Spatial and Temporal Data Parallelization of the H.261 Video Coding Algorithm

Nelson H. C. Yung, *Senior Member, IEEE*, and Kwong-Keung Leung, *Student Member, IEEE*

***Abstract*—In this paper, the parallelization of the H.261 video coding algorithm on the IBM SP2® multiprocessor system is described. The effect of parallelizing computations and communications in the spatial, temporal, and both spatial-temporal domains are considered through the study of frame rate, speedup, and implementation efficiency, which are modeled and measured with respect to the number of nodes (n) and parallel methods used. Four parallel algorithms were developed, of which the first two exploited the spatial parallelism in each frame, and the last two exploited both the temporal and spatial parallelism over a sequence of frames. The two spatial algorithms differ in that one utilizes a single communication master, while the other attempts to distribute communications across three masters. On the other hand, the spatial-temporal algorithms use a pipeline structure for exploiting the temporal parallelism together with either a single master or multiple masters. The best median speedup (frame rate) achieved was close to 15 [15 frames per second (fps)] for 352×240 video on 24 nodes, and 13 (37 fps) for QCIF video, by the spatial algorithm with distributed communications. For $n < 10$, the single-master spatial algorithm performs better with efficiency up to 90%, while the multiple-master spatial algorithm is superior for $n > 10$, with efficiency up to 70%. The spatial-temporal algorithms achieved average speedup performance, but are most scalable for large n.**

***Index Terms*—Domain decomposition, spatial parallelism, speedup, temporal parallelism, video coding.**

I. Introduction

ACTIVITIES in video coding can be traced back to over 30 years ago when analog techniques were primarily used for reducing video transmission bandwidth. In today's terms, the ways in which the video sequence is coded still determine much of the cost for storage and transmission. Among the variety of coding techniques and methods developed so far, international standards such as the H.261 [1], H.263 [2], MPEG-1 [3], MPEG-2 [4], and MPEG-4 [5] have given a clear direction to encoder/decoder implementation. They in turn fueled the rapid expansion of applications into multimedia computing, information storage, video phone/conferencing, remote sensing, medical imaging and many other audiovisual services [6]–[8].

As the application coverage of these standards increases, it becomes apparent that real-time coding (24–30 fps) is essential. However, due to the complex nature of coding itself, the real-time performance for a standard frame size has not been reached using our current single processor technology. Although many researchers have opted to consider new and faster coding techniques [9]–[11], there have been several attempts at parallel implementation of the current coding standards. For instance, Sijstermans and Van der Meer implemented an MPEG-1 encoder for CD-I production using a POOMA with 100 M68020 processors [12]. They achieved a measured speedup of 32 for an 352×240 image sequence, or an equivalent of 0.5 fps. Motion estimation was parallelized temporally, in which the video sequence was assigned to a set of nodes, with the rest of the coding process parallelized spatially by partitioning each frame into subsets of consecutive blocks. With this scheme, the scalability was dependent on the allocation of nodes to various coding functions, which was only determined experimentally. Akramullah, Ahmad, and Liou achieved real time performance of MPEG-2 coding on the Intel Paragon® XP/S using purely spatial partitioning and CIF format, an equivalent of a speedup of 128 on 330 nodes [13], [14]. The frame data was evenly partitioned by a two dimensional grid before mapping to the processors. The distribution of input frame data was such that the processors were allocated overlapped frame data so as to reduce inter-processor communication during motion estimation. But there was no indication of how the reconstructed data was handled. Also, in their calculation of coding delay, the communication delay between processors was not taken into account. For parallelization, this overhead cannot be neglected since inter-processor communication time can be substantial, largely determined by the number of nodes and the interconnection network used. Shen, Rowe, and Delp achieved 41-fps MPEG compression of CIF video sequences using 100 nodes of Intel Touchstone Delta or 144 nodes of Intel Paragon [15]. They used temporal parallelism in which groups of pictures (GOP) were encoded in parallel with each node processes a GOP. They highlighted the fact that data communication accounts for more than half of the total coding time. They used a custom I/O queue to reduce data access contention. Although this method achieved real-time performance, it requires a large number of frames being ready before the encoding actually starts, e.g., 100 nodes with GOP of 6 frames requires a total of 600 frames. Therefore, there is a delay of tens of seconds between the input and output bit stream. Besides, their temporal parallelism is limited to coding standards with GOP structure, which is not applicable to H.261. Shen and Delp further enhanced their method using spatial parallelism and as a result achieved 33 fps for a CCIR601 test sequence with 512 nodes [16]. In this case, each GOP is

Manuscript received November 5, 1997; revised June 1, 2000. This work was supported by Texas Instruments Tsukuba Research and Development Center, Japan, and by the University Grants Committee, Hong Kong, Area of Excellence in Information Technology, under Grant AOE98/99.EG01. This paper was recommended by Associate Editor R. Stevenson.

The authors are with the Department of Electrical and Electronic Engineering, the University of Hong Kong, Hong Kong, SAR, China (e-mail: nyung@eee.hku.hk; kkleung@eee.hku.hk).

Publisher Item Identifier S 1051-8215(01)00671-1.

processed by a group of processors in which each frame is spatially partitioned into slices before being mapped onto the group. For input data, the entire GOP is sent to each slave to reduce inter-processor data dependency and communication. Further, the output from each group is saved as separate files that require offline concatenation. While real-time performance is achieved, there exists a constant delay ranging from 10–30 seconds as before. Similarly, Agi and Jagannathan implemented an MPEG-1 encoder on a network of workstations and a CM5 system using temporal parallelization based on GOP [19]. They used the GLU master/worker architecture in which a master node handles scheduling of function execution to a number of worker nodes at run time. The system is fault-tolerant in that the master can detect the failure of workers or entry of new workers. A speedup of 7.5 on a 12-node cluster of Sun SPARC 2 was achieved, an equivalent of 3 fps, and 4.5 fps was achieved on a 16-node CM5.

Adopting a more dedicated approach, Akiyama *et al.* outlined a pipelined structure of dedicated digital signal processors for different stages of the coding computation [17]. Their simulation showed that computation time for each stage of the pipeline is within the limit for real-time encoding, but no implementation was given. Bouville *et al.* developed a flexible platform based on an array of TMS320C80 processors, where dedicated buses were used for video input and output and dedicated hardwired block-matching chips were used for motion estimation [18]. They adopted spatial parallelization in which each TMS 320C80 Multimedia Video Processor (MVP) processes a region consisting of horizontal strips of macroblocks in the frame. Within the MVP, the Master Processor (MP) is for upper-level bit stream generation and rate control, one Parallel Processor (PP) for lower-level bit stream generation, and the other 3 PPs for macroblock processing. Performance figures were calculated by counting clock cycles of various coding modules, where implementation results are yet to be seen.

As observed, most of these implementations exploited spatial or temporal data parallelization experimentally using general purpose or dedicated multiprocessors. Speedup figures and frame rates are usually presented. As realized by much research, it is crucial that both the spatial- and temporal-domain decomposition are fully exploited and the eventual parallel algorithm matches the parallel architecture [21]–[23]. Therefore, it is the purpose of this research to investigate the effect of parallelizing computations and communications in the spatial, temporal, and both spatial-temporal domains through the study of frame rate, speedup, and implementation efficiency. The whole investigation was focused on the H.261 coding standard because it has a reasonable degree of complexity, data dependency, communication constraints, and represents the basis of the four standards mentioned earlier. It was implemented on an IBM SP2® supercomputer, where a methodical approach has been adopted to model, predict and measure the execution time, computation time, communication time, idle time, speedup, frame rate, and efficiency with respect to the parallelization methods used and the number of processors (nodes) employed. This investigation leads to the development of four parallel algorithms. The first two exploited the spatial data parallelism in each video frame, of which one utilizes a single communication master, and the other attempts to distribute communications across three masters. To test the algorithms, two video formats were used: 352×240 and QCIF, as both are commonly tested by other researchers and supported by the H.261 software encoder. A speedup (frame rate) of close to 15 (15 fps) was achieved for 352×240 video on 24 nodes, and 13 (37 fps) for QCIF video. The single-master spatial algorithm performs better for $n < 10$, with efficiency ranges from 60% to 90%, while the multiple-master spatial algorithm is superior for $n > 10$, with efficiency ranges from 60% to 70%. The latter two algorithms exploited both the temporal and spatial data parallelism, with the pure temporal case given as a special case. They take into account inter-frame data dependency and exploit temporal parallelism by a pipeline coding structure, while spatial parallelism is implemented when coding a frame. A speedup (frame rate) of around 14 (14 fps) was achieved for 352×240 video on 23 nodes, and 12.5 (35.7 fps) for QCIF. Their implementation efficiencies vary within 50%–60% for a wide range of n.

The organization of this paper is as follows. Section II outlines the sub-functions of the H.261 coding standard, and introduces a serial version and its delays measured on the IBM SP2® multiprocessor. Section III discusses the parallel methodology and preliminary considerations before developing the algorithms. Section IV presents the method used for parallelizing the encoder spatially. Section V describes the parallelization of the encoder in both the spatial and temporal domains. Section VI discusses and analyzes the measured results. This paper is concluded in Section VII.

II. H.261 Encoding Algorithm

A. Functional Blocks

The H.261 encoder is a hybrid of inter-picture prediction and transform coding for reducing temporal and spatial redundancy of the video respectively. The functional architecture of the coding algorithm is depicted in Fig. 1, where the major components are the motion estimation/compensation (ME/MC), discrete cosine transform (DCT), quantization (QUANT), zigzag arrangement (ZIGZAG), and variable length entropy coding (VLC). The decoding part basically consists of a receiving buffer, a VLC decoder, and performs the inverse of the following: ZIGZAG, QUANT, DCT, and MC. For simplicity, the coding control unit is not shown.

In the H.261, each frame is divided into a number of macroblocks (MBs) of size 16×16 pixels, which is the basic data unit for motion compensation. An MB is INTER coded if there is motion compensation; otherwise, it is INTRA coded. For INTER-coded MBs, the pixels are coded as motion-compensated residues after subtracting by the pixels in the reference frame. Since the residues are usually smaller in magnitude than the pixels themselves, data compression is achieved by coding the residues and the motion vector. Motion estimation is the process of searching for the closest MB from the reference frame within the search area, which is a bounded and enlarged area with a spatial position offset from the position of the currently coded MB. After motion compensation, the frame of pixels or residues is coded to reduce spatial data redundancy. The frame is divided into blocks of 8×8 pixels, aligned with

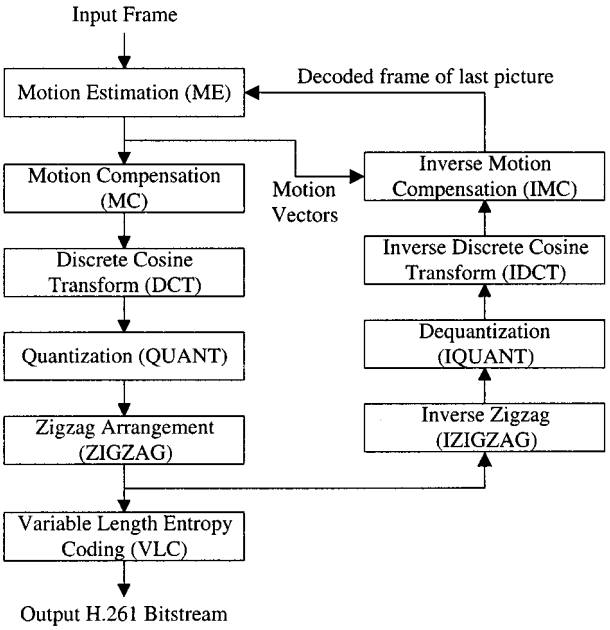


Fig. 1. Functional block diagram of an H.261 encoder.

