Adaptive parallel video coding algorithm

K. K. Leung[*], N. H. C. Yung, P. Y. S. Cheung
Department of Electrical & Electronic Engineering,
The University of Hong Kong, Pokfulam Road, Hong Kong SAR

ABSTRACT

Parallel encoding of video inevitably gives varying frame rate performance due to dynamically changing video content and motion field since the encoding process of each macro-block, especially motion estimation, is data dependent. A multiprocessor schedule optimized for a particular frame with certain macro-block encoding time may not be optimized towards another frame with different encoding time, which causes performance degradation to the parallelization. To tackle this problem, we propose a method based on a batch of near-optimal schedules generated at compile-time and a run-time mechanism to select the schedule giving the shortest predicted critical path length. This method has the advantage of being near-optimal using compile-time schedules while involving only run-time selection rather than re-scheduling. Implementation on the IBM SP2 multiprocessor system using 24 processors gives an average speedup of about 13.5 (frame rate of 38.5 frames per second) for a CIF sequence consisting of segments of 6 different scenes. This is equivalent to an average improvement of about 16.9% over the single schedule scheme with schedule adapted to each of the scenes. Using an open test sequence consisting of 8 video segments, the average improvement achieved is 13.2%, i.e. an average speedup of 13.3 (35.6 frames per second).

Keywords: Video coding, multiprocessor scheduling, inter-processor communication, data flow graph, pipelining, H.261

1. INTRODUCTION

Variation in video encoding time exists due to data dependent processes of the coding algorithm. For instance, the motion estimation (ME) may use early-jump-out technique to skip a search position depending on the partially accumulated sum-absolute-different (SAD)[1,2,3]. The skipping may also be based on the predicted SAD to compare with a threshold SAD value. Besides ME, Erol et al[2] and Tye et al[3] used zero block prediction prior to DCT to eliminate the computation of some macro-blocks (MB) using the pixels sum absolute value as predictor. Erol reported a speedup of 1.6 using such algorithmic optimization over a single processor implementation. Together with MMX instruction extension, a speedup of 2, i.e. a frame rate of 15 frames per second (fps), was reported in doing H.263+ encoding at QCIF resolution on a Pentium MMX 200MHz PC. Tye reported an implementation on a PCI-board consisting of a TMS320C80 in which 2 parallel processors were used for encoding and one for decoding. It achieved 7-10 fps for QCIF video. Also for speed enhancement, the ME as proposed by Akramullah et al[4] skips the search points that are outside the 3x3 or 5x5 search window if the minimum SAD found is smaller than a threshold. Using the visual instruction set, they achieved 12.17 fps on an UltraSPARC-1 workstation of 167MHz for H.263 encoding at QCIF size. All these speed enhancement methods have variable computation time towards different video content.

For better image quality and faster coding speed, Chung et al[5] proposed an adaptive non-linear three-step-search giving PSNR very close to that of full-search and search time faster than three-step-search. The algorithm stops in the first step if the position found is the center position or its 8 neighbors. Otherwise, further refinement process is carried out around the minimum SAD position. Furthermore, the refinement searching terminates sooner if the minimum position from the first step is closer to the search center. Similarly, Nitta et al[6] proposed a hardware architecture for scene adaptive three-step-search that allows hopping vector, prioritized zero vector and prioritized unchanged vector from the last encoded vector. The search area can be contracted if the scene is a still image or when there is little motion. For bit rate control, Tiwari and Viscito[7] used an iterative algorithm to compute the nominal quantizer for a frame based on a piecewise linear model between the quantizer reciprocal and the scene complexity. If the number of bits produced violates the virtual buffer verifier constraint, then the picture is re-encoded using a modified nominal quantizer. With these scene adaptive algorithms, it is expected that the encoding time across the MB's and the frames varies over a wide range.

This variation is the cause for performance degradation on parallel implementation. Owing to this, run-time scheduling methods have been proposed. For example, He et al[8] proposed a parallel MPEG-4 coding algorithm for variable size video

---

[*] Correspondence: Email: kkleung@eee.hku.hk; Tel: 852-2857-8414; Fax: 852-2559-8738

objects by balancing the number of MB's over the processors. They achieved 15-30 fps for CIF video using 20 SUN UltraSPARC connected by a ForeSystems ATM switch. However, the variation in MB computation time is not compensated and the communication overhead for search area is not included. The load-balancing algorithm proposed by Yung and Chu[9] considered variation in MB encoding time. The algorithm performs computation and communication in separate phases and tries to balance the computation time over the processors. Simulation results show that 6.45% improvement can be obtained, i.e. 1 fps improvement from 15.5 fps, using the MB encoding time estimated from the last frame. Merely balancing the computation may not lead to scalable implementation because there are inter-processor communication overhead and task precedence constraints that must also be considered.

Some methods are based on distributed schemes that offer flexible processor management. For example, Barbosa et al[10] proposed a multithread configuration with a number of threads running on a number of processor. Whenever a thread becomes free, it selects a task that is ready to execute from a global task pool. This method has achieved 43 fps MPEG-1 encoding for 352x240 video using purely-temporal parallelism on a SUN Enterprise 10000 system composed of 32 processors and 8GB of shared memory. Although there is no explicit inter-processor communication (*IPC*) instruction, the overhead due to synchronization and I/O limits the scalability and causes the speedup to drop from about 9.5 to 6.5 when the number of threads increases from 16 to 32. On an Ethernet-connected SUN-Classic workstation cluster, Nang and Kim[11] implemented a master-slave configuration in which the slave processors request from the master processor computation tasks to execute. It achieved a frame rate of 7.8 fps for MPEG-1 encoding at 320x240 resolution, with GOP level task decomposition. The percentage time for communication increases steadily from about 34% to 63% for number of slaves from 1 to 30. In other words, the efficiency or the percentage time performing useful computation is decreasing due to excessive *IPC* overhead. As a matter of fact, it can be shown that the shared network or memory has increasing access time with the number of processors due to access contention[12].

As observed, first, the communication overhead is an important issue that can constitute to substantial performance degradation if not scheduled efficiently. Second, the parallelization involves not only balancing the computation time, but also the scheduling of the tasks under the data dependency of the algorithm. Third, run-time scheduling is constrained by the short time available for making scheduling decision such as task selection heuristic for execution in a ready slave processor. The system utilization may be low using this simple heuristic. In fact, multiprocessor scheduling is an NP-hard problem[13,14]. Currently, compile-time iterative scheduling algorithms are able to provide only near-optimal solution under realistic model with consideration of *IPC* and resource limitations[15,16]. Moreover, the schedule generation time is long compared to the frame encoding time. Thus, it is inherently challenging to consider run-time optimization of parallel video coding.

Therefore, we are motivated to tackle this problem. Our approach is based on a run-time selection scheme with a number of compile-time schedules. This method has the advantage of being near-optimal using compile-time schedules and simultaneously, involving only run-time selection rather than re-scheduling such that the run-time overhead can be reduced. The data communication and computation for run-time selection are also included in the algorithm being scheduled. Implementation of a parallel H.261[17] encoder on the IBM SP2 multiprocessor system gives an average speedup of about 13.5 (frame rate of 38.5 fps) for a CIF sequence consisting of segments of 6 different scenes. This is equivalent to an average improvement of about 16.9% over the single schedule scheme with schedule adapted to each of the scenes. Using an open test sequence consisting of 8 video segments, the average improvement achieved is 13.2%, i.e. an average speedup of 13.3 (35.6 fps).

