A Multicast Control Scheme For Parallel Software-only Video Effects Processing

Ketan Mayer-Patel Lawrence A. Rowe
Berkeley Multimedia Research Center
University of California, Berkeley
{kpatel,rowe}@cs.berkeley.edu

Abstract

We have developed a parallel software-only processing system for creating real-time video effects such as titling and compositing (e.g., picture-in-picture) using compressed Internet video sources. The system organizes processors into a hierarchy of levels. Processes at each level of the hierarchy can exploit different types of parallelism and coordinate the actions of lower levels. To control the effect, control messages must be distributed to processors in the hierarchy while preserving the independence of each level. This requires a control mechanism that supports efficient delivery of messages to groups of processors, tunable reliability semantics, and recoverable state information. We describe a mechanism that meets these requirements that uses IP-Multicast, the Scalable Reliable Multicast protocol, and the Scalable Naming and Announcement Protocol. We also describe an optimization that provides a flexible framework for linking the control of different aspects of one or more related video effects.

1 Introduction

Video effects like titling, transitions, and compositing are an important tool for enhancing video data and maintaining audience interest [11]. Traditional video effects processing hardware used in the television and film industries, however, is not well suited for streaming Internet video sources which are compressed and experience variable jitter, delay, and packet loss in the network.

We are developing a software-only video effects processing system designed for streaming Internet video. A software-only solution using commodity hardware provides the flexibility needed to match different application requirements and keep up with developments in streaming video technology. The key to a software-only system is to exploit parallelism because current processor speeds are insufficient for real-time processing of many effects (e.g., chroma-key, complex transformations, etc.). Processor improvements will only be met with greater application demands for more complicated effects, larger images, and higher quality.

Our system is an example of a class of applications in which multimedia data types (e.g. compressed packet video) are not simply transmitted and displayed, but manipulated. In the course of building the **P**arallel **S**oftware-only **V**ideo effects **P**rocessing system (PSVP), we have developed new mechanisms and video representations for exploiting different types of parallelism. The design and implementation of these mechanisms and representations are often constrained by Internet media transport protocols (e.g., RTP [16], RTCP [16], etc.) and video formats (e.g., M-JPEG [14], H.261 [17, 9], H.263 [18], MPEG [13], etc.) that were not designed for manipulation. We describe many of these mechanisms in previous publications [7, 8]. In this work, we describe how these mechanisms exchange control information to coordinate themselves as a distributed process.

A key feature of PSVP is a recursive parallel mapping strategy that hierarchically organizes processors to implement a video effect. Parallelism is independently managed at each level of the hierarchy. To control the effect, control messages must be distributed to processors in the hierarchy in a manner that preserves the independence of each level. Traditional methods for controlling a distributed system such as remote procedure calls (RPC) and remote method invocation (RMI) can not satisfy the requirements of PSVP. These requirements include efficient delivery of control messages to groups of processors, tunable reliability semantics associated with control messages on a per message basis, and recoverable state information. To meet these requirements we developed a control mechanism using IP-Multicast [4], the Scalable Reliable Multicast protocol [5], and the Scalable Naming and Announcement Protocol [15]. The resulting control mechanism includes an optimization for composing control elements from different levels of the process hierarchy as well as from two or more different effects.

The main research contributions described in this paper are the recursive parallel mapping strategy and the control mechanisms developed to coordinate the components of PSVP. These mechanisms demonstrate the efficacy of exposing application-level semantic information about the control messages at the transport level.

