Exploiting Spatial Parallelism For Software-only Video Effects Processing

Ketan Mayer-Patel Lawrence A. Rowe
{kpatel,rowe}@cs.berkeley.edu
Computer Science Division, EECS
University of California, Berkeley
Berkeley, CA 94720

Abstract

Video effects play an important role in adding production value to video programs. The use of video effects with Internet Video sources, however, is still uncommon because traditional hardware-based solutions are poorly suited to the Internet environment. In previous work, we described a parallel, software-only video effects system designed for Internet Video and explored the use of temporal parallelism. This paper explores the use of spatial parallelism. In particular, an intermediate semicompressed video format is desribed that was designed to exploit spatial parallelism, and performance measurements are reported on the use of this representation.

1 Introduction

Experience from the television, video, and film industries shows that visual effects are an important tool for communicating and maintaining audience interest [11]. Titling, for example, is used to identify speakers and topics in a video presentation. Compositing effects that combine two or more video images into one image are used to present simultaneous views of people or events at different locations. Blends, fades, and wipes are transition effects that ease viewers from one video source to another.

The use of video effects with Internet Video (IV) sources is uncommon. Live Internet broadcasts of conferences, classes, and other special events require improved production values that video effects provide.

Traditionally, video effects are created using a video production switcher (VPS). A VPS is a specialized hardware device that manipulates analog or digital video signals to create video effects. It is usually operated by a technician or director at a control console. Unfortunately, traditional video effects processing hardware is poorly matched to the IV environment. IV sources are characterized by variable frame rates, bit rates, and jitter. Traditional hardware, whether analog or digital, depends on constant frame rates, constant bit rates, and tightly synchronized signalling.

We are developing a software-only video effects processing system designed for the IV environment. A software-only solution using commodity hardware provides the flexibility required to handle IV. Variable frame rates, packet loss, and jitter can be dealt with gracefully with dynamic adaptation. A software system can be written to handle IV formats already in use (e.g., JPEG, H.263, etc) and extended to handle new formats. And, standard IV multicast communication protocols (e.g., RTP [14]) can be used. Using general-purpose processors allows a software system to benefit from continuous improvements in processor and networking technology.

The key to a software-only solution is to exploit parallelism because the computation required for many effects (e.g., chroma-key, complex transformations, etc.) is beyond the capability of a single processor. Even as processors become faster, the demand for more complicated effects, larger images, and higher quality will increase. Improvements in processor and networking technology will only be met with greater application demands.

Fortunately, video processing contains a high degree of parallelism. Three types of parallelism can be exploited for effects processing: functional, temporal, and spatial. Functional parallelism decomposes the video effect task into smaller subtasks and maps them onto the available computational resources. Temporal parallelism can be exploited by demultiplexing the stream of video frames to different processors and multiplexing the processed output. Spatial parallelism can be exploited by assigning regions of the video stream to different processors.

In previous work, we outlined the design and architecture of a software-only parallel video processing system and explored issues related to exploiting temporal parallelism [8]. This paper explores issues related to spatial parallelism. Specifically, an intermediate semicompressed video format is described that was designed to exploit spatial parallelism. Performance measurements are reported on the use of this representation.

The paper is organized as follows. Section 2 reviews related work. The overall system architecture and design is described in Section 3. Issues related to exploiting spatial parallelism are discussed in Section 4. The intermediate semicompressed video format is described in Section 5. Performance measurements are reported in Section 6. Finally, Section 7 summarizes the paper.

2 Related Work

Several hardware systems have been developed to explore parallel video effects processing. The Cheops system developed by Bove and Watlington at MIT is composed of interconnected special-purpose hardware components that implement specific functions (e.g., discrete cosine transform (DCT), convolution, etc.) [1]. The IBM Power Visualization System is a parallel processor composed of up to 32 identical processors interconnected by a global bus [4]. It was designed specifically to support the IBM EFX suite of editing, effects, and compression software. The Princeton Engine is a parallel processor composed of up to 2048 custom-designed processing elements [2]. Many other hardware systems have also been developed [6, 7, 13]. The system proposed here differs fundamentally from these systems by not assuming any particular underlying parallel architecture and concentrating on a software-only solution.

More recent work by Bove and Watlington describes a general system for abstractly describing media streams and processing algorithms that can be mapped to a set of networked hardware resources [16]. In this system, hardware resources may be special-purpose media processors or general-purpose processors. The system is centered around an abstraction for media streams that describes any multidimensional array of data elements. The system achieves parallelism by discovering overlaps in access patterns and scheduling subtasks and data movement among processors to exploit them. The system uses a general approach that is not specific to video or packet video formats and that is independent of networking protocols. This research shares some of the same goals and solutions that we are working toward. Our system is different in that we are taking advantage of representational structure present in compressed video formats, and we are constrained to standard streaming protocols for video on the Internet (i.e., RTP).

Dali is a low-level set of image operators that operate on a specific representation of data elements [15]. We are using Dali as the language to express primitive effects processing tasks. The semicompressed memory representation available in Dali was a starting point for developing the intermediate video format used in exploiting spatial parallelism in our system.