This paper is organized as follows: Section 2 presents the model for scheduling and derives the evaluation process of the schedule performance. Section 3 illustrates the effect of task execution time variation and introduces the run-time selection method for video coding. Section 4 outlines the experiments, test conditions and test results together with detailed discussions. Finally, this paper is concluded in Section 5.

2. MODEL FOR MULTIPROCESSOR SCHEDULING

2.1. Data flow graph

In order to achieve true parallel performance, a realistic model for scheduling is crucial[15]. We adopted the data flow graph (DFG) to represent the coding algorithm since it highlights the data dependency between the coding tasks. A parallel program is described by a DFG $G(V_T \cup V_D, E_{TD} \cup E_{DT})$ where V_T represents a set of non-preemptive computation tasks while V_D represents a set of data objects. E_{TD} (or E_{DT}) contains a set of directed edges connecting vertices from V_T to V_D (or V_D to V_T), which represents the production (or referencing) of data objects by the tasks. For example, a task can be the MB encoding function, which references data objects such as the input MB and reference MB's to produce data objects such as the encoded bit-stream and reconstructed MB. G may be an iterative DFG such as video encoding in which an iteration

corresponds to the encoding of a frame. The unfolding of G induces an acyclic DFG $\underline{G}(\underline{V_T} \cup \underline{V_D}, \underline{E_{TD}} \cup \underline{E_{DT}})$ where $\underline{V_T} \cup \underline{V_D}$ contains a sequence of isomorphic and disjoint instances of $V_T \cup V_D$ and $\underline{E_{TD}} \cup \underline{E_{DT}}$ is the set of directed edges between the vertices in $\underline{V_T} \cup \underline{V_D}$. For $T_x \in V_T$ and $D_y \in V_D$, $T_{x,i}$ and $D_{y,i}$ denote respectively the instances of T_x and D_y in the ith iteration ($i=1,2,\ldots$). The edge set $\underline{E_{TD}} \cup \underline{E_{DT}}$ is formed from $E_{TD} \cup E_{DT}$ based on the dependence distance d of the data objects. For T_x, $T_z \in V_T$ and $D_y \in V_D$, if $(T_x,D_y) \in E_{TD}$ and $(D_y,T_z) \in E_{DT}$, then $(T_{x,i},D_{y,i}) \in \underline{E_{TD}}$ and $(D_{y,i},T_{z,i'}) \in \underline{E_{DT}}$ where $i=i'\text{-}d(D_z)$. It means that T_z references the D_y in the $\{d(D_y)\}^{th}$ previous iteration. An example DFG is depicted in Fig. 1. A summary of all the symbol definitions can be found in the Appendix.
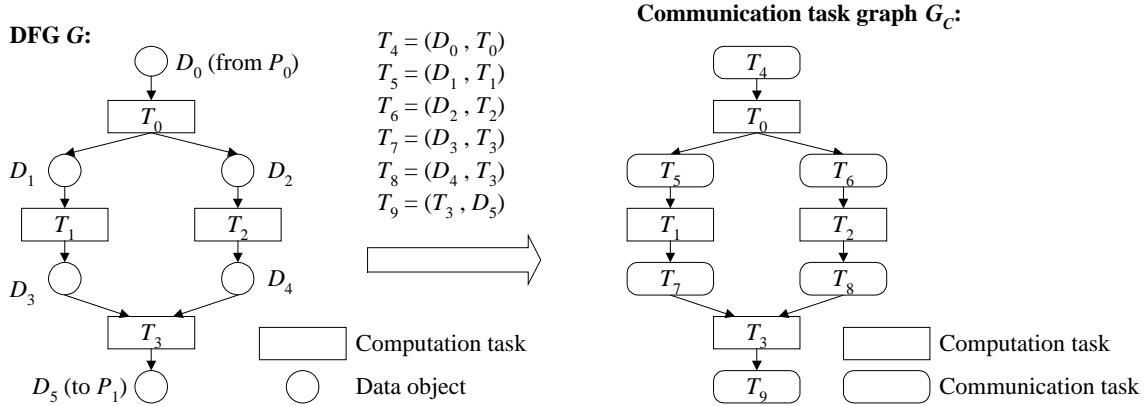

Figure 1. Data flow graph and transformation to communication task graph

2.2. Platform model

The platform model (S_R, S_P, S_C) consists of a set S_R of resources such as processors and communication links. S_P is a set of processors connected with a communication network of a given topology. Between each pair of processors, there is a static circuit-switched communication channel made up of a number of resources in S_R. Each data transfer is non-preemptive such that the channel resources are exclusively occupied throughout the duration of transfer. For different channels, if they share resources in common, the data transfers through them have to occupy non-overlapped time duration. For each computation task T, the execution time, $ET(T)$, is assumed to be known *a priori*. For a data transfer, the execution time may be modeled by the channel setup time, the product of the effective channel bandwidth and the data size.

2.3. Communication task graph

For scheduling of communication and computation, the communication task graph $G_C(V_T \cup V_{CT}, E_C)$ is introduced, in which V_{CT} is a set of communication tasks and E_C is the set of directed edges between the tasks, either computation or communication. The communication tasks are formed from the input DFG considering the edges incident on the data objects. For $D_y \in V_D$ and $T_x, T_z \in V_T$, if $(T_x,D_y) \in E_{TD}$ and $(D_y,T_z) \in E_{DT}$, then a unique communication task T_c is generated for the edge (D_y,T_z). The dependence distances between T_x, T_c and T_z are correspondingly $d(T_x,T_c)=d(D_y)$ and $d(T_c,T_z)=0$. It means that T_c transfers the instance of D_y produced by T_x in the $\{d(D_y)\}^{th}$ previous iteration. On the other hand, if D_y has no outlet edge, then a unique communication task is generated for the inlet edge of D_y. Fig. 1 lists the communication task of the example DFG. Unfolding of G_C induces an acyclic task graph $\underline{G_C}(\underline{V_T} \cup \underline{V_{CT}}, \underline{E_C})$ where $\underline{V_T} \cup \underline{V_{CT}}$ contains a sequence of instances of $V_T \cup V_{CT}$. and $\underline{E_C}$ is the set of directed edges between the vertices in $\underline{V_T} \cup \underline{V_{CT}}$. For $T_x,T_y \in V_T \cup V_{CT}$, we have $(T_x,T_y) \in E_C \Leftrightarrow (T_{x,i},T_{y,i'}) \in \underline{E_C}$ and $i=i'\text{-}d(T_x,T_y)$.

2.4. Schedule characterization

In the execution of an iterative program, each loop consists of one instance of each task. All the loops execute according to the same schedule, which can be characterized by the tuple (RI,Map,E_{DS},Seq). For $T \in V_T \cup V_{CT}$, $RI(T)$ is defined as the relative iteration index of T, i.e. the instance of T in the jth loop belongs to iteration $j+RI(T)$. $Map(T)$ is defined as the processor mapping of T for $T \in V_T$. E_{DS} is a set of directed edges that represents the data forwarding relation between communication tasks. For $T_x,T_y \in V_{CT}$, if $(T_x,T_y) \in E_{DS}$, then T_y uses, as its source, the data object transferred and buffered in the destination processor of T_x. As such, the source processor of T_y becomes the destination processor of T_x, and T_x becomes a predecessor of T_y. Seq is defined as the order of the tasks to be scheduled. It is a topological ordered sequence that satisfies the precedence relation among the tasks. The performance of the schedule is evaluated with the help of several intermediate task graphs. First, the precedence relations between the tasks are determined with respect to their relative iteration indices.

Then, an augmented precedence graph G_P' is derived from the precedence relations of the precedence graph G_P and the platform resource constraints.