This paper is organized into 9 sections. Section 2 briefly

To appear in ACM Multimedia 1999

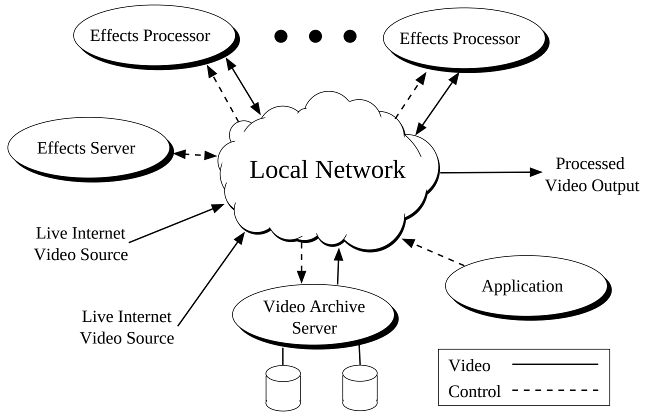

Figure 1: System Architecture

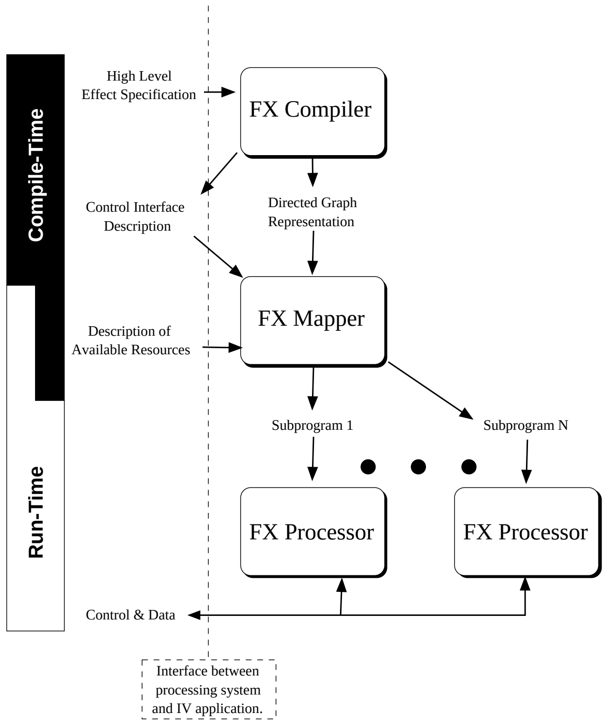
Compile-Time
Run-Time
High Level Effect Specification
FX Compiler
Control Interface Description
Directed Graph Representation
FX Mapper
Description of Available Resources
Subprogram 1
Subprogram N
FX Processor
FX Processor
Control & Data
Interface between processing system and IV application.

Figure 2: Software Architecture

describes the PSVP system to provide a context for the rest of the paper. Section 3 describes in greater detail how processors are organized to provide a parallel implementation of a video effect. The control requirements that stem from how PSVP organizes processors are described in Section 4. The underlying reliable multicast protocols used by PSVP are described in Section 5. Section 6 describes the design and implementation of the control mechanisms. A performance optimization of these mechanisms that provides additional control functionality is described in Section 7. Alternative approaches and related work are described in Section 8. The paper is summarized in Section 9.

2 The PSVP System

This section reviews the system and software architecture of the PSVP system. An understanding of how the system is organized provides a context for the control requirements of the system. Additional details of the system are described in previous publications [7, 8].

The overall system architecture is shown in Figure 1. The oval labeled “Application” represents an Internet video 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 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 (i.e., fade) video effect expressed as a directed graph of primitive operators.

The FX Mapper takes the intermediate representation and maps it onto the available resources. The FX Mapper determines how the effect is parallelized. Section 3 describes the types of parallelization available and how the FX Mapper constructs a parallel implementation for an effect. The FX Mapper produces effect “subprograms” that will be executed on a particular computational resource. The coordinated actions of these subprograms implement the effect.

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.

3 Mapping Strategy

This section describes the strategy used by the FX Mapper to construct a parallel video effect implementation given a directed graph representation of the effect. We describe the three different types of parallelism and a recursive mapping process that creates a layered parallel implementation of the video effect.

The FX Mapper exploits three types of parallelism: functional, temporal, and spatial. Functional parallelism decomposes the video effect into subtasks and maps these subtasks 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. For example, one processor may deal with odd numbered frames while another processor deals with even numbered frames. Spatial parallelism can be exploited by assigning regions of each frame to different processors. For example, one processor may process the left half of each frame while another processor deals with the right half.

The FX Mapper uses a recursive partitioning process to generate possible configurations. A predictive cost model for