Our system is built on top of the MASH toolkit. MASH is a flexible software environment for building distributed continuous media applications [10]. It supports existing Internet protocols including RTP, RTCP, RTSP and SRM [14, 5, 12].

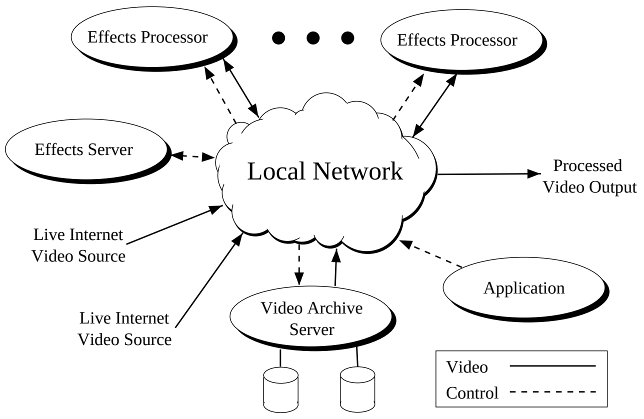

Figure 1: System Architecture

3 System Architecture

This section briefly describes the design and architecture of the system we are building. A more detailed description of this design is given in a previously published paper [8].

The overall system architecture is shown in Figure 1. The oval labeled "Application" represents an application that requires video effects processing. The application sends a specification of the desired video effect to the "Effects Server." The ovals labeled "Effects Processor" represent general-purpose computers. The "Effects Server" allocates system resources, maps effects onto available processors, and provides the application with a means to control the effect (i.e., change any relevant parameters).

The system is composed of three major software components: the FX Compiler, the FX Mapper, and the FX Processor. The relationship between these components is illustrated in Figure 2. Placing these components in the overall system architecture depicted in Figure 1, the FX Compiler and FX Mapper are part of the "Effects Server" and the FX Processor is the software executing on an "Effects Processor."

The FX Compiler translates a high-level description of a video effect into an intermediate representation. Our target representation is a directed graph of video operators (i.e., a data flow graph). Figure 3 shows a cross-dissolve video effect expressed as di-

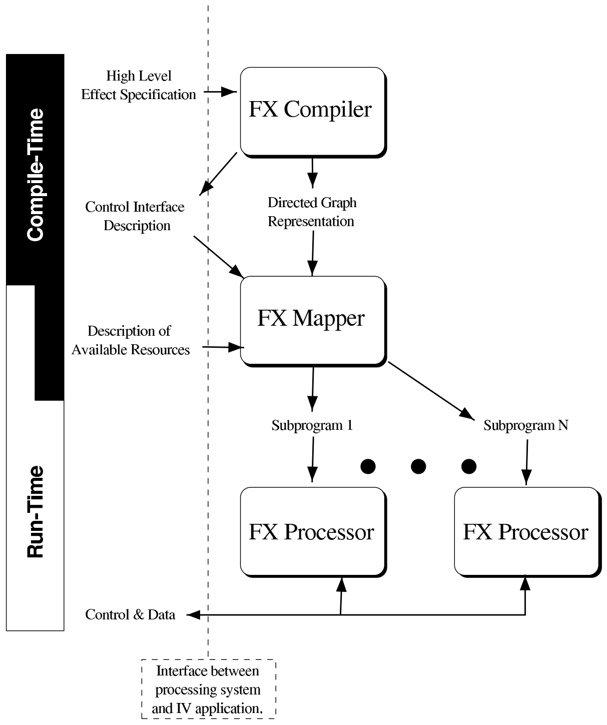
Compile-Time
Run-Time
High Level Effect Specification
FX Compiler
Control Interface Description
Directed Graph Representation
FX Mapper
Description of Available Resources
Subprogram 1
Subprogram N
FX Processor
FX Processor
Control & Data
Interface between processing system and IV application.