2.4.1. Precedence graph G_P

The precedence relation of the tasks in the schedule is represented by the precedence graph $G_P(V_T \cup V_{CT}, E_P)$, which is derived from G_C and RI. In the jth loop, the tasks T_x and T_y belong to iterations $i=j+RI(T_x)$ and $i'=j+RI(T_y)$ respectively. $T_{x,i}$ is a predecessor of $T_{y,i'} \Leftrightarrow (T_{x,i}, T_{y,i'}) \in \underline{E_C} \Leftrightarrow (T_x, T_y) \in E_C$ and $i=i'-d(T_x,T_y) \Leftrightarrow (T_x,T_y) \in E_C$ and $RI(T_x)=RI(T_y)-d(T_x,T_y)$. Therefore, $(T_x,T_y) \in E_P \Leftrightarrow (T_x,T_y) \in E_C$ and $RI(T_x)=RI(T_y)-d(T_x,T_y)$. Note that E_P contains a subset of the edges of E_C. Thus, the precedence constraints can be alleviated by properly overlapping successive iterations in the schedule. Fig. 2 depicts the precedence graphs with T_0, T_4, T_5, T_6 having a RI of 0 while $T_1, T_2, T_3, T_7, T_8, T_9$ having a RI of -1.
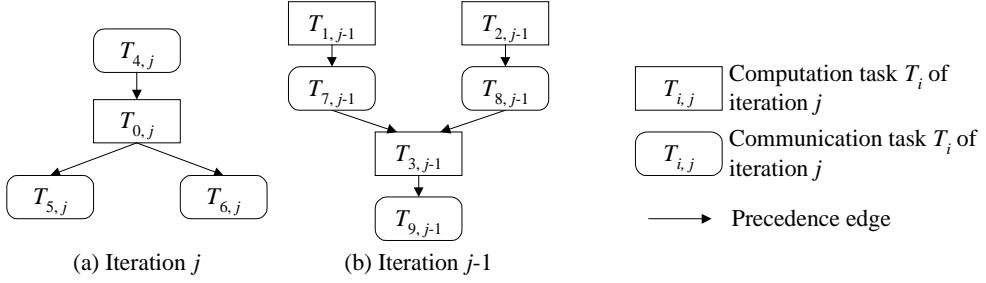
Figure 2. Precedence graphs

2.4.2. Execution time-line generation

Based on a given schedule, the execution time line is generated by traversing and scheduling the tasks in the order of *Seq* one after another. A communication task is scheduled to the resources of the channel between the source and the destination processors. Due to the serialization of the tasks to be executed in each resource, and the data forwarding between communication tasks, there exist precedence relations in addition to that of G_P, which can be represented by an augmented precedence graph G_P'. That is, $(T_x,T_y) \in E_P'$ if $(T_x,T_y) \in E_P$, or $(T_x,T_y) \in E_{DS}$, or T_x is just before T_y in some resource. The overall make-span of the schedule, or the schedule length, is equal to the critical path length of G_P' given by

$$ScheduleLength = \max\{tlevel(T) + ET(T) | T \in V_T \cup V_{CT}\}, \tag{1}$$

where the function $tlevel(T)$ is the start time of T. To determine $tlevel$, the tasks can be traversed in the order of *Seq* by using

$$tlevel(T) = \max\{0, tlevel(T_p) + ET(T_p) | (T_p, T) \in E_P'\}. \tag{2}$$

3. RUN-TIME SCHEDULE SELECTION FOR VIDEO CODING

3.1. Effect of task execution time variation

To illustrate the effect of task execution time variation on the schedule length, Table 1 shows an example schedule (Sch. 1) with the timeline depicted in Fig. 3(a). The schedule length is 5 time units. If the execution times of T_1 and T_2 are exchanged, the schedule length is increased to 6 time units as shown in Fig. 3(b). The increase in $ET(T_1)$ causes delay to the start time of T_7 and its successor tasks such as T_5, T_6, T_7, T_8. It also leads to high idling time of the processors. It should be noted that the precedence from T_7 to T_5 and T_8 is not due to data dependency as there is no precedence edge in the G_P in Fig. 2. Instead, this precedence is due to resource constraint. By re-scheduling the task sequencing, another schedule (Sch. 2) as depicted in Table 2 has the schedule length reduced as shown in Fig. 3(c). In this schedule, T_8 is scheduled before T_7, thus filling the idling time of P_1 and P_3. T_6 is also scheduled before T_5 that fills the idling time of P_0 and P_3. On the other hand, if T_1 and T_2 have not exchanged the execution time, then Sch. 2 is less optimized than Sch. 1 as depicted in Fig. 3(d). This shows that the variation in task execution time can cause significant performance reduction and that the task sequencing can be modified to restore the performance. In fact, the optimization of a schedule is based on a set of tasks and their execution time. The schedule so generated is specifically optimized to the given execution time.

3.2. Video coding DFG with run-time selection

Fig. 4 depicts the run-time schedule selection scheme. The *NSP* algorithm[15] is first used to generate a number of compile-time schedules each based on the execution time of a training image. During run-time, the task execution time as measured

is used to evaluate the schedules to select the one giving the shortest frame time to encode the next frame. Fig. 5 shows the DFG for video encoding. We assume that the input frames originate from an I/O device attached to processor P_0. The output in the form of bit-stream is delivered in processor P_1. Both P_0 and P_1 are dedicated for these video I/O tasks and the associated data distribution and collection tasks. The encoding of each MB includes functions from motion estimation to variable length encoding to form the encoded MB, and the inverse counterparts of these functions to generate the decoded MB. The task *HDVLC* performs the VLC of all the headers and concatenation of encoded MB's. This task is executed sequentially due to the spatial data dependency among the MB headers. As shown in the figure, the encoding of each MB needs to reference the decoded MB's of the last frame around the position of the current MB. Thus, the dependence distance of all the decoded MB's is one.

(a) A pipelined and optimized schedule (Sch. 1) with overlapping between successive loops

(b) When $ET(T_1)$ and $ET(T_2)$ are exchanged, i.e. $ET(T_1)=3$ & $ET(T_2)=2$, the length of Sch. 1 is lengthened.

(c) With $ET(T_1)=3$ and $ET(T_2)=2$, a second schedule (Sch. 2) that is more optimized than Sch. 1 .