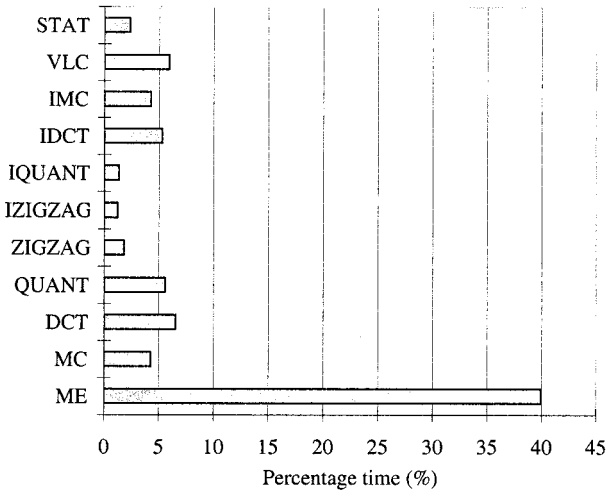


Fig. 2. Percentage encoding time break down.

the MB boundary for transform coding, quantization, zigzag traversal, and variable-length coding. After encoding, the frame is decoded and referenced by the next frame for motion estimation and compensation.

B. Serial H.261 Encoding Performance

The IBM SP2® system employs a purely distributed memory architecture in which inter-processor communication is performed via either the Ethernet or the High Performance Switch (HPS) through message passing [24]–[26]. The platform used for this investigation is a 32-node system at the University of Hong Kong. Of the 32 nodes, 24 can be used exclusively by an application within a limited time window. Each node consists of a 66.7 MHz POWER2® RISC processor with 64-MB main memory and a 2-GB disk storage, providing a peak performance of 266 MFLOPS. The HPS is a bi-directional multistage interconnection network that allows simultaneous message passing between different pairs of nodes. The measured inter-processor communication bandwidth and latency are 31 MB/s and 14 ms, respectively, using the U.S. protocol, on the AIX 4.2 operating system. The tools available for parallelization are the parallel operating environment, MPL, PVM, MPI, C/C++ compilers, Fortran and HPF. A *LoadLeveler* is also available for resource allocation and queuing of applications [27].

The software H.261 video encoder used is the PVRG-P64 Codec from the Portable Video Research Group at Stanford [28]. This codec accepts several image formats, such as CIF (352×288), QCIF (176×144), and 352×240. A number of parameters can be altered including frame rate, range of motion estimation search window, frame skip index, and quantization value. Rate control can be performed at the frame level in which the frame skip index (default $= 1$, i.e., no frame skip) can be varied to drop frames on demand. For motion estimation, the full search scheme is used in the search window. The maximum search window range is ± 15 pixels, while the default is ± 7. For each MB in the search area, the Sum-Absolute-Difference (SAD) is used for similarity comparison. The encoder has a method to speed up the SAD evaluation that if the sub-sum of the absolute difference exceeds the minimum SAD found so far, then the search position is skipped without further evaluating the complete SAD. Thus the time taken for motion estimation is dependent on the data content (motion).

The above serial program was implemented on an SP2 node and timing figures were taken using *gettimeofday*( ) functions enclosing major functional blocks. The program was compiled with optimization options "-O3" and "-qarch = pwr2", and some of its codes were also optimized manually [29]. It was executed through the *LoadLeveler* for exclusive use of the node at run-time. For encoding, the input video was 39 frames of the "table tennis" sequence in 352×240 resolution. Each frame has three files corresponding to the Y, Cr, and Cb components.

Fig. 2 depicts the median percentage time of each block over 3–5 runs for all 39 frames. As expected, motion estimation contributes almost 40% of the encoding delay. The DCT and VLC contribute around 7%. The IDCT and QUANT contribute around 5%, whereas the rest are 2%–4%. An extra function shown here is STAT, which calculates the coding error statistics. The average frame time measured is approximately 0.9882 s.

III. Methodology and Preliminary Considerations

A. Parallelization Methodology

Conceptually, the parallelization methodology adopted in this research consists of four steps: partitioning, communication, agglomeration, and mapping [30], [31]. In the first step, *partitioning* can be on data (domain decomposition), or on computation (functional decomposition). The step *communication* determines the communication required to coordinate task execution, defining appropriate channels and specifying the messages that are to be sent and received. The step *agglomeration* evaluates the task and communication structures defined in the first two steps with respect to performance requirements and the implementation platform, aiming to reduce communication costs by increasing granularity while keeping flexibility of the parallelization. Lastly, *mapping* assigns each task to a processor. The

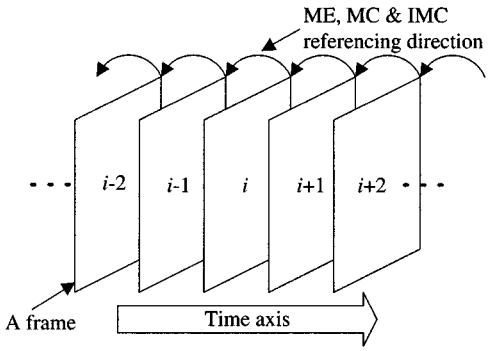


Fig. 3. Inter-frame data dependency.

goal of a mapping scheme is to minimize total execution delay, where two guiding principles are normally observed: concurrent tasks are executed on different processors, and tasks that communicate frequently are executed on the same processor. In this research, domain decomposition is chosen, in which the input frame data or frames are partitioned into a number of units and are mapped to the processors for computation, while the computations performed by each processor are identical.

B. Data Partitioning Issues

For spatial data partitioning, a unit can be as small as a pixel, although such fine grain partitioning introduces a huge amount of communications. As an MB is an inseparable unit in coding, it seems logical to consider an MB as the base unit rather than smaller. If a unit is larger than an MB, the degree of data parallelism will be unavoidably reduced too. Therefore, if an MB is treated as the smallest data unit, we still have plenty of rooms for parallelization as most picture formats contain several hundreds of MBs. In general, if the MBs are evenly partitioned, for a frame containing m MBs and the system having n processing nodes, each node would be allocated $\lceil m/n \rceil$ MBs. However, the MBs may be partitioned unevenly based on workload balancing criteria, if so desired [32]. The algorithms presented in this paper employ even partitioning without workload balancing.

C. Data Dependency Issues

There are two types of data dependencies in MB processing. First, in the temporal direction, data dependency exists between successive frames while performing ME, MC and IMC, where references to the decoded MBs in the search area in the previous frame are required, as depicted in Fig. 3. Second, there is data dependency between neighboring MBs in a frame due to MB header fields being coded with differential coding. This makes the VLC of MB headers inherently serial and has to be performed sequentially over the MBs in the frame. Apart from these, all the other steps can be executed independently for each MB. The algorithms presented here attempt to tackle these dependencies so as to reduce communications.

D. Communication Issues

To encode MBs in a node, there are three types of data communication: distribution of input MB, collection/distribution of decoded MBs, and collection of encoded MBs and statistics. Assume the input data is acquired from a video source or a set of files, it has to be distributed to the other nodes for coding. Similarly, the decoded data needs to be collected and re-distributed, while the encoded bitstream and statistics need to be collected and output. For the S1 algorithm, a node is chosen for doing all three communications. For the S2 algorithm, these communications are shared between three masters. In general, inter-processor communications are required for all three cases, which form a substantial overhead affecting the overall execution time and the scalability of the implementation.

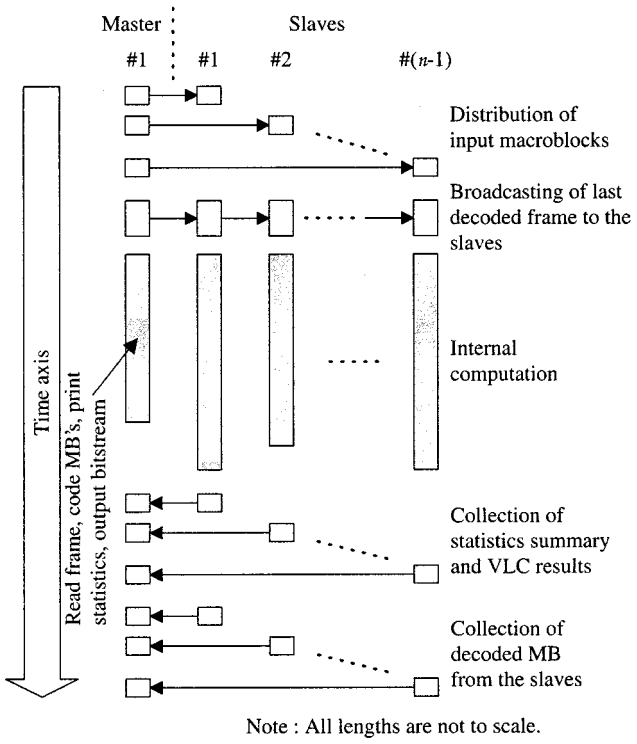


Fig. 4. S1 algorithm.