Figure 2: Software Architecture

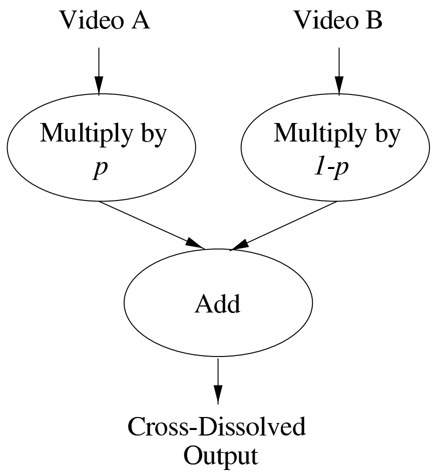

Figure 3: Cross-Dissolve Directed Graph Representation

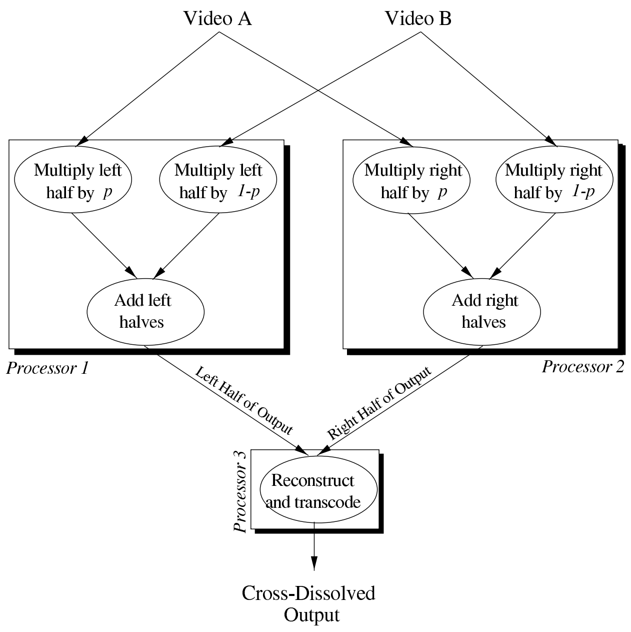

Figure 4: Spatial Partition of Cross-Dissolve Graph

rected graph of primitive operators.

The FX Mapper takes the intermediate representation and maps it onto the available resources. It produces effect "subprograms" that will be executed on a particular computational resource. We express these subprograms as Dali scripts. Again consider the cross dissolve example illustrated in Figure 3. Figure 4 shows a possible partitioning using spatial parallelism. Each input frame is sent to both processors. But a processor only operates on a subregion which in this case is the left or right half of the frame. The graph is augmented with an operator to reconstruct output frames from the partial results (i.e., left and right halves).

The FX Processor instantiates the effect subprogram, opens the appropriate input sockets, executes the subprogram when presented with data, and responds to control signals sent from the application.

Our system relies only on a set of general-purpose processors connected by a local area network. It does not rely on any specialized parallel architecture. If, however, processors are tightly coupled with either shared memory or a high-speed low-latency interconnect, optimizations can be made to improve performance.

4 Spatial Parallelism

There a two basic problems that must be solved to exploit spatial parallelism. First, each FX Processor must acquire the appropriate input data to compute a share of the solution. Second, the results of all FX Processors must be recombined to form a single output frame. Control over which processor will produce which subregion and how the subregions are calculated are separate issues not discussed in this paper. The following subsections outline the issues that must be resolved when dealing with these two problems and our solution.

4.1 Input Distribution

The problem of distributing the input video is heavily dependent on the computation being done and the format of the input video sources. For some effects, computation of an output subregion requires access to the directly corresponding subregions of the inputs. For example, when performing a cross-dissolve (i.e. fade) between two input video sources, any subregion of the output only requires the corresponding subregions of the two inputs. Other effects, however, have more complex relationships between inputs and outputs. An affine transformation (i.e., scale, rotation, translation, etc.) may require any portion of the input video frame to compute a particular subregion of the output. The relationship between inputs and outputs in these kinds of effects are generally captured by user-specified parameters (e.g., angle of rotation, center of rotation, or scaling factor). These parameters may also vary in time (e.g., fading from one stream to another). Since many effects are the result of applying two or more transformations, tracking the inverse relationship between output subregion and required input regions can become complex.

Even if the required region of the input video is easily computed, the input video format may not lend itself to a simple extraction of that region. Motion-

JPEG (M-JPEG) video streams, for example, code only the difference between the DC coefficients of successive 8x8 DCT blocks. Thus, the value of any given block's DC coefficient depends on the value of the DC coefficients of all blocks that came before it. MPEG video streams use motion vectors that may require the value of pixels outside the region of interest. If the region of interest changes, formats that use conditional block replenishment such as the widely used H.261 video format may require decoding a block several frames in the past which was not in the region of interest to determine the pixel values of a particular block in the present which now is in the region of interest [9].

One possible approach is to translate the input video into an intermediate format that facilitates extracting only the necessary regions. Once transcoded, this intermediate video stream can be multicast to the processors involved in computing the video effect. Each processor extracts only the required information. We rejected this approach for several reasons. First, it adds more latency as each input stream must be transcoded and retransmitted. Second, we still have the problem of maintaining the inverse relationship between output subregion and the necessary input regions.

Our approach to this problem is to have each processor receive and decode each of the input video streams in its entirety. This approach keeps the input video decoding process as simple as possible at the expense of decoding unnecessary regions of the input video streams and replicating decoding effort at each of the participating processors. We believe these costs are outweighed by the advantage of not having to maintain the inverse relationship between outputs and inputs. In addition, the problems of time varying regions of interest in the presence of conditional replenishment schemes and motion vectors are sidestepped. One optimization that might be made if processors are tightly coupled is to share decoding results (e.g., maintain a shared cache of decoded blocks).

4.2 Output Reconstruction

The problem of reconstructing the output from results produced by each FX Processor requires communicating these subregions to a central location. The format of these intermediate results heavily impacts the amount of work necessary to reconstruct the output frame.

Traditional IV formats are badly suited to this task. Each subregion must specify its own geometry (i.e., width and height) as well as its relationship to the larger geometry of the whole frame (i.e., vertical and horizontal offset). M-JPEG, H.263, H.261, and MPEG do not have mechanisms for communicating this information. Extending RTP payload formats for these video formats to include this information is possible, but it makes the new payload syntax incompatible with existing software.