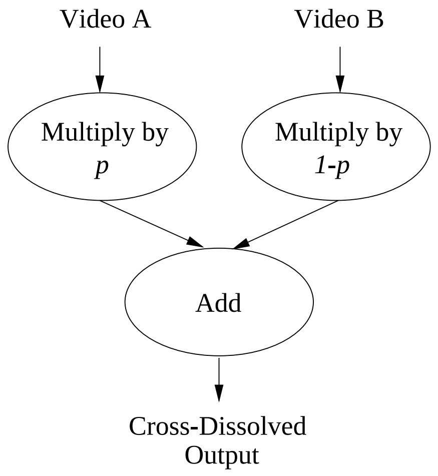

Figure 3: Cross-Dissolve Directed Graph Representation

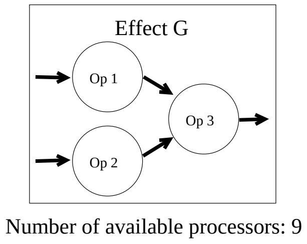

Figure 4: Mapping Strategy Example Part 1

the video operators will be developed to estimate the performance of these configurations and choose the best one. The partitioning process starts by choosing a particular type of parallelism to exploit and dividing the graph into two or more subgraphs. The FX Mapper instantiates mechanisms to coordinate the subgraphs specific to the type of parallelism exploited. Recursively, each subgraph is further processed in the same manner. The base case is a single processor implementation of a subgraph.

For example, figure Figure 4 depicts an effect *G* that is to be mapped onto a set of 9 processors. The effect is represented as a directed graph of three operators (*Op1*, *Op2*, and *Op3*) and takes two inputs and produces one output. At the first level, the FX Mapper may choose to utilize temporal parallelism. To do so, it creates two new subgraphs,

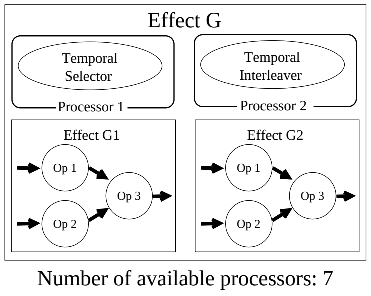

Figure 5: Mapping Strategy Example Part 2

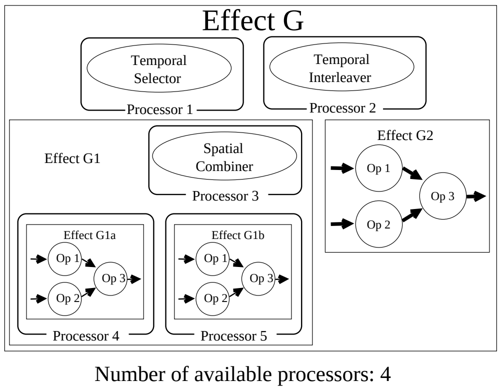

Figure 6: Mapping Strategy Example Part 3

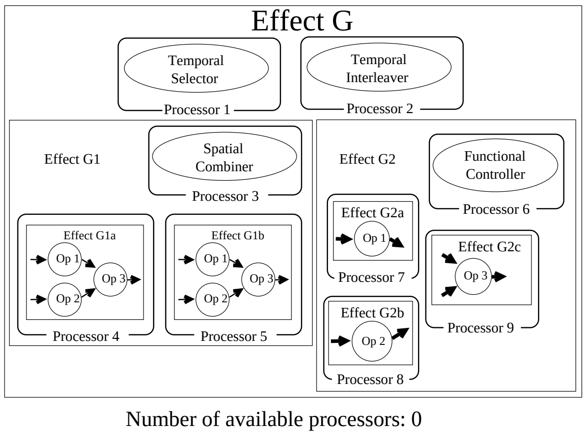

Figure 7: Mapping Strategy Example Part 4

G1 and *G2*, which are copies of the original effect graph *G*. The mapper also creates two mechanisms for managing the temporal parallelism. One mechanism is the *temporal selector* which is responsible for assigning frames to *G1* and *G2*. The other mechanism is the *temporal interleaver* which takes the outputs of *G1* and *G2* and interleaves the results back into one stream. These mechanisms are described in greater detail in a previous paper [7]. The selector and interleaver processes are assigned processors leaving 7 processors for the implementation of *G1* and *G2*. This configuration is illustrated in Figure 5. The mapper can now divide the remaining processors among the implementation of *G1* and *G2* and recursively parallelize their implementation. Suppose 3 processors are assigned to the parallelization of *G1* and 4 processors are assigned to the parallelization of *G2*.

Given 3 processors to parallelize *G1*, the FX Mapper may opt to exploit spatial parallelism. Two copies of *G1* are constructed (*G1a* and *G1b*) and are modified slightly so that one produces the left half of the output stream and one produces the right half of the output stream. A mechanism for controlling and coordinating these processes is instantiated and assigned to a processor. This process, called the *spatial combiner*, is described in a previous paper [8]. Now, only two processors are left for the implementation of *G1a* and *G1b*. Each graph is assigned one processor. No further parallelization is possible and the mapper produces a single processor implementation for each subgraph. This configuration is illustrated in Figure 6.