(d) With $ET(T_1)=2$ and $ET(T_2)=3$, Sch. 2 is less optimized than Sch. 1.
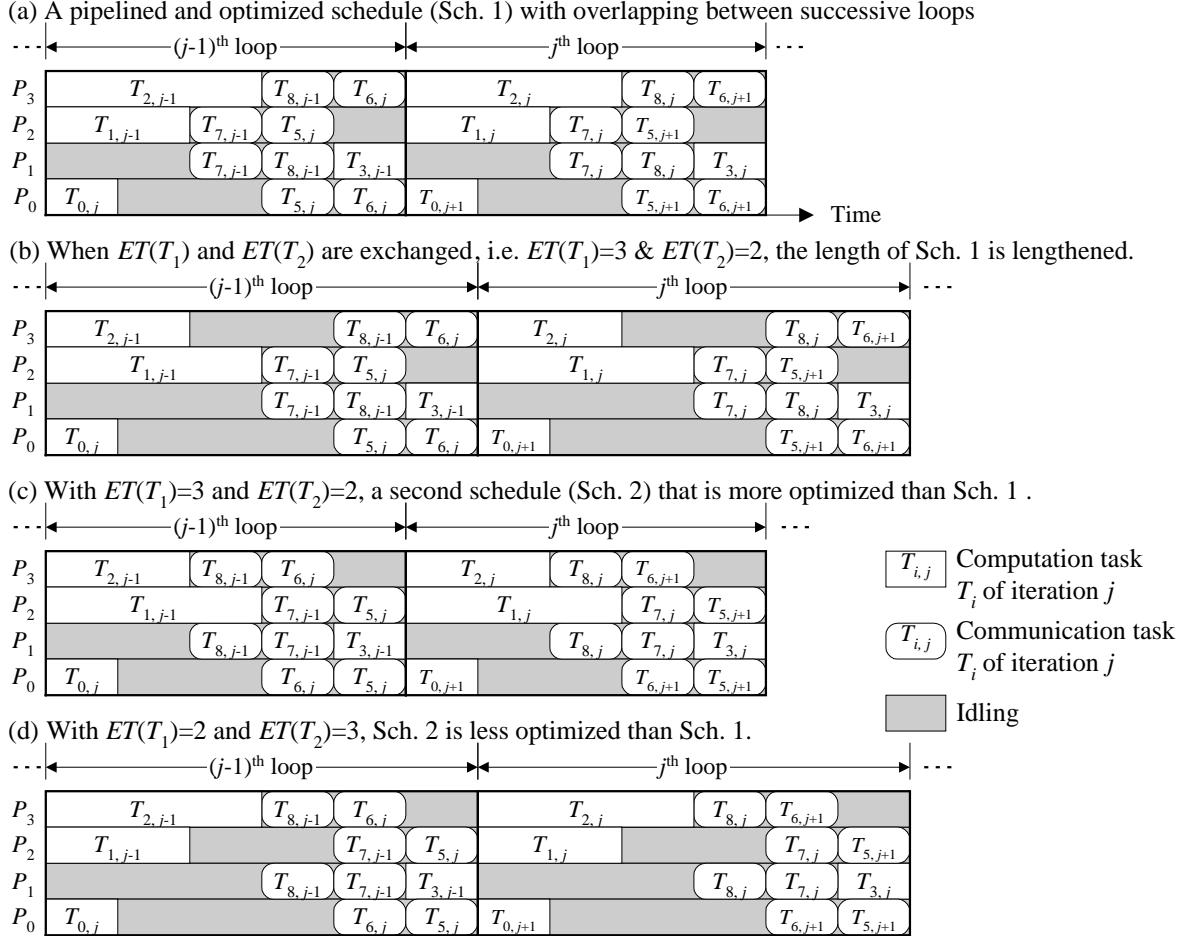
Figure 3. Effect of task execution time variation on schedule length

Task T	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Map	P_0	P_2	P_3	P_1	-	-	-	-	-	-
RI	0	-1	-1	-1	0	0	0	-1	-1	-1
Seq	T_4	T_0	T_2	T_1	T_7	T_5	T_8	T_6	T_3	T_9

Table 1. Schedule 1

| Task T	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Map	P_0	P_2	P_3	P_1	-	-	-	-	-	-
RI	0	-1	-1	-1	0	0	0	-1	-1	-1
Seq	T_4	T_0	T_1	T_2	T_8	T_6	T_7	T_5	T_3	T_9

Table 2. Schedule 2
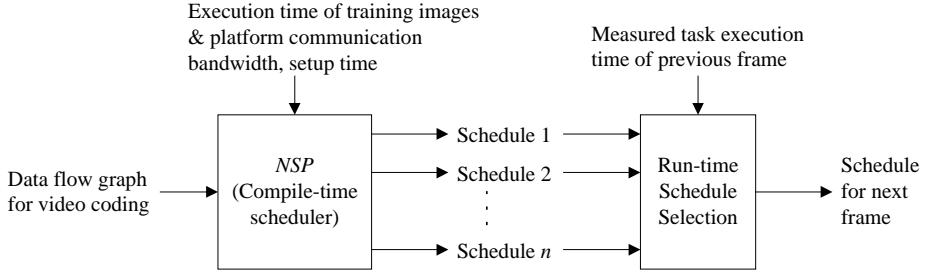
**Figure 4.** Run-time schedule selection scheme
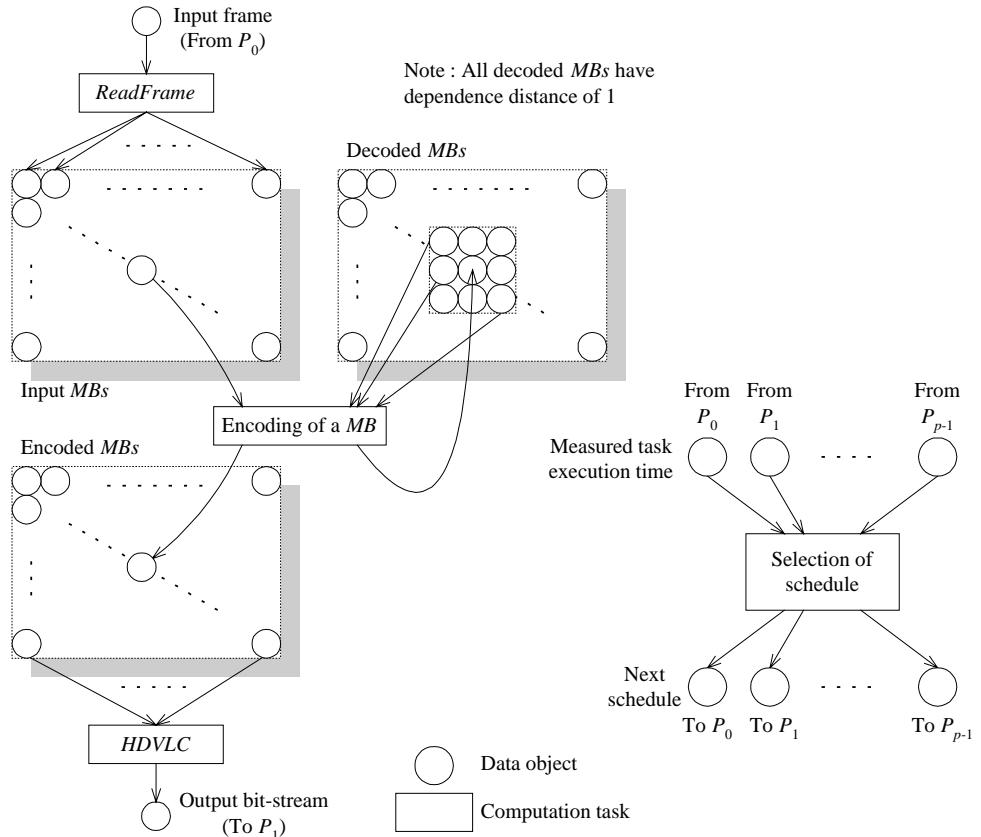
Figure 5. DFG for video coding

Apart from the video coding tasks, there is one more task for schedule selection. Processor P_2 is dedicated to execute this task. It evaluates the compile-time schedules using (1) and (2) by substituting the task execution time measured and collected from the other processors. The schedule giving the shortest frame time is sent to the processors. In practice, a copy of the schedules is resident in each processor so that it requires only the schedule identifier to be transferred.

3.3. Schedule generation

There are two limitations to the schedules for run-time selection. First is the constraint on the task processor mapping. If the schedules have different *Map*, then it requires communication overhead for data relocation upon schedule switching. Since the schedules are generated based on the assumption of static scheduling, any inter-loop data referencing assumes static processor mapping of the computation tasks and the data objects produced from them. With different *Map*, some of the communication tasks have different source processor in different schedules. So, there is explicit communication for relocation of the data objects involved.