Our approach to this problem is to design our own intermediate format specifically for the task of reconstructing whole frames from subregions. This new intermediate format is used only to communicate subregion results from the participating processors to one particular processor that will reconstruct the frame and transcode it into the target output format. RTP is used as the transport protocol throughout our system so we have defined a new payload type for this intermediate format. Figure 4 illustrates the relationship between different processes involved in exploiting spatial parallelism in our cross-dissolve example. Note that multicast is used to communicate the input video streams to all processors involved and that the intermediate subregions are transmitted to the reconstructing process in our newly developed format.

5 Semicompressed Video Format

This section discusses the issues encountered when designing the intermediate format, justifies our design decisions, and provides details about the format developed.

Region Dimensions	Size (kB)	Bitrate @ 30 fps (Mb/s)
8x8	0.1	0.02
80x60	7.2	1.70
160x120	28.8	6.90
320x240	115.2	27.60
640x480	460.8	110.60
1280x720	1400.0	331.80

Table 1: Uncompressed Video Sizes and Bitrates

5.1 Design Issues

Three main issues were encountered when designing the intermediate format:

1. Should compression be used? If so, how much and in what form?
2. How will subregions be packetized?
3. What subregion geometries will be allowed? How will they be specified?

The first question concerns compression. Because the intermediate format is not the output target format, any compression used at this stage may have to be undone at the final transcoding stage. Unfortunately, raw formats are large. Table 5.1 shows the size of a single frame of uncompressed video for various region sizes. Each frame is represented by an 8-bit luminance plane and two subsampled chrominance planes (i.e., two chrominance pixels for every four luminance pixels). Also shown are the corresponding bit rates required at 30 frames per second.

Exploiting spatial parallelism will make the most sense as frame sizes grow and the portion of computation that can be effectively parallelized is large. Using an uncompressed intermediate format limits the applicability of spatial parallelism by quickly exhausting network resources. The problem is exacerbated by the fact that all intermediate results have to go to one place for reconstruction and transcoding.

Our approach is to use a simple DCT block-based compression scheme. There are several advantages to this approach. First, DCT is at the base of many widely used compression schemes (i.e., M-JPEG, H.263, and MPEG). Thus, in the final transcoding stage, the DCT coefficients can be used directly. Second, if the intermediate results must be rate limited due to network resource constraints, a DCT-based solution provides a convenient representation to throw away data that is perceptually less significant (i.e., high frequency coefficients).

The second design issue is how to deal with packetization. Even though the intermediate format will be used to describe subregions of a larger frame, we may have to use more than one RTP packet to transmit the data for the subregion. The application level framing principle upon which RTP is built mandates that each RTP packet be processed independent of any other packet [3]. This principle led to two design decisions. First, the entire subframe geometry and its relationship to the larger geometry of the original frame must be specified in each and every packet. Second, the coding granularity must be small enough to fill packets efficiently. The decision to use a DCT based compression scheme led us to choose 8x8 blocks of pixels as the smallest unit of coding.

The third design issue concerns subregion geometries. Decisions made when dealing with the previous two issues constrain our options. The use of DCT block-based compression implies that subregion geometries must be rectilinear and in multiples of the base 8x8 block size. The necessity of describing full geometry information in every packet makes hierarchical geometry cumbersome. We decided to allow only one level of subgeometry. The system may decompose a frame several times using spatial decomposition. But if a subregion is further subdivided, each piece retains only its own geometry and the original frame geometry. In other words, the intermediate subregion geometry is lost.

5.2 Format Details

This section describes the intermediate format we developed in detail and points out key features that facilitate the reconstruction of whole frames from subregions. We will refer to our new format as the semi-compressed format (SC).

The following assumptions are made about the subregion video data coded into the SC format:

1. The video data consists of three planes: a luminance plane (Y) and two chrominance planes (Cr, Cb).
2. The width and height of the Y plane are multiples of 8.
3. The Cr and Cb planes are possibly subsampled.
4. If the Cr and Cb planes are subsampled, the subsampled width and height are still multiples of 8 and both planes are subsampled to the same degree.

The SC format allows for the description of any subregion which is rectilinear along 8x8 block boundaries. The last assumption stated above may restrict the subregion geometry to courser block boundaries since the subregion dimensions must be multiples of the subsampling factor as well. For example, if the chrominance planes are subsampled by a factor of 2 horizontally and not subsampled at all vertically, subregion widths must be multiples of 16 while subregion heights may be multiples of 8.

A subregion is described by one or more SC packets. Figure 5 shows the components of each SC packet. Each packet is composed of an RTP header followed by an SC header and the description of one or more 8x8 blocks of pixels as DCT coefficients. Figures 6 and 7 show further details of the RTP and SC headers. In the RTP header, the *flags* field is 8 bits wide, the *frame marker bit* is a single bit, the *type* field is 7 bits wide, and the *sequence number* is 16 bits wide.

The *marker bit* in the RTP header is set when the SC packet is the last packet of a sequence of packets describing the contents of the subregion for a particular timestamp. SC packet decoders should not depend on this bit being set because the last packet may be lost. The decoder should be able to detect when all available SC packets for a subregion have been received from the change in timestamp values in the RTP header. The *type* field in the RTP header identifies the rest of the packet as being in the SC format.