IV. Parallelization in the Spatial Domain

A. Spatial Parallel Algorithm-S1

After considering the issues discussed in the preceding section, one of the parallel algorithm developed is a purely spatial parallel algorithm (S1), where a master is designated for handling the communication and ordering of MBs, and the slaves are responsible for the computation [31]. The data partitioning scheme is such that the MBs of the frame are arranged in the order as the coded bit stream. The sorted list of MBs is divided into segments of MBs for allocation to the slaves. In this way, the slaves are able to code the MB headers with differential coding without referencing each other, except for the first MB allocated to each slave. However, this can be overcome by having the slave computes the referenced MB itself. As depicted in Fig. 4, Master#1 reads the input frame data, reorganizes the array of pixels into an ordered array of MBs, then distributes them to the slaves. Master#1 also broadcasts the last decoded frame to the slaves, which is collected at the last iteration. Upon receiving the decoded frame and the MBs from the current frame, the slaves perform encoding in parallel. In this case, the master also codes m_0 MBs where $m_0 = \max(\lfloor N_{\mathrm{MB}}/n \rfloor - 30, 0)$ for a

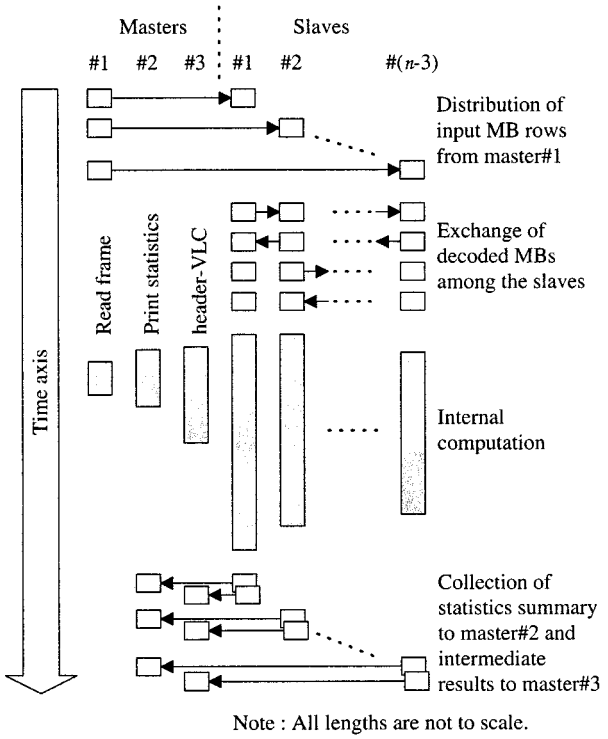


Fig. 5. S2 algorithm.

frame consisting of N_{MB} MBs. This number is chosen to keep Master#1 busy while it is not communicating with the slaves. Finally, Master#1 collects the statistics, coded bit stream and the decoded MBs from the slaves. In this algorithm, most of the computations are carried out in the slaves in parallel, with all the communications being managed by a master, in which $\{(N_{\mathrm{MB}}-m_0) \bmod (n-1)\}$ slaves are assigned $\lfloor (N_{\mathrm{MB}}-m_0)/(n-1) \rfloor + 1$ MBs each, while the others get $\lfloor (N_{\mathrm{MB}}-m_0/(n-1) \rfloor$. Although the number of MBs distributed to each slave is near even, due to the difference in motion content in each MB and the serial encoder used, the computing delay for each MB, or a group of MBs, are different.

B. Spatial Parallel Algorithm-S2

This algorithm aims to spread the serial communications having three masters separately responsible for handling the distribution of input MBs, collection of encoded MBs, and the collection of statistics, as depicted in Fig. 5. Instead of handling the decoded MB by a master, communication is further reduced by having these data exchanged locally among the slaves according to the MB allocation. For example, if each slave is allocated a row of MBs, then it exchanges decoded MBs with only two slaves holding the upper and lower neighboring rows. The MBs in the frame are ordered from top to bottom and left to right. With this scheme, the VLC of MB headers have substantial data dependency between those allocated to different slaves. In order to reduce this sequential component, VLC is further divided into two functions, namely header-VLC and TC-VLC, corresponding to the VLC of the MB headers and transformed coefficients, respectively, which TC-VLC can be parallelized while header-VLC is performed sequentially in Master#3. The data flow is as follows: once the input MBs are received by the slaves from Master#1, coding begins. When coding is completed, Master#2 collects the statistics, and in parallel, Master#3 collects the TC-VLC results and performs header-VLC before sending the bit stream to the standard output. In this algorithm, the masters do not perform any MB coding.

C. Performance Prediction

Let n be the number of nodes available for parallelization; $T_{\mathrm{cp}}(j)$ be the internal computation time of node j, where $j = 1, 2, \ldots, n$; M_i be the size in bytes of an input frame; M_o be the average size of a coded frame; M_s be the size of a statistics record; T_w be the asymptotic bandwidth of the communication channel in second per byte; T_s be the overall startup time of the communication channel; T_s' and T_w' be the constants in the broadcast delay expression [20]. The time for encoding a frame using the S1 algorithm [31] is given by

$$\begin{aligned} T_{\mathrm{frame}} = & \left\{(n-1)\left(\frac{M_i}{n}T_w + T_s\right)\right\} \\ & + \left\{(n-1)\left[\left(\frac{M_o}{n}T_w + T_s\right) + (M_s T_w + T_s)\right]\right\} \\ & + \{\max\{T_{\mathrm{cp}}(1) + T_{\mathrm{in}} + T_{\mathrm{out}}, \\ & \quad T_{\mathrm{cp}}(j; j = 2, 3, \ldots, n)\}\} \\ & + \left\{(n-1)\left(\frac{M_i}{n}T_w + T_s\right)\right\} \\ & + \{T_s' \log(n) + [T_w' \log(n)]M_i\} \end{aligned} \tag{1}$$

where the first $\{\}$ represents the time taken for the master to send M_i/n bytes to the slaves; the second $\{\}$ represents the time taken for collecting the VLC results and statistics; the third $\{\}$ represents the computation critical path; the fourth $\{\}$ represents the time taken to collect the decoded data; and the fifth $\{\}$ represents the time taken to broadcast the decoded frame. Also, T_{in} represents the time for reading a frame and rearranging the MB, and T_{out} represents the time for writing the encoded results into an output bit stream.

For the S2 algorithm, assume T_{in} and the statistic collection time is small compared with $T_{\mathrm{cp}}(j)$ and T_1, the frame encoding time is given by $T_{\mathrm{frame}} = \max\{T_1, T_2\}$, where

$$T_1 = \left\{(n-3)\left(\frac{M_o}{n-3}T_w + T_s\right) + T_{\mathrm{out}}\right\} \tag{2}$$

$$\begin{aligned} T_2 = & \max\{T_{\mathrm{cp}}(j); j = 4, 5, \ldots, n\} \\ & + \{M_s T_w + T_s\} + \left\{\frac{M_i}{n-3}T_w + T_s\right\} \\ & + \left\{\frac{M_o}{n-3}T_w + T_s\right\} + \left\{4\left(\frac{M_i}{R}T_w + \left\lceil\frac{n}{R}\right\rceil T_s\right)\right\} \end{aligned} \tag{3}$$

where

- R number of MBs in a column;
- T_1 delay due to the third master, which consists of the time taken to collect the TC VLC, and to compile the VLC header (T_{out});
- T_2 delay due to slave computations

in which the first $\{\}$ represents the computation critical path; the second $\{\}$ represents the time taken to send the statistics to

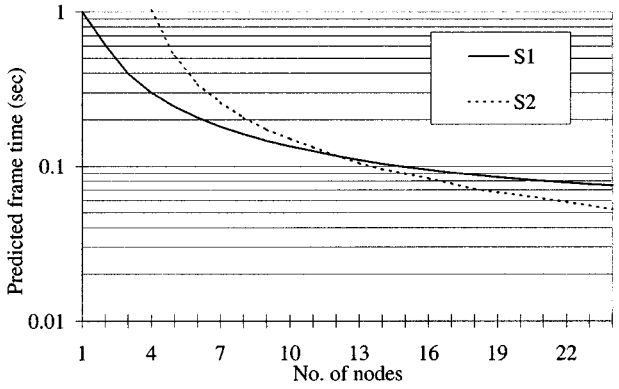


Fig. 6. S1/S2 Predicted frame time.

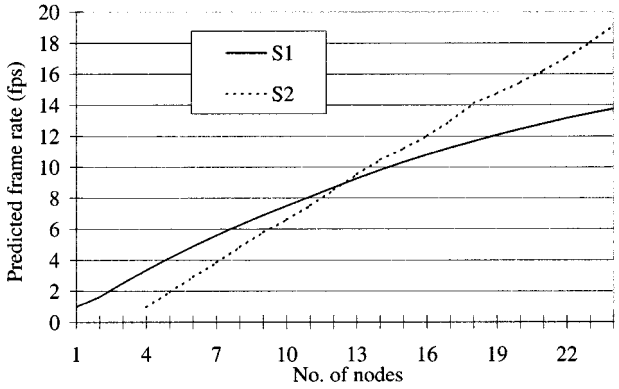


Fig. 7. S1/S2 Predicted frame rate.

the second master; the third { } represents the time taken for the slaves to receive the input MBs; the fourth { } represents the time taken to send the VLC result to the third master; and the fifth { } represents the time taken for a slave to exchange MBs with its neighbors. For each neighbor, there is an exchange operation, which is made up of a send and a receive operation. Since the number of neighbors equals to $2\lceil n/R \rceil$, the total number of messages involved is $4\lceil n/R \rceil$ and the total amount of decoded data exchanged is $4M_i/R$.

Figs. 6 and 7 depict the predicted frame time and frame rate for both algorithms for 352×240 video. In the calculation, the value of T_w is $1/(31\times10^6)$; T_s is 14×10^{-6}; T'_w is 0.029×10^{-6}; T'_s is 52×10^{-6}; M_i is 126 720; M_o is 1350; M_s is 3000; T_{out} is 1.35×10^{-2}; N_{MB} is 330 and R is 15. T_{in} is 2.38×10^{-2} for S1 because of the MB ordering, and 7×10^{-3} for S2.

V. Parallelization in Both the Temporal and Spatial Domains

A. Parallelization Model and Frame Dependency

To achieve parallelization in the temporal domain, the smallest temporal unit is a frame. Although coding a frame takes considerable time (~1 fps in this case), the average frame time may be improved if the frames can be coded in a pipeline over a number of clusters with each frame spatially parallelized on a cluster, where a cluster is a group of nodes. This model is considered general because if there is only one node in each cluster, then the parallelization is purely temporal. However, if there is only one cluster with multiple nodes, then the parallelization is purely spatial.