The second limitation is on the relative iteration indices. All the schedules must have identical setting of *RI* for run-time schedule selection although it is possible to have different *RI* for different tasks. In other words, no matter which

schedule is selected to execute the next loop, the instance of each task is determined as a continuation of the instance from the current loop. Otherwise, if the schedules have different *RI* setting, then some of the tasks may have missing or duplicated instances.

In our approach, we adopt a constant processor mapping scheme for the tasks so as to eliminate the data relocation upon schedule transition. Essentially, the MB's are ordered from top to bottom and left to right in each row. Then groups of 4 consecutive MB's each are allocated to the processors in a round robin fashion with the aim to balance the computation time over the processors. The setting of *RI* is fixed in a way such that when the MB's of the current frame are being encoded, the next frame is read and distributed from P_0. In parallel, the encoded MB's of the last frame are collected to P_1 for bit-stream output. As such, there is a latency of 3 frames encoding time from frame reading to the output of bit-stream. This pipelined setting can eliminate the precedence constraints from the input MB communication tasks to the MB encoding tasks. Similarly, the precedence constraints from the MB encoding tasks to the encoded MB communication tasks are also eliminated. This helps to break down the computation critical path of the DFG.

With fixed *Map* and *RI*, the schedules differ only in the *Seq* and E_{DS} components. The *NSP* algorithm is executed to improve a given schedule by successive modification to these components under this scheme. For each video scene, a compile-time schedule is generated. It requires the MB computation time of a representative frame of the scene. To find the representative frame, consider a particular MB position, the computation time is obtained by taking the average MB encoding time measured over a training sequence of frames at that particular position.

4. EXPERIMENTS AND DISCUSSIONS

4.1. Evaluation conditions

To illustrate the performance of the scheme, implementation was done on the IBM SP2[18]. A parallel encoder was built from a H.261 encoder from Hung[19]. In this implementation, the computation time of a MB depends largely on the ME search time, which has considerable variation towards different video content due to the use of early-jump-out technique. For benchmarking, all the task execution time and frame time were measured in units of microsecond by calling *gettimeofday*. The evaluation used a video sequence consisting of 6 scenes each consisting of 50 frames in CIF resolution, i.e. a total of 300 frames. In the Multiple Schedules (*MS*) scheme, there are six schedules, each generated based on the representative frame encoding time of one of the scenes.

For each scene, a compile-time schedule was generated by executing *NSP* for 10 times using 10 randomly generated schedules as seeds. Then the best schedule was adopted. Various numbers of processors were included in the test. As a total, there were 360 random schedules used to generate 360 near-optimal schedules, from which 36 were picked corresponding to 6 scenes and 6 different number of processors, i.e. 4, 8, 12, 16, 20, 24.

The schedules were used to encode the 300 frames of training sequence as well as an open test sequence composed of 8 segments each consisting of 30 frames. Besides testing for the *MS* scheme, each of the 6 schedules was also tested individually in a single schedule scheme. During the test, all the processors are synchronized before each frame and the frame time is taken to be the finish time of the latest task relative to the synchronization point. To eliminate the effect of the Unix OS intermittent intervention, the test has 8 runs and the median frame time over the 8 runs is taken for each frame.

4.2. Results and discussions

The speedup curves of the 6 schedules for the training sequence with increasing number of processors are depicted in Fig. 6. For number of processors less than 12, the curves are closer to each other. For higher number of processors, the curves become more diverges. For example, at *p*=24, the schedule from "Miss A" has the widest spread of speedup from about 9.29 (37.1 fps) to 14.63 (29.8 fps). The schedule from "Weather" has the narrowest range of about 9.97 (37.1 fps) to 12.47 (25.4 fps), which is also a substantial variation of speedup. The schedule length is the critical path length in the augmented precedence graph. With more processors, the critical path is shorter and each path contains smaller number of MB's. Therefore, the lengths of the paths are more sensitive to variation in the MB encoding time. The schedules then have more diverged frame time for different schedules. Note that the speedup is relative to the sequential encoding time.

From Fig. 6(a), the schedule from "Akiyo" shows decreased or nearly unchanged speedup from *p*=20 to 24 for the scenes "Bream", "Miss A", "News" and "Salesman". This schedule shows high processor idling time towards frames of these scenes at *p*=24. In fact, the "Akiyo" scene has a very different spatial distribution of encoding time from that of these scenes. For example, the spatial encoding time distribution for "Akiyo" and "Bream" are depicted in Fig. 7. The background MB's of "Akiyo" take less than 0.6ms to encode while that of "Bream" take more than 1.2ms. It is because the "Akiyo"

scene has an artificial black background such that the pixels have identical luminance value. In performing ME, the SAD is then exactly equal to zero at the search center such that the ME algorithm can skip all the other search points. For "Bream", the background has a uniform dark intensity with slight variation but the luminance is not exactly identical over the pixels. In performing ME, all the search positions have non-zero SAD with very small variations. The algorithm has to completely evaluate the SAD of each search point before it can determine whether to skip or accept. This explains why the background takes higher encoding time than the foreground. Since the schedule from "Akiyo" has a critical path containing many MB's in its background locations. When substituted with the encoding time of "Bream", the critical path becomes much longer. Similar to "Bream", the scenes "Miss A" and "Salesman" also have a background with high encoding time. The "News" scene differs from "Akiyo" in the foreground in that "News" shows two persons side-by-side rather than one in the middle.
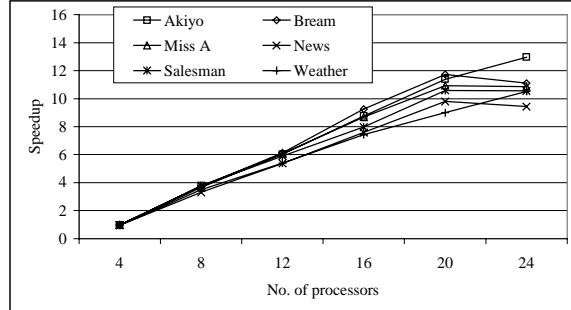
Figure 6(a). Speedup of Sch. from "Akiyo" for the 6 scenes
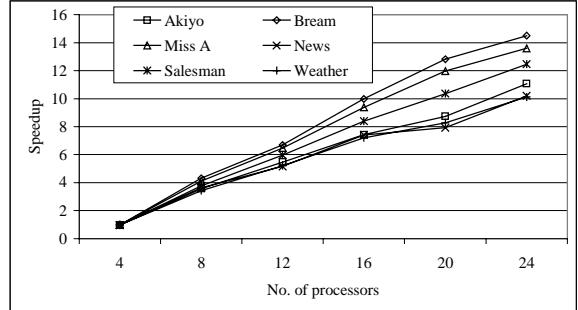
Figure 6(b). Speedup of Sch. from "Bream" for the 6 scenes
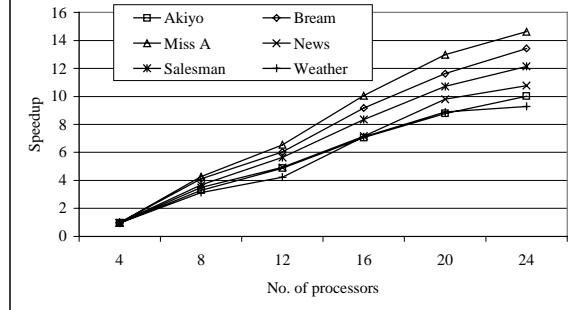
Figure 6(c). Speedup of Sch. from "Miss A" for the 6 scenes
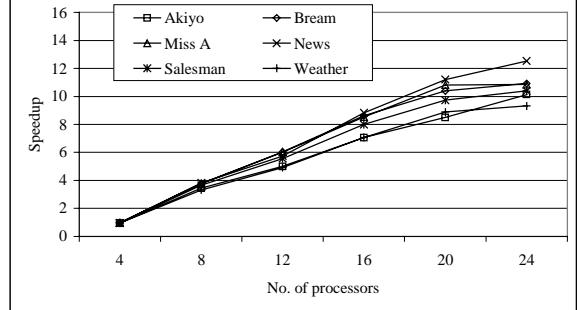
**Figure 6(d).** Speedup of Sch. from "News" for the 6 scenes
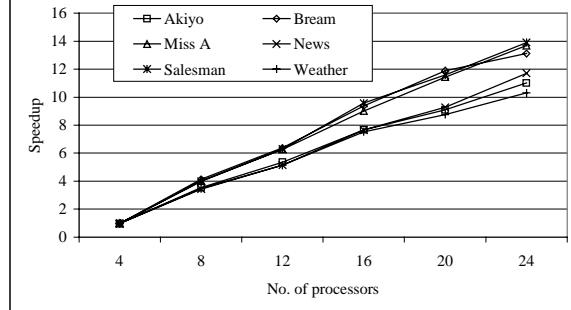
Figure 6(e). Speedup of Sch. from "Salesman" for the 6 scenes
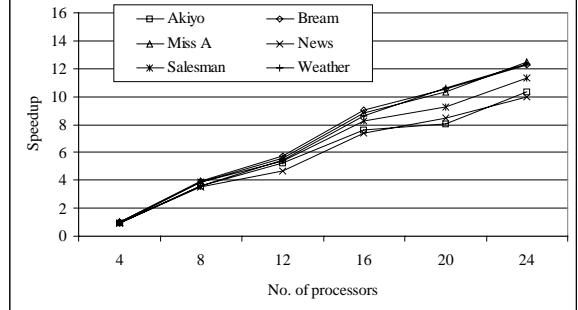