Given 4 processors to parallelize *G2*, the FX Mapper chooses to exploit functional parallelism. The *G2* graph, which is composed of 3 nodes, is decomposed into 3 new effect graphs, each of which contains a single node (*G2a*, *G2b*, *G2c*). A mechanism for coordinating the actions of these three graphs, called the *functional controller*, is instantiated and assigned to a processor. The mapper constructs single processor implementations for each of the three subgraphs *G2a*, *G2b*, and *G2c* and assigns each to a processor. The final configuration is in Figure 7.

A key feature of the recursive mapping strategy is the separation of each level of parallelism. At any level of parallelism the details of lower and higher levels are hidden. In our example illustrated in Figure 7, the existence of the temporal mechanisms at level *G* is unimportant to the spatial mechanisms at level *G1*.

The processes that implement *G1* are unaware that their actions are being coordinated to exploit temporal parallelism at a higher level. Similarly, the mechanisms instantiated at level *G* to coordinate the actions of *G1* and *G2* are unaware that these effect graphs are implemented using further levels of parallelism. This separation is made possible because the interface exposed by an effect graph implementation is constant. In other words, the interface between the controlling application and the topmost level of the effect implementation is the same interface used between any two levels of the implementation.

By hierarchically organizing processes into levels of parallelism, any portion of the solution hierarchy can be dynamically reconfigured. For example, in the previously illustrated scenario, the implementation of *G1* can be dynamically reconfigured to use additional processors or even a completely different type of parallelism as long as the resulting solution maintains an implementation of *G1*. Single processor implementations of an effect graph (i.e., the leaves of our solution hierarchy) can be dynamically parallelized.

Supporting a dynamic solution hierarchy also allows PSVP to take advantage of advanced distributed programming environments. For example, the Berkeley Network-of-Workstations (NOW) project provides mechanisms for starting a software process on a collection of computers in a load balancing manner. The AS-1 active services framework provides mechanisms for starting and maintaining processes on a collection of computing resources [1]. The Condor project provides an environment in which processes can be migrated between processors to exploit idle computing resources [6]. Leveraging these advanced services is simplified if PSVP assumes nothing about the location of its component processes.

To support a dynamic solution hierarchy, the mechanisms that implement a specific level of parallelism need to coordinate the abstract effect graphs immediately below without any knowledge of exactly how these effect graphs are implemented. In our example, the two temporal mechanisms at the topmost level are aware of each other and coordinate with each other to manage the underlying levels *G1* and *G2*. These mechanisms are responsible for translating any control messages from higher levels to the appropriate messages for lower levels. Distributing control messages is challenging because higher level mechanisms are not aware of how lower level effect graph abstractions are implemented. The number, location, and organization of processes at the lower levels must remain hidden.

4 Control Requirements

The previous section described the hierarchical organization of processes implementing an effect. This section describes the control information needed to manage an effect and the requirements for distributing these control messages.

To control an effect, the controlling agent, which is an application or parallel mechanisms at the next higher level, must know or be able to discover the number and type of inputs, outputs, and parameters for the effect. Each input, output, and parameter is identified by a name and has one or more controllable attributes. The attributes in the current implementation are summarized in Table 1.

Trigger commands are control messages that cause an effect implementation to produce a frame of output data. Different types of trigger commands are provided to implement different synchronization schemes among input streams. An effect can also be "auto-triggered" by the arrival of new input data. An auto-trigger, which is an attribute of a particular input, is listed in Table 1.

A completion token is a control message produced by an effect to indicate that an output frame has been produced. Completion tokens provide performance feedback.

The control protocol requirements for distributing control messages must preserve the advantages of the multilayered independent mapping strategy described in the previous section. The first requirement is that the number and location of processors that implement an effect is unknown to the controlling agent. Similarly, the number and location of controlling agents is not known by the processors implementing the effect. We use IP-Multicast [4] to support this requirement. Each level of the effect implementation is associated with a specific IP-Multicast session. Controlling agents and processors that implement the effect level join this session. Control messages are sent and received through this session. IP-Multicast provides efficient delivery of messages to members of the session by replicating and transmitting messages in the network at routers as necessary. The IP-Multicast session acts as a level of indirection. Session members do not know how many other members exist or where they are located.