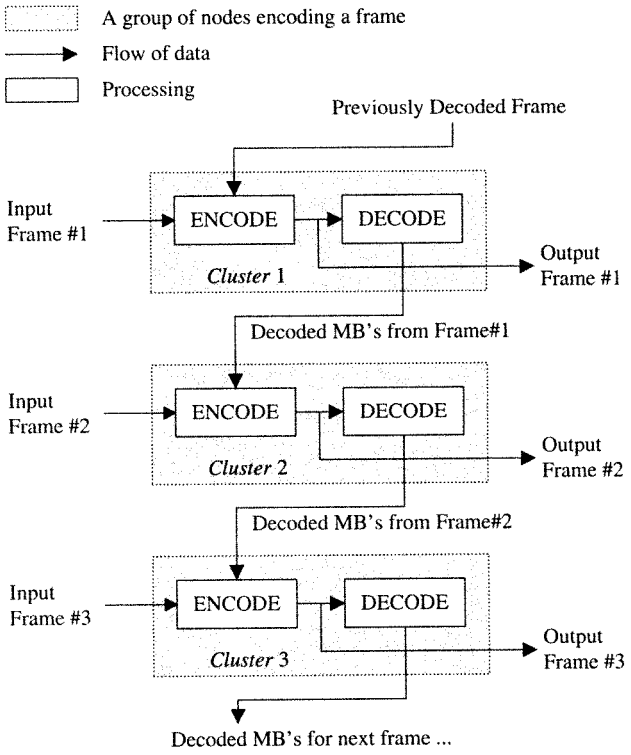


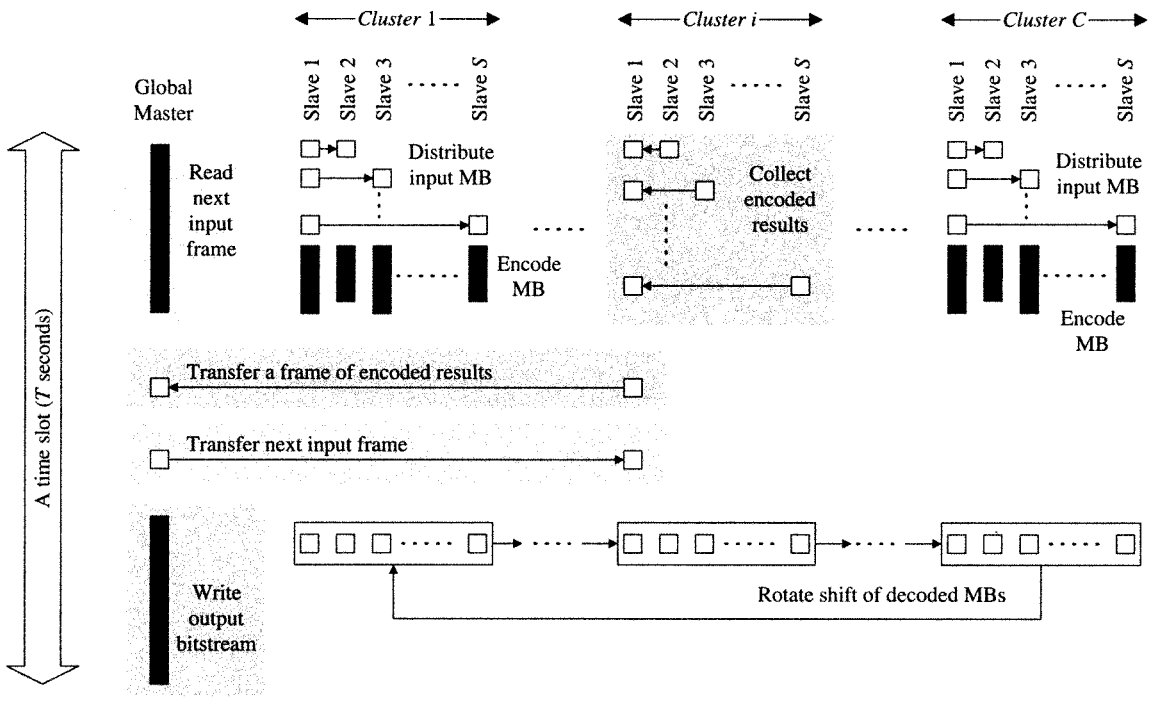
Fig. 8. Temporal data flow across a number of clusters.

Fig. 8 depicts the data flow across a number of clusters. The frame dependency is that the decoded frame from one cluster is needed for ME in the neighboring cluster. Apart from this, the coding of the next frame can start while the coding of the current frame is still in progress, as long as a minimum number of MBs needed for ME/MC/IMC are available then.

B. Spatial-Temporal Parallel Algorithm-ST1

Base on the above model, the ST1 algorithm was developed with all the functions being parallelized except the header-VLC. Fig. 9 depicts the algorithmic structure in which each frame is coded by one of C clusters and a global master is responsible for reading/distributing input frames, collecting encoded results, header-VLC. and writing output bit stream. Inside each cluster, slave 1 is designated for communicating with the global master. This method of centralizing parallelization control in slave 1 simplifies the interface between the clusters and the global master. Assume n nodes in the implementation, if there are C clusters. and each cluster contains S nodes, then $n = SC + 1$. The computation is modeled as time slots such that in each time slot, the S slaves in a cluster process S MBs in total. The MBs are processed starting from the left edge of the frame and extends to the right. Assume a frame has N_{COL} columns and R rows of MBs. To initiate the coding of the next frame, the minimum number of MBs required is R MBs from the first column plus $S+1$ MBs from the secnd column. The communication of decoded MBs is such that each slave of cluster i sends its currently decoded MB to the S slaves of cluster $i + 1$ and receives the S decoded MBs from cluster $i - 1$.

As illustrated in Fig. 9, there are four types of time slots or cycles associated with this algorithm. A *normal slot* (T_{NS}) con-



Note :

1. A master slot with frame input only (T=T_{MSi}) includes all portions in the diagram except those shaded
2. A master slot with frame output only (T=T_{MSo}) includes all portions in the diagram except those shaded
3. A master slot with both frame input and output (T=T_{MS}) includes every portion, including both shaded portions.
4. A normal slot (T=T_{NS}) has no global master communication and does not include any shaded portions.

Fig. 9. ST1 algorithm.

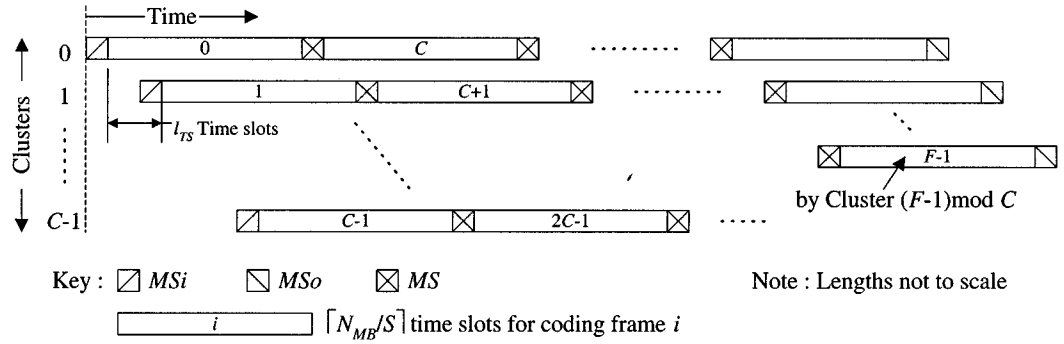


Fig. 10. Time line for ST1.

sists of coding the MB only. An *input master slot* (T_{MSi}) includes reading of the next input frame and transferring of the frame data to a cluster, during which the other clusters may be performing MB coding. An *output master slot* (T_{MSo}) includes the collection of encoded results to slave 1 within a cluster, transferring the encoded data to the global master and writing the output bit stream, while other clusters may be performing MB coding. An *I/O master slot* (T_{MS}) includes reading of a frame, transferring to and from the clusters, writing the output bit stream and MB coding.

The working of the algorithm begins with reading of a frame data by the global master and transferring the data to the clusters. Then, Cluster 1 codes $(R + S + 1)$ MBs using $l_{\mathrm{TS}} = \lceil (R+S+1)/S \rceil$ cycles and the decoded MBs are sent to Cluster 2 as soon as they are available. The coded results are then collected from the slaves to slave 1 within a cluster, and transferred to the global master. Therefore, at the start of the process, the cycles are mainly T_{MSi}, whereas at the end, the cycles are mainly T_{MSo}. The cycles in between are either T_{MS} or T_{NS}. This is depicted in Fig. 10, in which some clusters begin coding later than others, and at the end of the sequence, some clusters complete earlier and become idle. While the clusters are filled with normal or master slots, sustained computation in each cluster is maintained.

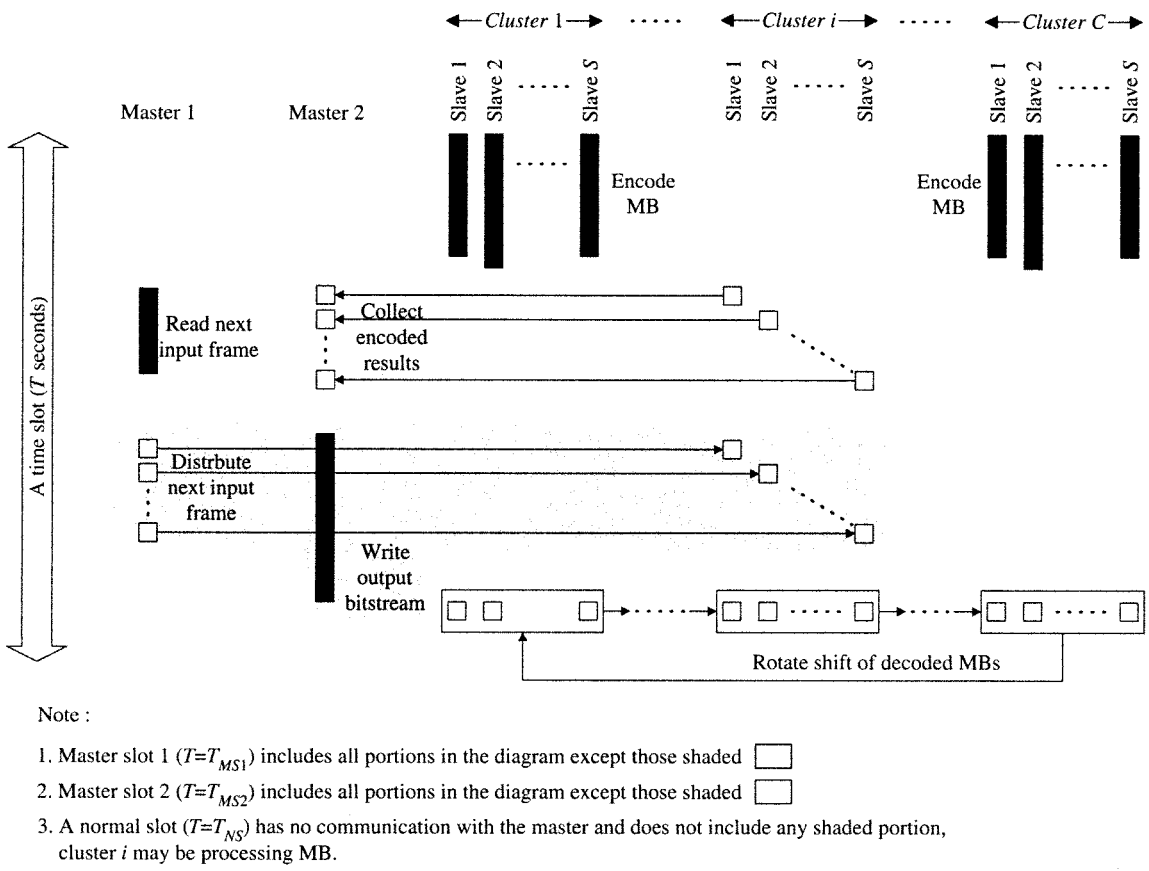


Fig. 11. ST2 algorithm.