Figure 6(f). Speedup of Sch. from "Weather" for the 6 scenes

At p=24, each schedule shows the best speedup for its training video segment. For example, the schedule from "Akiyo" has a speedup of about 12.96 (49.5 fps) for "Akiyo" segment while the other 5 schedules are in the range from 10.02 (38.4 fps) to 11.07 (42.4 fps). Similarly for the "Bream" segment, the schedule from "Bream" gives speedup of about 14.50 (29.1 fps) while the others in the range 10.92 (22.0 fps) to 13.43 (27.0 fps). Also observed is that some video segments have higher average speedup due to their higher sequential encoding time. For example, "Bream" has a higher average speedup than "Akiyo" at p=24. Obviously from Fig. 7(a) and (b), "Bream" has a higher sequential encoding time than "Akiyo".

The utilization of the processors is given by the system efficiency, which is defined as the ratio of speedup to p. Fig. 8(a) & (b) depict respectively the speedup and efficiency of *MS*. It is found that *MS* gives a more bunched-up speedup up to

24 processors. At p=24, the speedup range is 12.30 (48.5 fps) to 14.75 (29.8 fps) and the corresponding efficiency range is 51.25% to 61.47%. The speedup comparison at p=24 is depicted in Fig. 9(a) & (b). From the data, *MS* is at most 0.5% lower than the best for all the video segments. In the 8 repeated tests, *MS* can select the corresponding schedule that was trained for the frame under encoding for about 96.2% of the time. About 2% of the mismatched selections occur in the frames after scene changes as the selection is based on the measured execution time of the second previous frame. Therefore, more mismatched selections are expected with more frequent scene changes. The remaining 1.8% mismatched selection occurs in some isolated frames with outlying long execution time, which leads to inaccurate estimation. This is most likely caused by the Unix OS overhead.
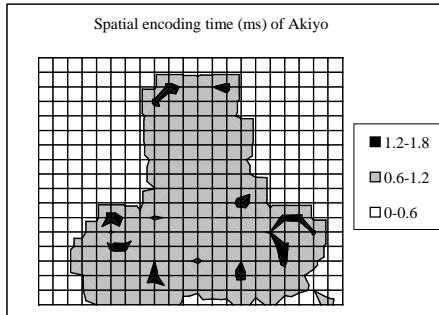
Figure 7(a). Spatial encoding time of Akiyo scene
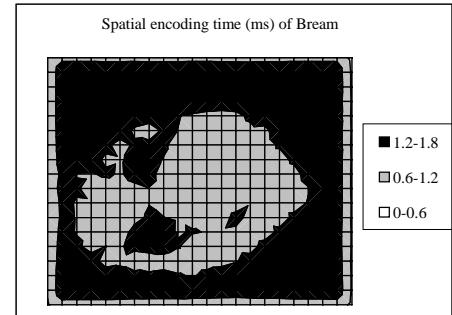
**Figure 7(b).** Spatial encoding time of Bream scene
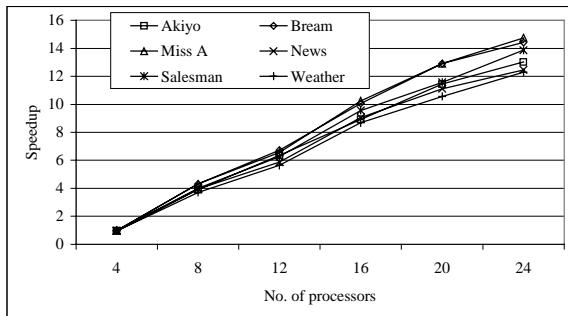
**Figure 8(a).** Speedup of *MS* for the 6 scenes
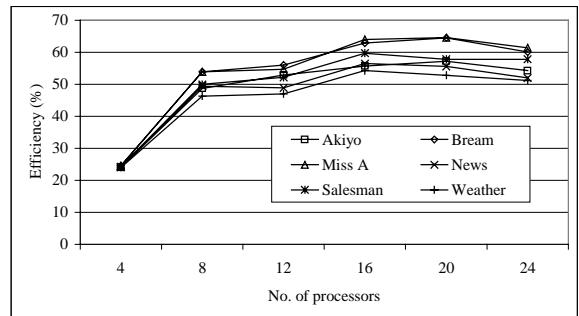
**Figure 8(b).** Efficiency of *MS* for the 6 scenes
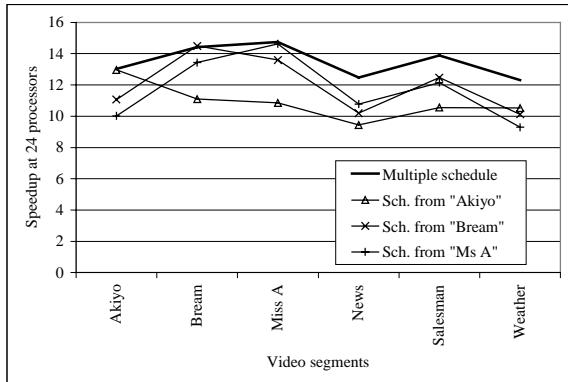

Figure 9(a). Speedup comparison at *p*=24
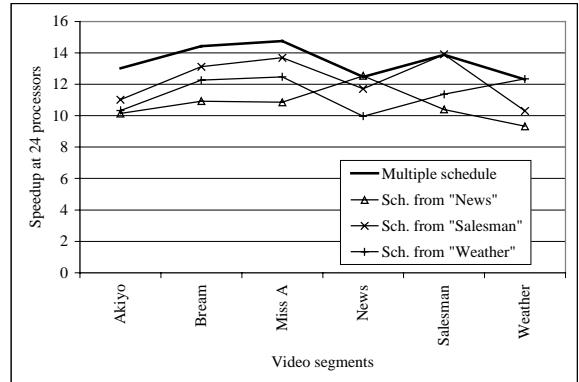
Figure 9(b). Speedup comparison at *p*=24