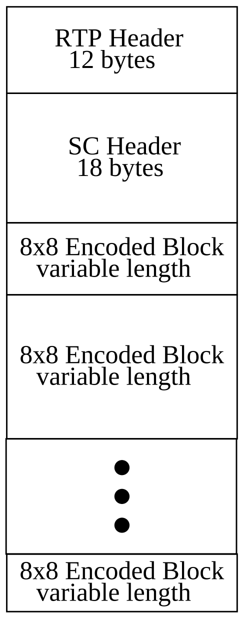

Figure 5: SC Packet Format

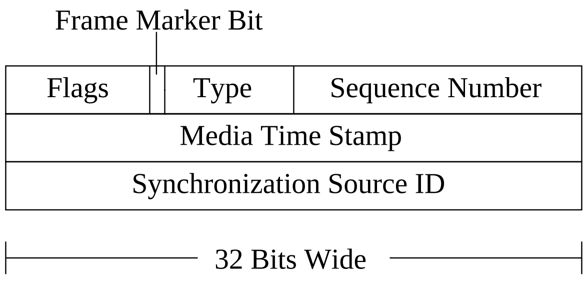

Figure 6: RTP Header

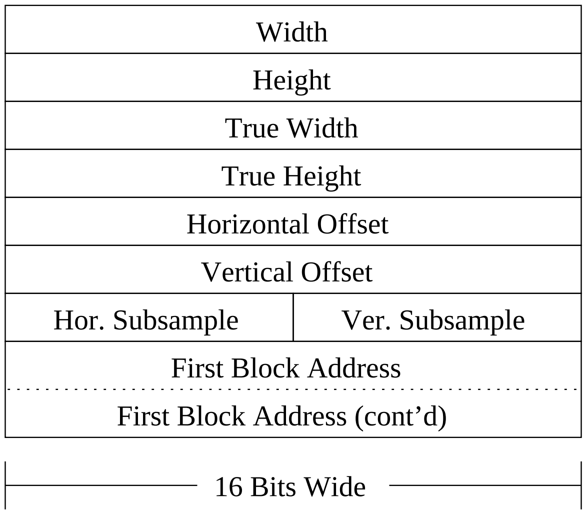

Figure 7: SC Header

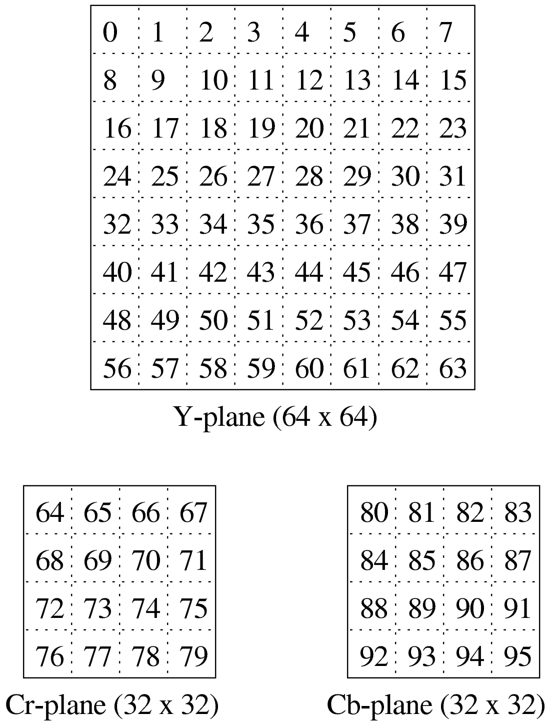

Figure 8: Block Addressing

The SC header consists of 9 fields:

width The width of the subregion in pixels.

height The height of the subregion in pixels.

true width The width of the original frame in pixels.

true height The height of the original frame in pixels.

horizontal offset The horizontal offset of the subregion in pixels.

vertical offset The vertical offset of the subregion in pixels.

horizontal subsample The horizontal subsampling factor of the chrominance planes.

vertical subsample The vertical subsampling factor of the chrominance planes.

first block address The address of the first block of pixels described in this packet.

Each 8x8 block of pixels is given a "block address" that uniquely determines both the position and plane of the pixel values. The address is calculated relative to the original parent geometry by enumerating the 8x8 blocks in row-order starting with the upper left block of the Y-plane and continuing with the Cr- and Cb-planes, respectively. Figure 8 shows this addressing scheme for a frame 64 pixels wide and 64 pixels tall with 4:1:1 chrominance subsampling.

The format of a block encoding is shown in Figure 9. Each block is made up of a DC coefficient, zero or more AC coefficients, and a block address increment. The coefficients have 12 bit precision and are unscaled. The DC coefficient is coded with 16 bits. Each AC coefficient is coded with either 16 or 32 bits depending on the number of zero coefficients that precede it. The block address increment determines the address of the next block described in this packet. It is coded in either 16 or 32 bits depending on its value.