C. Spatial-Temporal Parallel Algorithm-ST2

This algorithm was developed for sharing the cluster communication between two global masters. In each cycle, a cluster processes a column of R MBs, where $R \geq S$. The R MBs are ordered from top to bottom and then evenly partitioned to the S slaves. A frame requires at least two columns of MBs processed before the next frame can start. A cluster takes N_{COL} cycles to code a frame. Denote the jth slave of cluster i by $\mathrm{SL}(i,j)$. Spatially, each $\mathrm{SL}(i,j)$ processes $\lceil R/S \rceil N_{\mathrm{COL}}$ MBs per frame. At the end of each time slot, $\mathrm{SL}(i,j)$ sends all of its decoded MBs to $\mathrm{SL}(i+1,j)$. Also, $\mathrm{SL}(i,j)$ sends its first decoded MB to $\mathrm{SL}(i+1,j-1)$ and its last to $\mathrm{SL}(i+1,j+1)$. Afterward, $\mathrm{SL}(i,j)$ receives the required decoded MBs from $\mathrm{SL}(i-1,j)$, $\mathrm{SL}(i-1,j-1)$ and $\mathrm{SL}(i-1,j+1)$. Therefore, in each cycle, for decoded MB communication, there are totally six messages involving $(2\lceil R/S \rceil + 4)$ MB of data.

As depicted in Fig. 11, one global master is responsible for reading the input frame data and distributing them to each slave, and the other is responsible for collecting the results from the slaves and writing the output bit stream. There are three different types of cycles in this case. A *normal slot* (T_{NS}) performs MB coding only. A *master slot 1* (T_{MS1}) consists of reading the frame data and collecting results from a cluster, while other clusters are coding MBs. A *master slot 2* (T_{MS2}) consists of writing the output bit stream, distributing the next frame data to a cluster, and coding the MBs in other clusters. As there are two global masters, some of the read/write/transfer/coding cycles can overlap with each other. For example, when master 1 reads the next frame, results can be collected into master 2, as opposed to ST1, in which the global master communicates with a single representative in each cluster, the global masters in ST2 communicate with each slave in each cluster directly.

D. Performance Prediction

For the ST1 algorithm, in coding F frames, the first C frames require CT_{MSi} while the last C frames require CT_{MSo}. In between, there are $(F-C)T_{\mathrm{MS}}$ involving both frame input and output. The total execution time T_{exe} in coding F frames is given by

$$T_{\mathrm{exe}} = CT_{\mathrm{MSi}} + T_{\mathrm{MSo}} + (F-C)T_{\mathrm{MS}} + (N_{n1} - C - F)T_{\mathrm{NS}} \tag{4}$$

where N_{n1} is the total number of slots given by

$$N_{n1} = \left\{ \left\lceil \frac{F}{C} - 1 \right\rceil \max\left(\left\lceil \frac{N_{\mathrm{MB}}}{S} \right\rceil + 1, Cl_{\mathrm{TS}} \right) \right\} + \left\{ \left\lceil \frac{N_{\mathrm{MB}}}{S} \right\rceil + 1 \right\} + \{[(F-1) \bmod C] l_{\mathrm{TS}}\} + 1 \tag{5}$$

where $l_{\mathrm{TS}} = \lceil (R + S + \mathrm{sign}(S \bmod R))/S \rceil$, and $T_{\mathrm{MSi}}, T_{\mathrm{MSo}}, T_{\mathrm{MS}}$ and T_{NS}, are given by (6) to (9)

$$T_{\mathrm{NS}} = \{(S-1)T_{\mathrm{AR}} + \mathrm{sign}(S-1)T_{\mathrm{AR}}\} + \{(S-1)(M_{\mathrm{MB}}T_w + T_s)\} + \left\{ \frac{T_{\mathrm{seq}}}{N_{\mathrm{MB}}} \right\} + \{2S(M_{\mathrm{MB}}T_w + T_s)\} \tag{6}$$

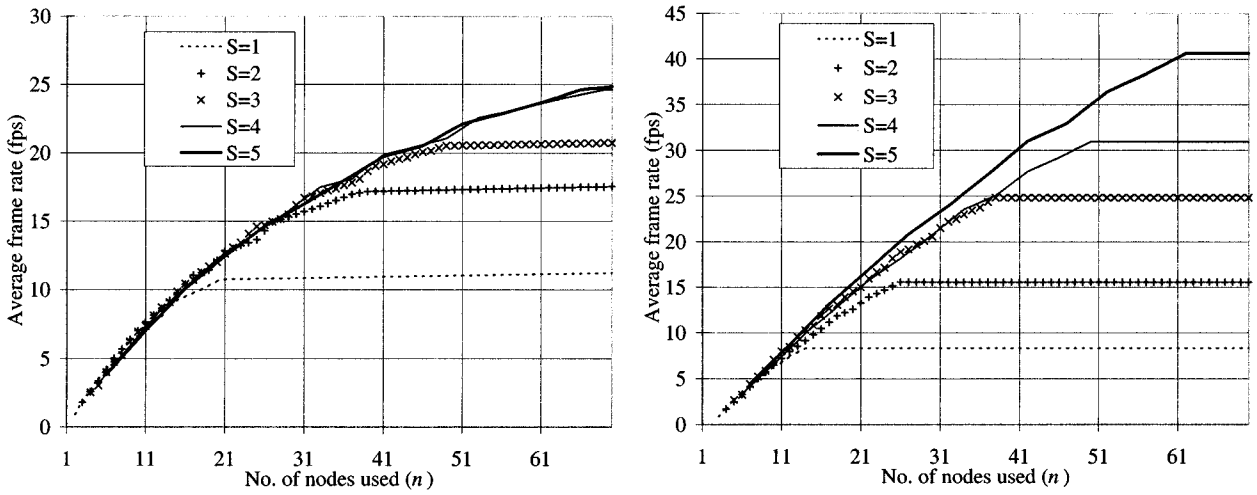


Fig. 12. (a) ST1—Predicted frame rate ($n = SC + 1$). (b) ST2—Predicted frame rate ($n = SC + 2$).

$$T_{\mathrm{MSi}} = \{T_{\mathrm{in}}\} + \{M_i T_w + T_s\} + \{2S(M_{\mathrm{MB}} T_w + T_s)\} \quad (7)$$

$$T_{\mathrm{MSo}} = \left\{(S-1)\left[(M_s T_w + T_s) + \left(\frac{M_o}{S} T_w + T_s\right)\right]\right\} + \{(SM_s T_w + T_s) + (M_o T_w + T_s)\} + \max\{2S(M_{\mathrm{MB}} T_w + T_s), T_{\mathrm{out}}\} \quad (8)$$

$$T_{\mathrm{MS}} = \max\left\{T_{\mathrm{in}}, (S-1)\left[(M_s T_w + T_s) + \left(\frac{M_o}{S} T_w + T_s\right)\right]\right\} + \{(SM_s T_w + T_s) + (M_o T_w + T_s)\} + \{M_i T_w + T_s\} + \max\{T_{\mathrm{out}}, 2S(M_{\mathrm{MB}} T_w + T_s)\} \quad (9)$$

where

M_{MB} size of a MB in bytes;

T_{AR} time for rearranging the input format of an MB;

l_{TS} minimum number of time slots for a frame before the next frame can start;

T_{seq} sequential frame time.

The term $\mathrm{sign}(S-1)T_{\mathrm{AR}}$ in (6) represents the time for MB rearrangement after receiving from slave 1, which is equal to zero when $S = 1$. From (4), the average frame rate is T_{exe}/F.

For the ST2 algorithm, T_{MS1} occurs only once every frame, i.e., number of T_{MS1} is equal to F. Similarly, the number of T_{MS2} is also F. The number of normal slots is $N_{n2} - 2F$, where N_{n2} is the total number of time slots. The total execution time for F frames is given by

$$T_{\mathrm{exe}} = FT_{\mathrm{MS1}} + FT_{\mathrm{MS2}} + (N_{n2} - 2F)T_{\mathrm{NS}} \quad (10)$$

where

$$N_{n2} = \left\{\begin{array}{l}\left\lceil \frac{F}{C} - 1\right\rceil \max\{N_{\mathrm{COL}} + l_{\mathrm{TS}}, Cl_{\mathrm{TS}}\} \\ +N_{\mathrm{COL}} + [(F-1) \bmod C + 2]l_{\mathrm{TS}}\end{array}\right\}$$

l_{TS} equal to 2;

N_{COL} number of columns of MBs in a frame;

$T_{\mathrm{NS}}, T_{\mathrm{MS1}},$ given by (11)–(13).

and T_{MS2}

as

$$T_{\mathrm{NS}} = \left\{\frac{T_{\mathrm{seq}}}{N_{\mathrm{MB}}}\left\lceil\frac{R}{S}\right\rceil\right\} + \left\{2\left\lceil\frac{R}{S}\right\rceil M_{\mathrm{MB}} T_w + 4M_{\mathrm{MB}} T_w + 6T_s\right\} \quad (11)$$

$$T_{\mathrm{MS1}} = \max\left\{\begin{array}{l}T_{\mathrm{in}}, \\ \max\left[\begin{array}{l}(M_s S T_w + T_s S) + (M_o T_w + T_s S), \\ \frac{T_{\mathrm{seq}}}{N_{\mathrm{MB}}}\left\lceil\frac{R}{S}\right\rceil\end{array}\right] \\ \quad + \left[2\left\lceil\frac{R}{S}\right\rceil M_{\mathrm{MB}} T_w + 4M_{\mathrm{MB}} T_w + 6T_s\right]\end{array}\right\} \quad (12)$$

$$T_{\mathrm{MS2}} = \max\left\{\begin{array}{l}T_{\mathrm{out}}, \\ \max\left[M_i T_w + ST_s, \frac{T_{\mathrm{seq}}}{N_{\mathrm{MB}}}\left\lceil\frac{R}{S}\right\rceil\right] \\ \quad + \left[2\left\lceil\frac{R}{S}\right\rceil M_{\mathrm{MB}} T_w + 4M_{\mathrm{MB}} T_w + 6T_s\right]\end{array}\right\} \quad (13)$$

In (11), the first $\{\}$ represents the delays in the slaves, where each cluster processes a column of MBs such that each slave is allocated $\lceil R/S \rceil$ MBs. The second $\{\}$ represents the communication of decoded MBs between clusters. In (12), the inner $\max[]$ represents the maximum between data collection time and MB coding time; the lowest term represents the communication of decoded MBs and is added to the $\max[]$ term since the slaves that communicate with the masters also participate in decoded MB communication with other clusters. The term T_{in} represents the frame reading time in Master 1, which is done in parallel with the other slaves. Equation (13) is similar to (12), except that data collection is replaced by the distribution of input MBs and T_{in} is replaced by T_{out}. The average frame rate is given by T_{exe}/F. Fig. 12(a) and (b) depict the predicted frame rate versus n, for different choices of S, for both algorithms. The value of T_{in} is 6.4×10^{-3}; T_{out} is 9.6×10^{-3}; T_{seq} is 1.0; T_{AR} is 5.01×10^{-5}; M_{MB} is 384; F is 39; N_{COL} is 22.

