancing algorithm for the parallel implementation of the H.261 video encoder", to be presented in the *IEEE SMC98*.