The AC coefficients are encoded in row-major order

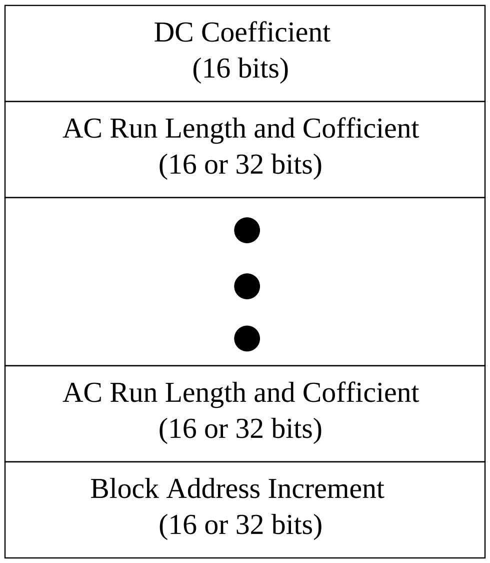

Figure 9: Block Encoding

using run length encoding to avoid coding coefficients with a value of zero. Each AC coefficient is encoded along with the number of zero coefficients that precede it (i.e., the run length). If the run length is less than 15 (the most common case), 16 bits are used. The top 4 bits encode the run length and the next 12 bits encode the coefficient value. If the run length is greater than 15, the run length is encoded in 16 bits with the top 6 bits set to an escape code and the next 10 bits encoding the run length. Following the escaped run length are 16 bits indicating the coefficient value. When the run length is greater than 15, many bits are not efficiently used. We expect this case to be rare.

The block address increment indicates that no more AC coefficients are encoded for the current block and determines the block address of the next block described in this packet by encoding the difference between the block address of the next block and the current block address. If this difference is less than 1023, the block address increment can be encoded into 16 bits. If greater than 1023, the block address increment is encoded into 32 bits. In either case, the first 6 bits are set to one of 2 escape codes indicating the end of the AC coefficients for the current block and determining the number of bits (10 or 26) used for the block address increment.

The description of a subregion need not include descriptions of every block in the subregion. The order of the blocks is also not strictly specified.

The SC format has two key features. First, if network constraints demand that each processor rate limit the resulting SC packet stream, several techniques are easily applicable. Small coefficients that may be quantized to zero by the eventual transcoding process can be discarded. High frequency AC coefficients can be discarded to reduce the number of coefficients encoded in each block. And, conditional replenishment can be applied by not coding blocks that have not changed since last being transmitted.

Second, reconstruction of the original frame can be separated into two stages: transforming SC packet streams describing subregions into a single SC packet stream that describes the larger parent geometry and transcoding this packet stream into the desired output format. Since the block addressing scheme used is relative to the original frame geometry and is independent of the subregion geometry, packets from several different sources describing different subregions can be easily transformed into what appears to be a single stream of SC packets that describe the larger, original geometry. This new stream of SC packets can then be sent to a transcoder that produces the desired output frame. This transformation simply changes the *width* and *height* fields of each SC packet to be equal to the *true width* and *true height* fields. Additionally, the RTP *sequence number* and *source id* fields are changed to interleave the separate streams into one valid stream of SC packets.

6 Measurements

This section reports measurements relating the size of the SC format relative to M-JPEG and the performance of the SC encoder and decoder. The SC format size is of interest because trading off compression with ease of reconstruction is one of the reasons we developed a new intermediate format. The perfor-

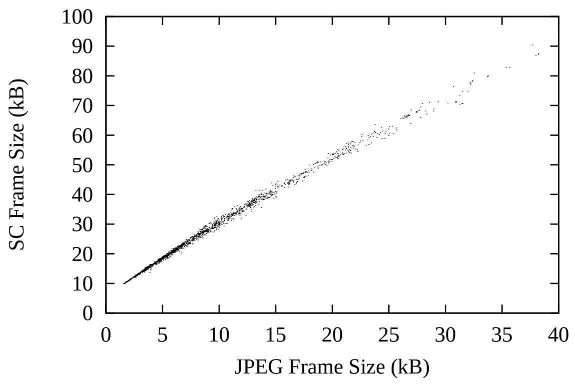

Figure 10: SC frame size vs. M-JPEG frame size.

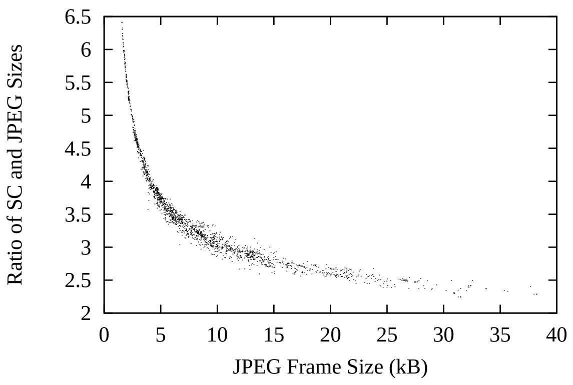

Figure 11: Ratio of SC frame size to M-JPEG frame size vs. M-JPEG frame size.

mance of the SC encoders and decoders is of interest because the cost of getting into and out of our intermediate format will be integral to predicting the cost of using spatial parallelism. Finally, end-to-end comparisons of video effects implemented using spatial parallelism and effects implemented using temporal parallelism are given.