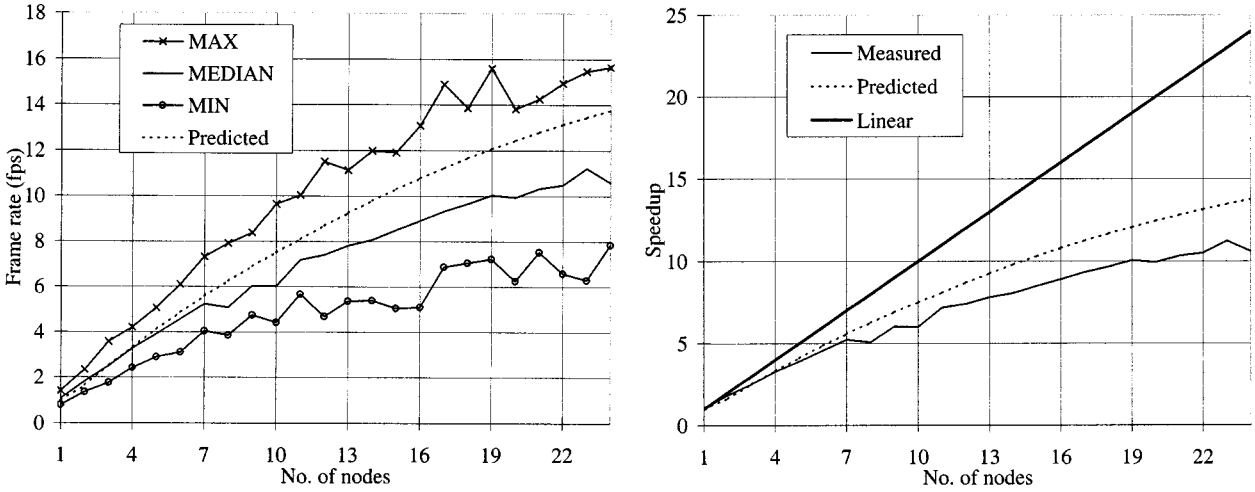


Fig. 13. (a) S1 measured frame rate. (b) S1 measured speedup.

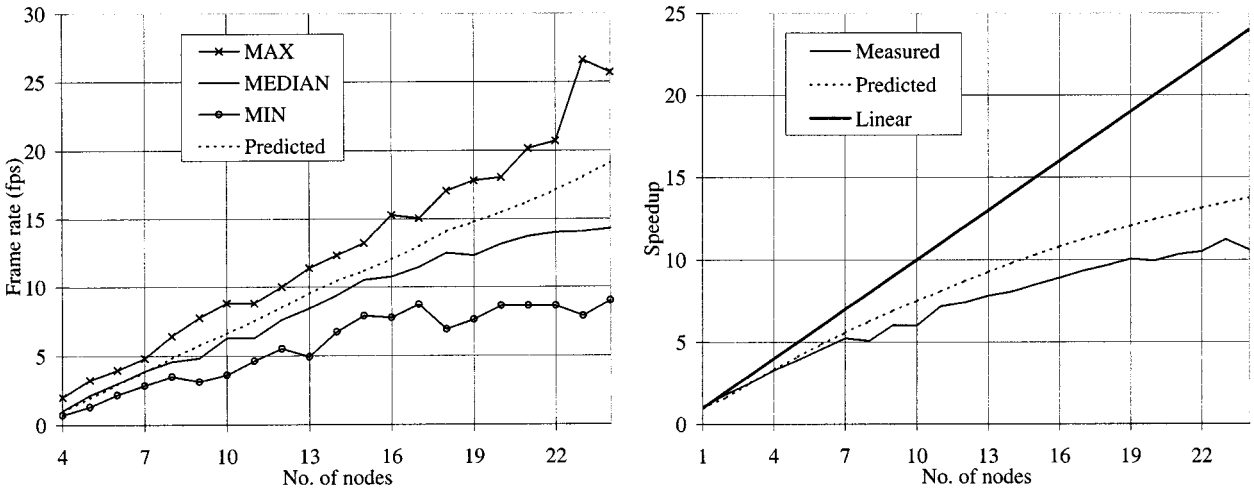


Fig. 14. (a) S2 measured frame rate. (b) S2 measured speedup.

VI. Results and Discussions

A. Data Collection Conditions

The H.261 program was compiled using $mpcc\text{-}O3\text{-}qarch = pwr2$ *[filename.c]*, using single precision integer format. The wall-clock time generated by *gettimeofday*( ) was used to measure the overall execution duration and individual execution time per stage, where all the nodes were synchronized and timed at the start of the execution. Blocking send and receive were used for all the point-to-point communications where fixed startup time and constant channel bandwidth were assumed. The broadcast times were assumed as in Hwang and Xu [20]. The HPS user space communication protocol was used to obtain the best performance from the network. The *LoadLeveler* was used in run-time to ensure exclusive use of the nodes. The program code was running on *Unix*, which would introduce slight variation in the execution time and measurement error. The H.261 parameters were kept at their default values throughout the test. The coded results were checked byte-by-byte against the serial program output. The coded output streams were also decoded and displayed for visual inspection and comparison.

B. Results of the S1 Algorithm

The measured frame rates (maximum, minimum and median) in fps for nodes ranging from 1 to 24, together with the predicted values are plotted in Fig. 13(a) and the corresponding speedup figures are depicted in Fig. 13(b). From Fig. 13, a number of observations can be made. First, the median frame rate tracks the predicted values closely for n less than 8, beyond which it drops off gradually with a frame rate of 11.2 fps at 23 nodes. Such behavior can be explained as when n is small, the number of MBs allocated to each slave is large and therefore computation dominates. As n increases, this number decreases, whereas the communication overhead and idle time increase. Hence, the slope of the curve decreases gradually for $n > 8$, and seems to level off beyond $n > 24$. Second, there is a substantial difference between the maximum and minimum frame rates measured. This is due to the way in which the SAD is calculated as explained in Section II-B , and the parallelization magnifies the difference. Third, the predicted frame rate is optimistic. This is unavoidable, as the overheads related to software control and communication contention have not been accounted for in the prediction. Fourth, there are ripples in both measured curves for

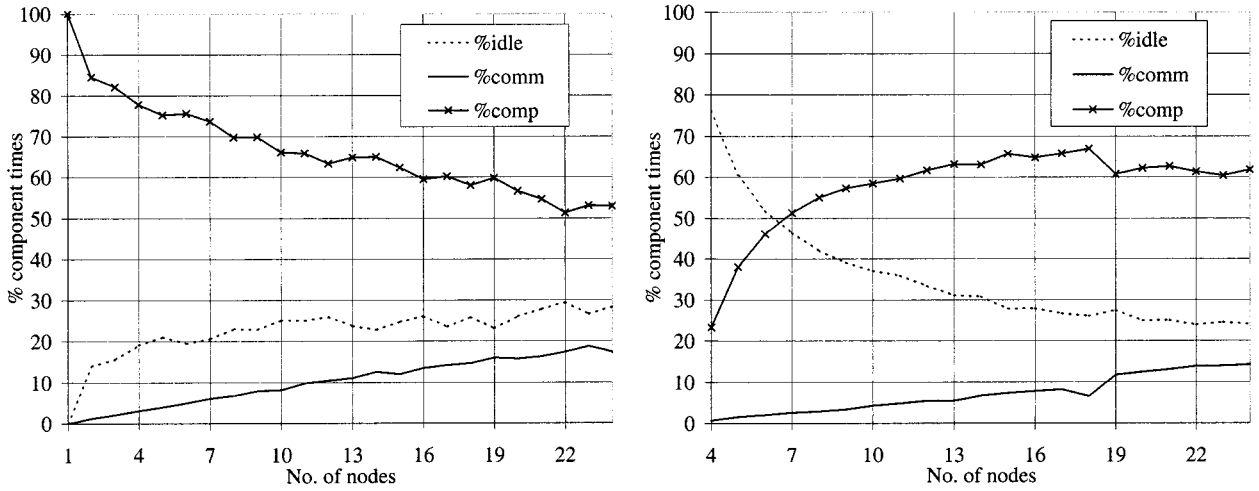


Fig. 15. (a) S1 percentage component time. (b) S2 percentage component time.

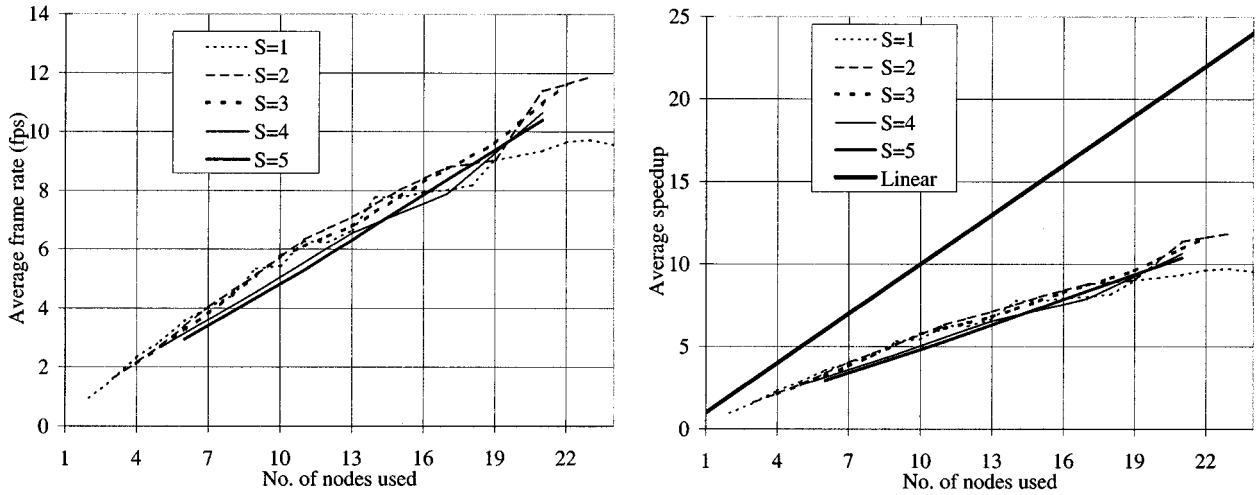


Fig. 16. (a) ST1 measured frame rate. (b) ST1 measured speedup.

$6 < n < 12$ and $n > 19$, which is likely due to measurement error caused by overhead of the operating system and the slightly uneven distribution of MBs.