The average speedup of *MS* over the whole training sequence is depicted in Fig. 10(a). A boundary curve is added for the schedules under the current MB allocation. This curve corresponds to the ideal case without communication overhead and task precedence constraint but under the current MB allocation scheme. It is found that there is slight bending downward at *p*=12. It is due to uneven distribution of computation load over the processors causing processor idling. In the implementation, groups of 4 neighboring MB's are merged to form coarser grain tasks in order to reduce the time for scheduling and run-time schedule evaluation as well as to enhance the locality of decoded MB access between neighboring MB's. The linear speedup of perfect load balance is higher than the measured by more than 3 because the scheme used 3 master processors dedicated for centralized communication. These masters have little computation to perform. For instance,

the time for schedule evaluation and *HDVLC* are approximately 5.59ms and 7.5ms respectively, which is well below the 33ms real-time requirement. This scheme is justified because, if there is only one master processor, then the master could become a bottleneck when large number of processors are used. As depicted in Fig. 10(b), the efficiency of the implementation is only about 24% at p=4, which shows that the master processors are inefficiently utilized. It is because they were not scheduled with any MB encoding tasks although there is no restriction of doing so. After p=4, the efficiency jumps to over 50% and increases to about 58.9% at p=16 as there are more processors for parallel encoding. Then the efficiency tends to level and drops slightly to about 56.1%. One reason for the drop is the data partitioning that becomes coarser with increasing number of processors, which is evidently shown in the drop of the ideal efficiency curve at p=24.
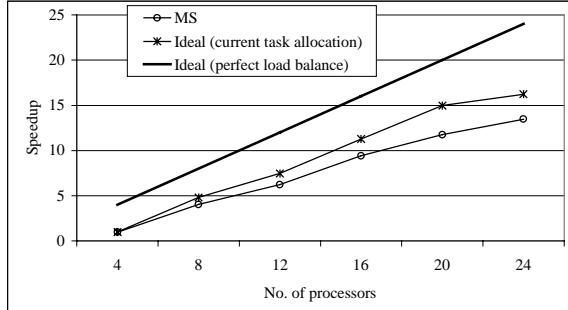

**Figure 10(a).** Average speedup of *MS* over whole sequence
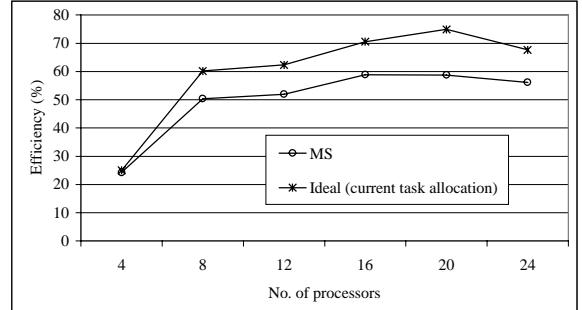
Figure 10(b). Average efficiency of *MS* over whole sequence

For the open test sequence, the speedup comparison at p=4 is depicted in Fig. 11(a) & (b). As observed, *MS* gives the best speedup for the "Akiyo", "Miss A" and "Weather" segments and second best for the other 5 segments with maximum deviation of about 0.16 (1.13%) from the best. The reason that *MS* cannot select the best schedule is the error in frame time estimation. In some cases, this error is larger than the frame time difference between the best schedule and the others. For example, the schedule from "Miss A" is the best for the "Claire" segment. However, *MS* selects the schedule from "Bream" for 29 frames due to under-estimation of the frame time by about 0.74ms (i.e. 2.3%) for the "Bream" schedule and over-estimation by 1.33ms (i.e. 4.3%) for the "Miss A" schedule. Also, the difference in frame time between these two schedules is only 0.69ms. Therefore, the estimation error is larger than the difference. One reason for the under and over-estimation is due to over-simplified modeling of the inter-processor communication network as a complete network. There exists buffering of some of the messages such that the sender can complete earlier than expected. On the other hand, some messages have longer transmission due to contention within the network.
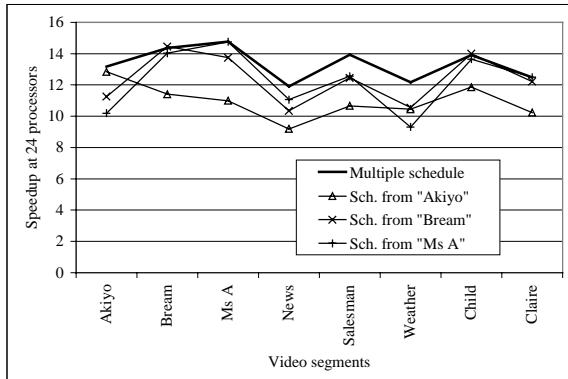

**Figure 11(a).** Open test speedup comparison at p=24
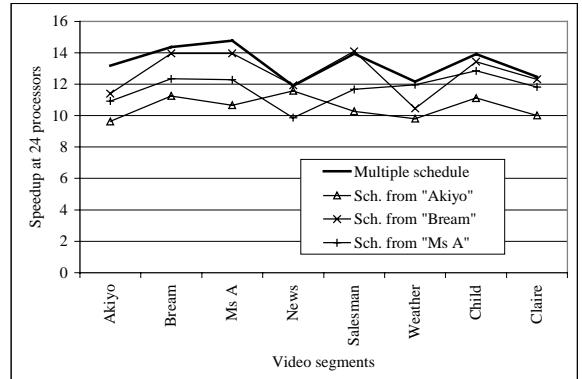
Figure 11(b). Open test speedup comparison at p=24

Fig. 12(a) & (b) depicts the frame rate results of *MS* for both the close and open tests. Comparing Fig. 8(a) and Fig. 12(a), roughly speaking, scenes such as "Bream" and "Miss A" have a comparably low frame rate but a high speedup at p=24. Whilst scenes such as "Akiyo", "News" and "Weather" have a high frame rate but low speedup. It should be noted that the parallel frame rate is an absolute performance metric, which is reference to the sequential encoding time. In this case, "Bream" and "Miss A" have similar sequential encoding time that is nearly two times that of "Akiyo", "News" and "Weather". Also observed is that *MS* has resulted in reduced frame rate variation relative to the average frame rate. For instance, from the close (or open) test results, the ratio of the frame rate span to the average frame rate over the 6 scenes is

decreased from about 0.62 to 0.51 (0.82 to 0.67) for *p* from 4 to 24. For each scene, *MS* can effectively select the best or second best schedule and therefore the speedup and frame rate are more converged towards the high end.
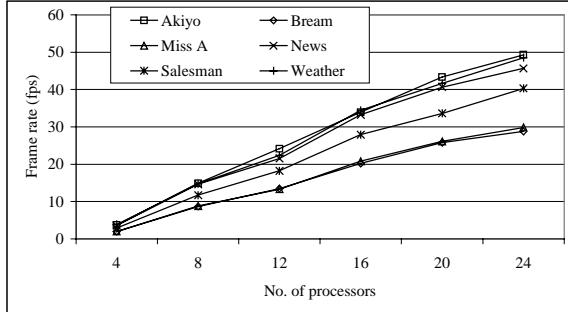