6.1 SC Format Size

Figure 10 shows a scatter plot of the size of over 2000 frames in our SC format relative to the size of the original M-JPEG frames. In these measurements the frame dimensions were 320x240. Larger M-JPEG frame sizes correlate to higher quality and more high frequency information. In other words, more DCT coefficients per block. No information was lost when reencoding the frame into our format (i.e., all coefficients were coded). We can see that SC format frame sizes range from 10kB to 70kB depending on the quality of the original M-JPEG frame. It appears that the SC format size is a linear scale of the M-JPEG size, but Figure 11 disproves this hypothesis.

Figure 11 shows the same data but now the ratio of the SC frame size and the M-JPEG frame size is plotted against the original M-JPEG frame size. We can see that when M-JPEG sizes are small (around 3kB), the corresponding SC format is around 6 times larger (around 18 kB), but as M-JPEG frame sizes grow, the corresponding SC format approaches being only twice as large. The decrease in the ratio happens because as the number of DCT coefficients per block increases, the effectiveness of M-JPEG entropy coding decreases.

6.2 Encoder/Decoder Performance

To measure SC codec performance, we performed two experiments. Both were conducted using an UltraSparc1 workstation. In the first, encoding time was measured as the size of the region encoded was varied. This encoding time corresponds to the work being done by individual FX Processors as they encode subregions to be sent for reconstruction and transcoding. Figure 12 shows the time spent encoding a single block as the total number of blocks (i.e., the subregion size) increases. We can see that the encoding time is constant on a per block basis. This result shows that we can predict the required encoding time for a given spatial subdivision. This property will be important when building the FX Mapper.

In the second experiment, the time necessary to transcode an SC frame into an M-JPEG frame was measured. This time corresponds to the final reconstruction and transcoding of the subregions into a

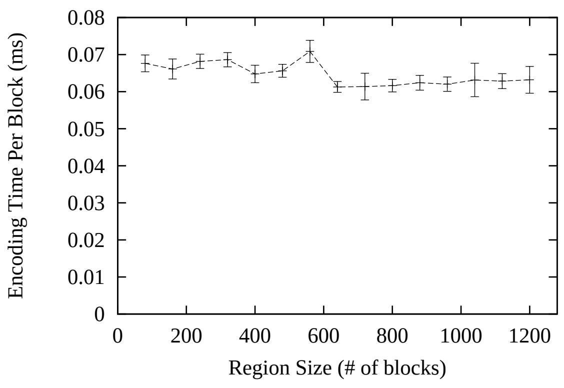

Figure 12: Encoding time of a single block in milliseconds vs. the total number of blocks encoded (i.e., subregion size).

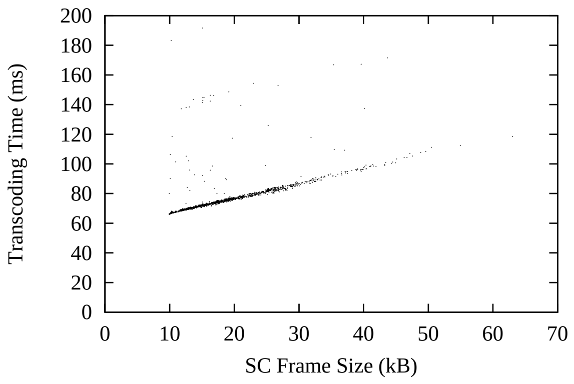

Figure 13: Reconstruction and transcoding time of an SC frame into M-JPEG vs. the size of the SC frame.

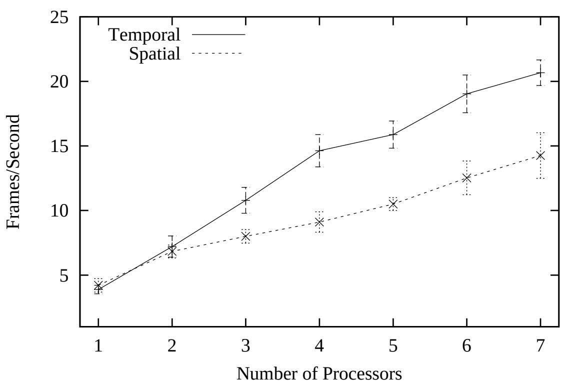

Figure 14: Performanace in frames per second for temporal and spatial parallelism.

single frame of the desired output format. Figure 13 shows the time required to reconstruct and transcode SC frames into M-JPEG frames as a function of the original SC frame size. In this experiment, the frame dimensions were 320x240. Clearly, the transcoding time is related to the size of the SC frame (i.e., the number of coefficients per block). This result indicates that rate limiting the size of the subregions can be used to reduce the cost of the final transcoding. Unfortunately, the cost of transcoding appears to be too high on this particular processor (i.e., UltraSparc1) to achieve full motion real-time frame rates on a single processor even for small SC frame sizes. To achieve real-time frame rates using spatial parallelism, we will have to increase the performance of the final transcoding step either through code optimizations or by applying temporal parallelism (i.e., using more than one processor for the transcoding).