C. Results of the S2 Algorithm

The frame rates measured for four nodes up to 24 nodes and the predicted values are depicted in Fig. 14(a), whilst the speedup figures are plotted in Fig. 14(b). From Fig. 14(a), it can be seen that, first, the lower bound for n is 4 (3 masters 1 slave). At this n, the frame rate is poor which is because three out of the four nodes are either communicating or idling, with only one node doing useful computation. Second, the frame rate increases linearly up to $n = 17$, and gradually levels off beyond this point. This can be explained as the multiple-master configuration is effective up to this point. Third, the predicted values are again optimistic, for the same reasons as in S1. Fourth, for $n = 24$ nodes, the median speedup is 14.3, which is 27% better than the S1 case. In fact, S2 has poorer speedup for $n < 10$, but better speedup otherwise. This can be explained by examining the percentage component times, as shown in Fig. 15.

Let $P(\mathrm{cp})$ be the percentage of computation time, $P(\mathrm{com})$ be the percentage of communication time, and $P(\mathrm{id})$ percentage of idle time. For the S1 algorithm, $P(\mathrm{cp})$ is high for small n but drops to around 53% at $n = 24$. On the other hand, $P(\mathrm{com})$ and $P(\mathrm{id})$ are small for small n and increase to 25%–30% and 15%–20%, respectively, for $n = 24$. On the contrary, $P(\mathrm{cp})$ of the S2 algorithm is only 23% for $n = 4$, but steadily increases to over 60% at $n = 10$, and drops off to 60% for $n > 17$, while $P(\mathrm{com})$ is very small for small n to about 15% for $n = 24$. For $P(\mathrm{id})$, it shows an opposite trend: large for small n (75%), and small for large n (25%). From these results, it can be argued that centralized communication would achieve better speedup for small n, while distributed communication would achieve better speedup for large n.

D. Results of the ST1 Algorithm

Fig. 16 depicts the measured median frame rates and speedup versus n, with the nodes per cluster (S) being a variable. The curve of $S = 1$ represents the purely temporal case, whereas the other curves represent varying degrees of mixed spatial-temporal parallelization. From Fig. 16, we can observe that first, the

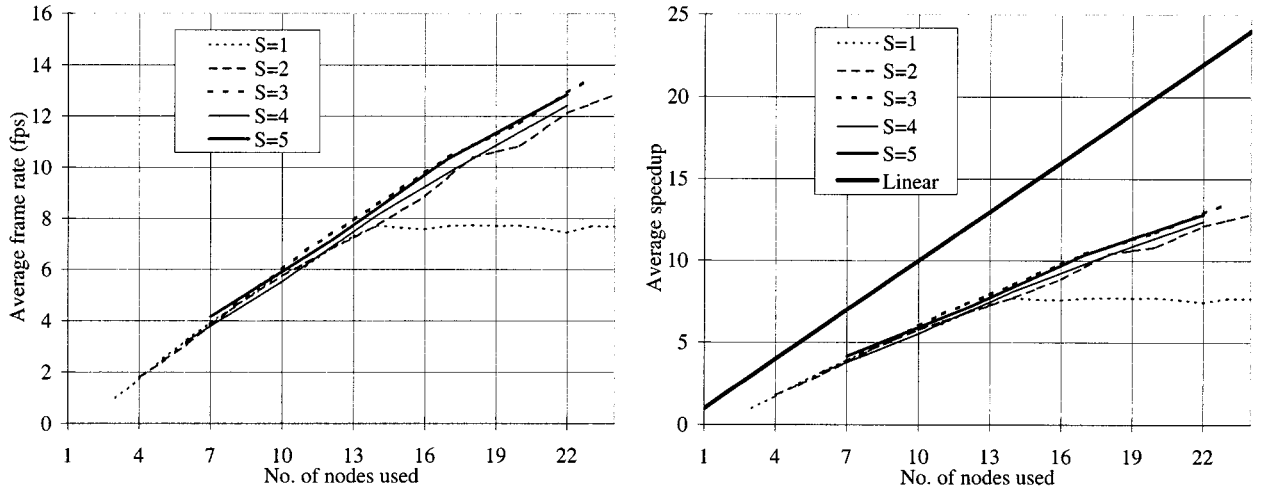


Fig. 17. (a) ST2 measured frame rate. (b) ST2 measured speedup.

curves are reasonably straight and bunched up. Upon close inspection, it is found that the $S = 1$ curve levels off after $n = 19$, and the $S = 2$ curve shows similar tendency, but for larger n. Such phenomenon agrees with the prediction in Fig. 12(a). Furthermore, cases of larger S seem to give poorer performance than those with smaller S, which is not clear in the prediction. Second, out of the five cases, $S = 2$ achieves the best speedup of 12 at $n = 23$, which is slightly better than the S1 case. This is not entirely unexpected, as both algorithms centralize communications onto a single master. However, it should also be noted that the ST1 algorithm is superior for large n and S, as indicated in Fig. 12(a). In theory, over 25 fps can be achieved with $n > 70$ for both $S = 4$ and 5, which is impossible for S1.

E. Results of the ST2 Algorithm

Fig. 17 depicts the measured frame rate and speedup for the ST2 algorithm. It can be observed that the speedup curves for various S are similar for $n \leq 14$. For $n > 14$, the speedup of the pure temporal case $(S = 1)$ remains almost constant at ~7.7, while the rest continue to increase to close to 13.5 in the case of $S = 3$ for $n = 23$. This figure is better than the ST1. Close inspection reveals that the cases with large S perform better than small S, by a small margin. Theoretically, this margin will increase for large n as depicted in Fig. 12(b). Comparing Fig. 17(a) with Fig. 12(b), the $S = 1$ case levels off at a measured frame rate of 7.7 for $n = 14$, some 0.5 fps below the predicted. In general, the measured frame rate or speedup agrees with the predicted but smaller. This once again highlights the effect of software overhead and communication contention on the actual parallel performance. According to the prediction, over 25 fps can be achieved for $n > 36$ and $S > 2$, which is much better than the ST1 case.

F. Summary

Fig. 18 depicts the median speedup for the four algorithms presented in this paper. It is noted that for $n < 10$, S1 achieves the best relative speedup, followed by ST1 $(S = 2)$, ST2 $(S = 3)$, and S2. This can be explained, as for small n, computation dominates. In this case, S1 has more nodes for computation than

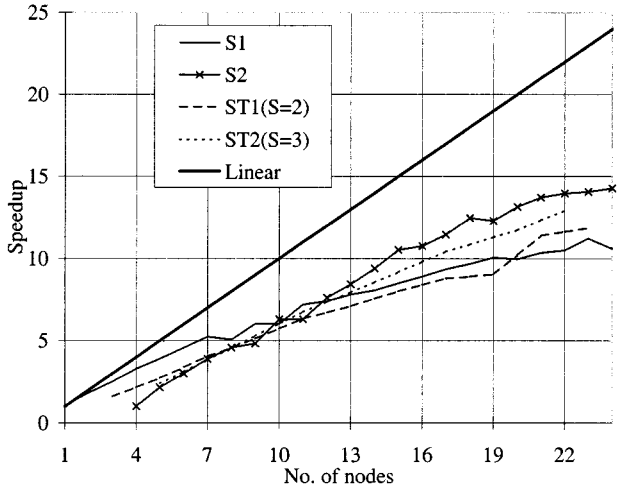


Fig. 18. Comparison of speedup—352×240.

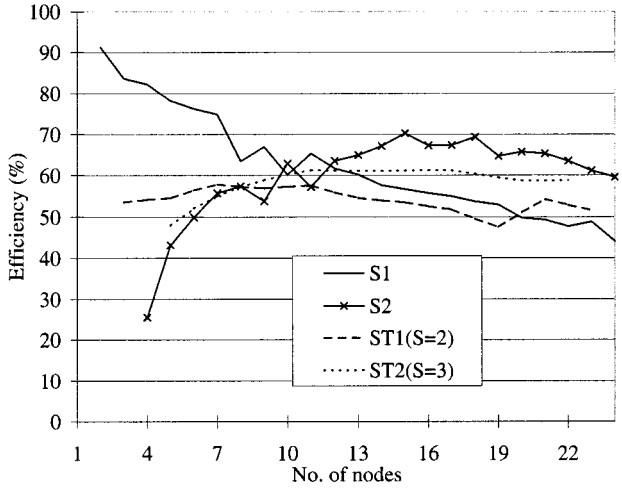


Fig. 19. Comparison of efficiency—352×240.

the others, while the others are less efficient because ST1 has the local masters, ST2 has two global masters and S2 has three masters. However, for large n, communication dominates, making it inadequate for one master to handle all the communication.

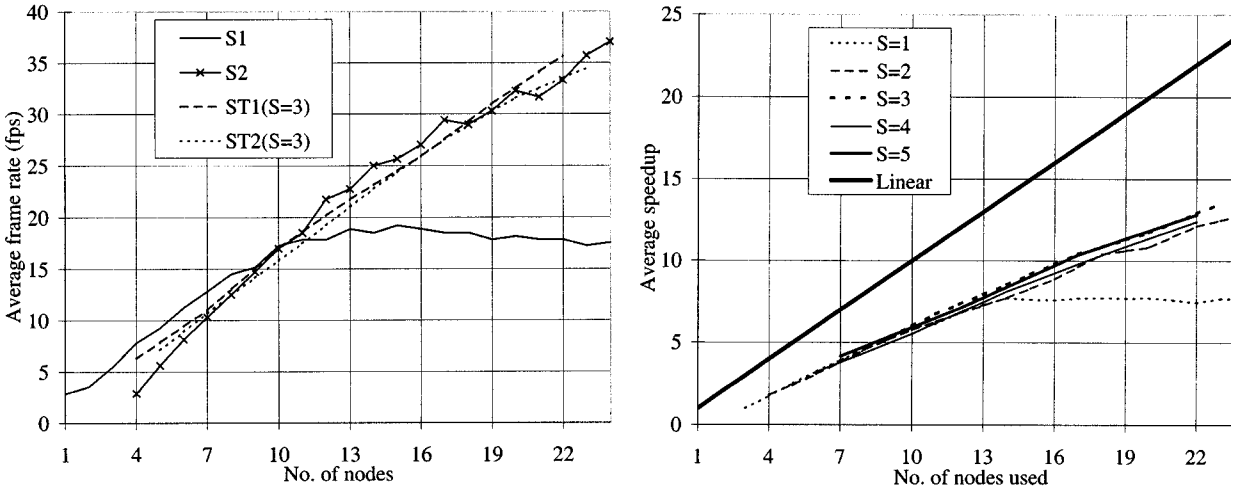


Fig. 20. (a) Frame rate comparison—QCIF video. (b) Speedup comparison—QCIF video.

In those cases, S1 and ST1 have similar performance, whereas ST2 and S2 are superior because of the multiple global masters. Between these two, the third master of S2 helps to further parallelize the communications and therefore achieves higher speedup.