Figure 12(a). Close test frame rate of *MS*
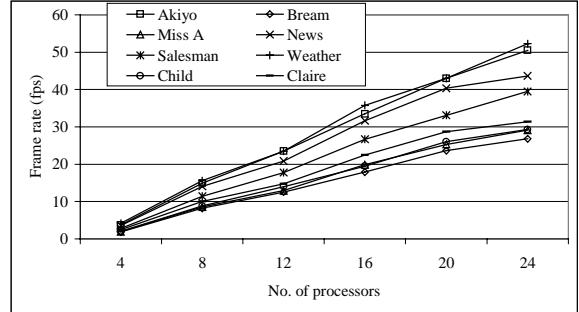
Figure 12(b). Open test frame rate of *MS*

5. CONCLUSIONS

From the results, we find that first, consideration of computation and communication scheduling in video coding algorithm parallelization is effective in minimizing the frame time. Second, further improvement can be obtained using multiple schedules with adaptive runtime selection. Third, for runtime schedule selection, the additional computation and communication overhead can be considered to be part of the algorithm for parallelization. In fact, the number of schedules may be increased to include wider spectrum of video scene computation characteristics at the cost of higher evaluation overhead, which can be tackled with multiple master processors[20]. Fourth, since the scheduling algorithm is generally applicable to arbitrary DFG, this method is equally applicable to other coding standards and possibly other video processing algorithms.

APPENDIX: DEFINITION OF NOTATIONS

DFG $G(V_T \cup V_D, E_{TD} \cup E_{DT})$:	
V_T, V_D	The sets of computation tasks and data objects.
E_{TD}, E_{DT}	The sets of directed edges from V_T to V_D and vice versa.
$d(D)$	Dependency distance of data object D.
$\underline{V_T}$, $\underline{V_D}$	The set of computation tasks and data objects in the unfolded version of G, i.e. $\underline{G}$.
$\underline{E_{TD}}$, $\underline{E_{DT}}$	The set of directed edges from $\underline{V_T}$ to $\underline{V_D}$ and vice versa in $\underline{G}$.
Communication task graph $G_C(V_T \cup V_{CT}, E_C)$:	
V_{CT}	The set of communication tasks.
E_C	The set of directed edges in G_C.
$d(T_x, T_y)$	The dependence distance from T_x to T_y, (T_y is dependent on T_x)
$\underline{V_T} \cup \underline{V_D}$	The set of tasks in the unfolded communication task graph $\underline{G_C}$.
$\underline{E_C}$	The set of directed edges in $\underline{G_C}$.
Platform model :	
S_R, S_P	The sets of resources and processors, $S_P \subset S_R$.
S_C	The set of channel resources between pairs of processors.
P	The number of processors.
Solution characterization :	
$RI(T)$	The relative iteration index of task T.
$Map(T)$	The processor to which computation task T is mapped onto.
$Seq(i)$	The ith task in the topological ordered sequence that satisfies precedence of G_P.
E_{DS}	The set of directed edges that represent the data forwarding relation between communication tasks.

Precedence graph $G_P(V_T \cup V_{CT}, E_P)$:	
E_P	The set of directed edges in G_P.
Augmented precedence graph G_P' $(V_T \cup V_{CT}, E_P')$:	
E_P'	The set of directed edges in G_P'.
$ET(T)$	The execution time of T.
$tlevel(T)$	The start time of T (also the longest path length in G_P' from an entry task to T).

ACKNOWLEDGMENT

REFERENCES

1. D.-Y. Hsiau, J.-L. Wu, "Real-Time PC-Based Software Implementation of H.261 Video Code", IEEE Trans. on Consumer Electronics **43**, No. 4, pp. 1234-1244, Nov. 1997.
2. B. Erol, F. Kossentini, H. Alnuweiri, "Implementation of a Fast H.263+ Encoder/Decoder", *Conf. Record of the 32nd Asilomar Conf. on Signals, Systems and Computers*, Vol. 1, pp. 462-466, 1998.
3. B. Tye, K. Goh, W. Lin, G. Powell, T. Ohya, S. Adachi, "DSP Implementation of Very Low Bit Rate Videoconferencing System", *Proc of the Int'l Conf. on Information, Communications and Signal Processing, ICICS*, pp. 1275-1278, Sept. 1997.
4. S. M. Akramullah, I. Ahmad, M. L. Liou, "Optimization of Software-based Real-time H.263 Video Encoding", *Proc. of the SPIE*, Vol. 3653, pp. 727-735, Jan. 1999.
5. H. Y. Chung, P. Y. S. Cheung, N. H. C. Yung, "Adaptive search center non-linear three step search", *Proc. of Int'l Conf. On Image Processing*, Vol. 2, pp. 191-194, 1998.
6. K. Nitta, T. Minami, T. Kondo, T. Ogura, "Motion estimation/motion compensation hardware architecture for a scene-adaptive algorithm on a single-chip MPEG2 MP@ML video encoder", *Proc. of the SPIE*, Vol. 3653, pp. 874-882,1999.
7. P. Tiwari, E. Viscito, "A parallel MPEG-2 video encoder with look-ahead rate control", *Proc. of the IEEE Int'l Conference on Acoustics, Speech and Signal Processing*, Vol. 4, pp. 1994-1997, 1996.
8. Y. He, I. Ahmad, M. L. Liou, "A Software-Based MPEG-4 Video Encoder Using Parallel Processing", IEEE Trans. on Circuits & Systems for Video Technology **8**, No. 7, pp. 909-920, Nov. 1998.
9. N. H. C. Yung, K. C. Chu, "Fast and Parallel Video Encoding by Workload Balancing", *Proceeding of the IEEE SMC'98*, pp. 4642-4647, Oct. 1998.
10. D. M. Barbosa, J. P. Kitajima, W. Weira Jr., "Parallelizing MPEG video encoding using multiprocessors", *Proc. of the Brazilian Symp. on Computer Graphics and Image Processing*, pp. 215-222, 1999.
11. J. Nang, J. Kim, "An Effective Parallelizing Scheme of MPEG-1 Video Encoding on Ethernet-Connected Workstations", *Proc. of Advances in Parallel and Distributed Computing*, pp. 4-11, 1997.
12. K. K. Leung, N. H. C. Yung, "Generalized parallelization methodology for video coding", *Proc. of the SPIE*, Vol. 3653, pp. 736-747, 1999.
13. M. R. Garey, D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., 1979.
14. D. Fernandez-Baca, "Allocating modules to processors in a distributed system", IEEE Trans. on Software Engineering **15**, No. 11, pp. 1427-1436, Nov. 1989.
15. K. K. Leung, N. H. C. Yung, P. Y. S. Cheung, "Novel Neighborhood Search for Multiprocessor Scheduling with Pipelining", *Proc. of the 4th HPC-Asia*, Vol. 1, pp. 296-301, May 2000.
16. L. Wang, H. J. Siegel, V. P. Roychowdhury, A. A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach", Journal of Parallel & Distributed Computing **47**, pp. 8-22, 1997.
17. "ITU-T recommendation H.261: video codec for audiovisual services at px64 kbits", International Telecommunication Union, 1990.
18. T. Agerwala, et al, "SP2 system architecture", IBM Systems Journal **34**, No. 2, pp. 152-184, 1995.
19. A. C. Hung, "PVRG-P64 Codec 1.1", Portable Video Research Group (PVRG), Stanford University, 1993.
20. N. H. C. Yung, K. K. Leung, "Parallelization of the H.261 video coding algorithm on the IBM SP2 multiprocessor system", *Proc. of the 3rd IEEE Int'l Conf. Algorithms & Architectures for Parallel Processing*, pp. 571-578, Dec. 1997.