6.3 Comparison With Temporal Parallelism

To compare spatial parallelism to temporal parallelism we measured the end-to-end performance of both approaches on a particular video effect. In our experiment, the effect was a general affine transformation (i.e., any combination of scaling, rotation,

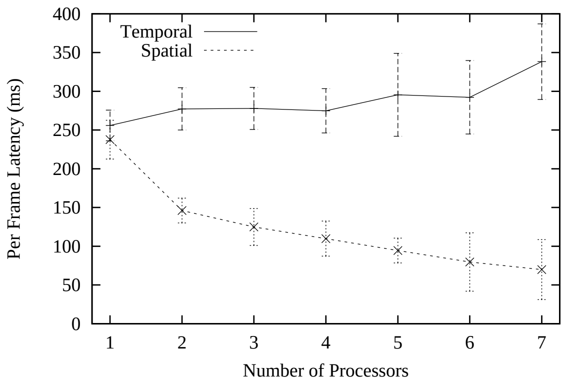

Figure 15: Per frame processing latency for temporal and spatial parallelism.

and translation). The input video format was M-JPEG and the output video format was our newly developed SC format. In an actual application, the output video would have to be further transcoded into a more common and compact representation like H.261. We used Ultra-Sparc1 workstations connected by a 10Mb/s switched Ethernet for the experiment.

Figure 14 shows the results of the experiment. The temporal mechanisms are able to scale to larger number of processors much more effectively than the spatial mechanisms. A direct consequence of the design decisions we made when dealing with the problem of input distribution is that each processor involved in exploiting spatial parallelism must spend some amount of time processing input video frames. This cost establishes an upper bound on spatial parallelism performance because it is incurred on a per-frame basis and is not related to the size of the output region.

The advantage of using spatial parallelism is lower processing latency. The time required for an input frame to be processed using temporal parallelism is relatively constant regardless of the number of processors involved. The temporal mechanisms achieve higher frame rates by overlapping the processing costs of different frames. The spatial mechanisms, however, achieve higher frame rates by reducing the per frame latency. Figure 15 shows the per frame processing latency in milliseconds for our experiment. The processing latency experienced by frames using temporal parallelism with 7 processors was about 340 milliseconds and only 70 milliseconds using spatial parallelism.

7 Summary

This paper explored issues related to exploiting spatial parallelism for video effects processing. In particular, an intermediate video format designed for exploiting spatial parallelism was described and performance measurments were reported. Key features of the intermediate format were the ability to specify subgeometry and a block addressing scheme that optimized reconstruction. Frames encoded in the new format are 2 to 6 times larger than the corresponding M-JPEG frames. The size penalty of the intermediate format decreases as video quality increases. Transcoding from the intermediate format to M-JPEG is related to the size of the intermediate format encoding and is not currently within real-time speeds for low-end single processors.

We plan on exploring the use of spatial parallelism further by constructing feedback mechanisms to dynamically adapt subregion assignments among processors.

Acknowledgments

This work was supported by National Science Foundation grant #CDA-9512332 and equipment donated by Intel Corporation.

References

[1] V. M. Bove, Jr. and J. A. Watlington. Cheops: A reconfigurable data-flow system for video processing. *IEEE Transactions on Circuits and Systems for Video Processing*, 5(2):140–149, April 1995.

[2] D. Chin, J. Passe, F. Bernard, H. Taylor, and S. Knight. The Princeton Engine: A real-time video system simulator. *IEEE Transactions on Consumer Electronics*, 32(2):285–297, 1988.

[3] D.D. Clark and D.L. Tennenhouse. Architectural considerations for a new generation of protocols. *Proceedings of ACM SIGCOMM '90 Symposium*, 20(4):200–208, 1990.

[4] D. A. Epstein et al. The IBM POWER Visualization System: A digital post-production suite in a box. *136th SMPTE Technical Conference*, pages 136–198, 1994.

[5] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, December 1997.

[6] T. Ikedo. A scalable high-performance graphics processor: GVIP. *Visual Computer*, 11(3):121–33, 1995.

[7] R. M. Lougheed and D. L. McCubbrey. The cytocomputer: a practical pipelined image processor. *Conference Proceedings of the 7th Annual Symposium on Computer Architecture*, pages 271–278, 1980.

[8] Ketan Mayer-Patel and Lawrence A. Rowe. *Exploiting Temporal Parallelism for Software-Only Video Effects Processing.* To appear in Proceedings of ACM Multimedia 1998.

[9] Steven McCanne. *Scalable Compression and Transmission of Internet Multicast Video.* PhD thesis, University of California Berkeley, December 1996.

[10] Steven McCanne et al. Toward a common infrastructure for multimedia-networking middleware. *Proceedings of the 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 1997.

[11] G. Millerson. *The Technique of Television Production.* Focal Press, Oxford, England, 1990.

[12] Anup Rao and Rob Lanphier. *RTSP: Real Time Streaming Protocol*, February 1998. Internet Proposed Standard, work in progress.

[13] Shigeru Sasaki, Tatsuya Satoh, and Masumi Yoshida. IDATEN: Reconfigurable video-rate image processing system. *FUJITSU Sci. Tech. Journal*, 23(4):391–400, December 1987.

[14] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. *RFC 1889, RTP: A Transport Protocol for Real-Time Applications*, January 1996.

[15] B. C. Smith. *Dali: High-Performance Video Processing Primitives.* Cornell University. Unpublished work in progress.

[16] J.A. Watlington and V.M. Bove, Jr. A system for parallel media processing. *Parallel Computing*, 23(12):1793–1809, December 1997.