If we define efficiency as $(\text{measured speedup}/n) \times 100\%$ [20], then Fig. 19 depicts the efficiencies of the four algorithms versus n. It can be seen that the S1 efficiency is between 60%–90% for $n < 10$, which is far better than the other three algorithms $(< 60\%)$ in this range. Obviously, its speedup reflects such high efficiency. For $n > 10$, the S1 efficiency decreases gradually to 44% for $n = 24$. At $n = 22$, their efficiencies are $\mathrm{S1} = 48\%$, $\mathrm{S2} = 64\%$, $\mathrm{ST1} = 53\%$ and $\mathrm{ST2} = 59\%$. On the other hand, their efficiencies peak at 70% at $n = 15$ for S2, 58% at $n = 12$ for ST1 and 61% at $n = 12$–17 for ST2. This implies that disregarding the actual frame rate achieved in each case, the algorithms are most efficient using these numbers of nodes. These figures are indeed comparable with those reported by Sijstermans *et al.* [12] (32%), Akramulla *et al.* [13] (39%) and Agi *et al.* [19] (62.5%), as the first two cases used very large n. In general, the trend of the S1 and S2 efficiencies seem to decrease as n further increases. However, the ST1 and ST2 efficiencies seem to remain between 50%–60% for large n.

For smaller problem size, Fig. 20 depicts the frame rates and speedup for the same video but being reduced to QCIF (176×144). It should also be noted that the curves representing the ST1 and ST2 algorithms are both of $S = 3$. From Fig. 20(b), it can be observed that first, for $n \leq 12$, the relative behavior of the speedup curves is similar to the 352×240 case, except with more prominent features. For large n, the differences between S2, ST1, and ST2 become less obvious, where the S1 curve levels off as in Fig. 18. Second, the S1 algorithm speedup peaks at 6.7 for $n = 15$, and gradually reduces to 6.1 for $n = 24$, which is not seen in Fig. 13(b) because of the large frame size. From this, we can expect for the 352×240 video, a peak will be reached, beyond which there will be no gain in speedup even if n continues to increase. Third, S2, ST1, and ST2 still show a steadily increasing speedup for n up to 24, where the best speedup (frame rate) achieved are 13 (37 fps), 12.5 (35.7 fps), and 12 (34.5 fps), respectively. All three algorithms achieved real time coding (30 fps) at $n = 19$. Fourth, S2, ST1, and ST2 have efficiency between 50%–60% for all n, while S1 has good efficiency for small n, but poor efficiency for large n.

VII. Conclusion

Broadly, the use of domain decomposition for parallelizing the H.261 video-coding algorithm is a viable approach as long as issues concerning data partitioning, dependency and communication are systematically dealt with. This means that the granularity should be defined, the dependency in both the spatial and temporal domains should be considered, and their communication requirement should be analyzed. From these, a performance model may be established, which can be a useful metric for gauging the actual performance of the parallel implementation. From the investigation described in this paper, a number of specific issues are identified. First, although a reasonable model can be derived, detailed knowledge in parallelization overhead, OS overhead and communication contention will certainly improve its accuracy. Second, for video coding, a mixed spatial-temporal parallelization represents a general approach. The purely temporal or spatial cases can simply be treated as special cases without losing generality. Third, centralizing communications through a single master among multiple slaves is a common approach, but it is effective only when the number of nodes is small. It becomes quite ineffective as compared with distributing communications across multiple masters, when the node count is high or communication is expensive, as in the case of workstation clusters. In our implementation, the S1 algorithm achieves the best speedup for $n < 10$, whereas the S2 algorithm is superior for $n > 10$. The same applies to the ST1 and ST2 algorithms, achieving slightly lower speedup than S2. Fourth, both the spatial-temporal algorithms scale well compared with the pure temporal or spatial cases, and the algorithms using multiple masters tend to scale a little better with more linear speedup. Concerning implementation efficiency, the S1 algorithm has

efficiency between 60%–90% for $n < 10$. For $10 < n < 24$, the S2 algorithm is more efficient. The efficiency of both the ST1 and ST2 algorithms consistently range between 50%–60%, which is fair. According to their models, these two algorithms have the potential of remaining at this efficiency level for large n, which is unlikely for S1 and S2.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Computer Center at the University of Hong Kong for their technical support and use of their IBM SP2 system.

REFERENCES

[1] *ITU-T recommendation H.261: Video codec for audiovisual services at $p \times 64$ kbits*, 1990.
[2] *ITU-T recommendation H.263: Video coding for low bitrate communication*, 1995.
[3] *MPEG-1: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s*, 1993.
[4] *MPEG-2: Generic coding of moving pictures and associated audio*, 1995.
[5] *Overview of the MPEG-4 standard*, 1997.
[6] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*. Norwell, MA: Kluwer, 1995.
[7] B. Furht, S. W. Smoliar, and H. J. Zhang, *Video and Image Processing in Multimedia Systems*. Norwell, MA: Kluwer, 1995.
[8] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard*. London, U.K.: Chapman and Hall, 1997.
[9] L. Torres and M. Kunt, *Video Coding: The Second Generation Approach*. Norwell, MA: Kluwer, 1996.
[10] C. Huang and J. L. Wu, "New Generation of real-time software-based video codec: Popular video coder II (PVC-II)," *IEEE Trans. Consumer Electron.*, vol. 42, pp. 963–973, Nov. 1996.
[11] K. Li and H. Yuen, "A high performance image compression technique for multimedia applications," *IEEE Trans. Consumer Electron.*, vol. 42, pp. 239–243, May 1996.
[12] F. Sijstermans and J. Van der Meer, "CD-I full-motion video encoding on a parallel computer," *Commun. ACM*, vol. 34, no. 4, pp. 81–91, 1991.
[13] S. M. Akramullah, I. Ahmad, and M. L. Liou, "A portable and scalable MPEG-2 video encoder on parallel and distributed computing systems," in *SPIE Proc. Visual Communications and Image Processsing*, 1996, pp. 973–984.
[14] ——, "A data-parallel approach for real-time MPEG-2 video encoding," *J. Parall. Distrib. Comput.*, vol. 20, no. 2, pp. 129–146, Nov. 1, 1995.
[15] K. Shen, L. A. Rowe, and E. J. Delp, "A parallel implementation of an MPEG1 encoder: Faster than real-time!," in *Proc. SPIE Conf. Digital Video Compression: Algorithms and Technologies*, San Jose, CA, Feb. 5–10, 1995, pp. 407–418.
[16] K. Shen and E. J. Delp, "A spatial-temporal parallel approach for real-time MPEG video compression," in *Proc. 25th Int. Conf. Parallel Processing*, Bloomingdale, IL, Aug. 13–15, 1996, pp. II100–II107.
[17] T. Akiyama *et al.*, "MPEG-2 video codec using image compression DSP," *IEEE Trans. Consumer Electron.*, vol. 40, no. 3, pp. 466–472, 1994.
[18] C. Bouville *et al.*, "DVFLEX: A flexible MPEG real time video codec," in *Proc. IEEE Int. Conf. Image Processing*, vol. II, 1996, pp. 829–832. ICIP'96.
[19] I. Agi and R. Jagannathan, "A portable fault-tolerant parallel software MPEG-1 encoder," in *Multimedia Tools and Applications*, 1996, vol. 2, pp. 183–197.
[20] K. Hwang and Z. Xu, "Scalable parallel computers for real-time signal processing," *IEEE Signal Processing Mag.*, pp. 50–66, July 1996.
[21] P. Chalermwat *et al.*, "Parallel image processing in heterogeneous computing network systems," in *Proc. IEEE Int. Conf. Image Processing*, vol. II, 1996, pp. 161–164. ICIP'96.
[22] Y. Sorel, "Real-time embedded image processing applications using the A^3 methodology," in *Proc. IEEE Int. Conf. Image Processing*, vol. II, 1996, pp. 145–148. ICIP'96.
[23] S. M. Bhandarkar and H. R. Arabnia, "Parallel computer vision on a reconfigurable multiprocessor network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, pp. 292–309, Mar. 1997.
[24] T. Agerwala *et al.*, "SP2 system architecture," *IBM Syst. J.*, vol. 34, no. 2, pp. 152–184, 1995.
[25] C. B. Stunkel *et al.*, "The SP2 high-performance switch," *IBM Syst. J.*, vol. 34, no. 2, pp. 185–204, 1995.
[26] M. Snir, P. Hochschild, D. D. Frye, and K. J. Gildea, "The communication software and parallel environment of the IBM SP2," *IBM Syst. J.*, vol. 34, no. 2, pp. 205–221, 1995.
[27] (1994) SP Parallel Programming Workshop—LoadLeveler. Maui High Performance Computing Center, Maui, HA. [Online]. Available: http://www.mhpcc.edu/training/workshop/index.html
[28] A. C. Hung, "PVRG-P64 Codec 1.1," Portable Video Research Group (PVRG), Stanford Univ., Stanford, CA, 1993.
[29] K. Hwang, Z. Xu, and M. Arakawa, "Benchmark evaluation of the IBM SP2 for parallel signal processing," *IEEE Trans. Parallel Distrib. Processing*, vol. 7, pp. 522–536, May 1996.
[30] I. Foster, "Designing and building parallel programs—Concepts and tools for parallel software engineering," . Reading, MA: Addison-Wesley, 1995.
[31] K. K. Leung, "Spatial and Temporal Data Parallelization of the H.261 Video Codec on the IBM SP2 Multiprocessor System," M.Sc. thesis, Dept. Elec. Electron. Eng., Univ. Hong Kong, Hong Kong, 1997.
[32] N. H. C. Yung and K. C. Chu, "Fast and parallel video encoding by workload balancing," in *Proc. IEEE SMC'98*, Oct. 1998, pp. 4642–4647.

Nelson H. C. Yung (SM'96) received the B.Sc. and Ph.D. degrees from the University of Newcastle-Upon-Tyne, Newcastle-Upon-Tyne, U.K., in 1982 and 1985, respectively.

He was a Lecturer at the University of Newcastle-Upon-Tyne from 1985 until 1990, where he was involved in the research and development (R&D) of digital image processing and parallel processing. From 1990 to 1993, he was a Senior Research Scientist at the Department of Defence, Australia, where he headed a team on the R&D of military-grade signal analysis systems. He joined the University of Hong Kong in late 1993 as an Associate Professor, where he leads a research group in Digital Image Processing and Intelligent Transportation Systems. He is the founding Director of the Laboratory for Intelligent Transportation Systems Research, and has published over 100 research papers. His biography is published in *Marquis' Who's Who in the World*.

Dr. Yung serves as Reviewer for the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B, IEEE TRANSACTIONS ON SIGNAL PROCESSING, *IEE SPIE Optical Engineering—Part G*, *HKIE Proceedings*, and the *Microprocessors and Microsystems Journal*. He is a Chartered Electrical Engineer and Member of the HKIE and IEE.

Kwong-Keung Leung (S'99) received the B.Sc. and M.Sc. (with distinction) degrees from the Department of Electrical and Electronic Engineering, University of Hong Kong, in 1990 and 1997, respectively, where he is currently working toward the Ph.D. degree.

His research interests include multiprocessor scheduling, dynamic load balancing, heterogeneous computing, and parallelization of multimedia systems.