The second requirement for the control protocol is that different messages are delivered with different levels of reliability. Trigger commands and completion tokens, for example, do not need reliable delivery because their value is significantly diminished over time. Messages that communicate the number and type of inputs, outputs, and parameters, however, should be reliably delivered since this information is valid for the lifetime of the effect. Messages setting the value of a particular attribute of an input, output, or parameter, should only be delivered reliably if the message is the most recent update for that attribute.

Finally, we require that the current state of an effect (e.g., attribute values for inputs, outputs, and parameters, etc.) be recoverable at any time. This soft-state approach allows for dynamic reconfiguration and robustness.

The next section describes the Scalable Reliable Multicast and the Scalable Naming and Announcement protocols that are used to implement the PSVP control protocol.

	Attribute	Description
Inputs	source	Specifies which RTP stream to use as this input. The specification includes the multicast address, port number, and source id of the video stream.
	auto trigger	If set to 1, indicates that the effect should produce output frames for every input frame received.
Outputs	dest	Specifies the multicast session that this output stream should be sent to. Specification includes the multicast address and port number of the session.
	geometry	This attribute is used by spatial parallelism mechanisms to specify the portion of the output frame to be produced. The attribute is set to four values from 0.0 to 1.0 which define the upper left and lower right corners of a subregion relative to the true frame size.
	format	Indicates the desired format for output frames. Possible value currently include M-JPEG, H.261, and SC. SC is a semi-compressed format we developed specifically to support spatial parallelism.
Parameters	type	Indicates the type of the parameter. Possible types include: real, integer, text, and list.
	domain	Specifies the domain of the parameter. Domain values are type specific. For example, for real parameters, domain is indicated by two real values indicating the range of possible values for the parameter.
	value	The current value of the parameter.

Table 1: Attribute descriptions for inputs, outputs, and parameters.

5 SRM and SNAP

The Scalable Reliable Multicast (SRM) [5] protocol extends IP-Multicast to provide reliable delivery of data through a multicast session. The protocol is an example of a receiver reliable protocol in which receivers, and not sources, are responsible for detecting losses and requesting repairs. Any member of the multicast session can respond to repair requests if it can provide the necessary data. The repair requests and the retransmission of data in response to them use a scalable feedback mechanism based on “multicast damping.” Multicast damping uses randomly selected timer values to prevent more than one session member from making the same repair request or retransmitting the same data.

A key feature of SRM is selective reliability. Since receivers are independently responsible for detecting and recovering from losses, each receiver can decide to recover losses based on application requirements. Some receivers may need to recover all lost data, some may tolerate losses of certain types of data, and some may detect losses but delay recovery until the data is actually needed.

To take advantage of selective reliability tuned to application needs, receivers must distinguish the relative importance of lost data. Unfortunately, SRM does not provide this information. Packets of data in SRM are given sequence numbers and loss is detected by gaps in the sequence number space of received packets. The sequence number of the lost packet is unable to convey to the receiver what type of data the packet contains. In short, the application-level semantic structure of data required to exploit selective reliability is lost when the data is mapped to a single linear namespace of packet sequence numbers.

Raman and McCanne developed the Scalable Naming and Announcement Protocol (SNAP) [15] to overcome this shortcoming of SRM. Built on top of SRM, SNAP provides a hierarchical namespace that applications can use to expose the semantic structure of data at the transport layer. This concept of tailoring network mechanisms to match application semantics is an example of Application Level Framing [3]. With SNAP, each source of data in the SRM session is associated with a tree of “containers.” Initially, each source starts with a tree comprised of a single root container. Sources can create and name new containers as children under any existing containers. In this way, a hierarchical namespace of containers is built on a source by source basis. The namespace information for each source is reliably disseminated using SRM.

Sources label each unit of transmitted data (i.e., packet) as belonging to a particular container within the namespace tree. SNAP maintains a sequence number space for each container. Packets are delivered to receivers labeled with the source from which they originated, the container with which it is associated, and the sequence number within that container. When lost packets are detected, receivers are notified to which container the lost packet belongs. Receivers can use this container information to selectively repair lost packets.

To take full advantage of SNAP, applications must construct namespace hierarchies that adequately expose the application-level semantic relationships of the transmitted data. We use SNAP as a foundation for control messages in PSVP. The following section describes how control messages are organized to meet the requirements outlined in the previous section.

6 PSVP Control

Control messages in PSVP are organized into the following namespace hierarchy. Under the root container are five containers labeled *inputs*, *outputs*, *parameters*, *triggers*, *map commands*, and *misc*. For each input of the effect, a child

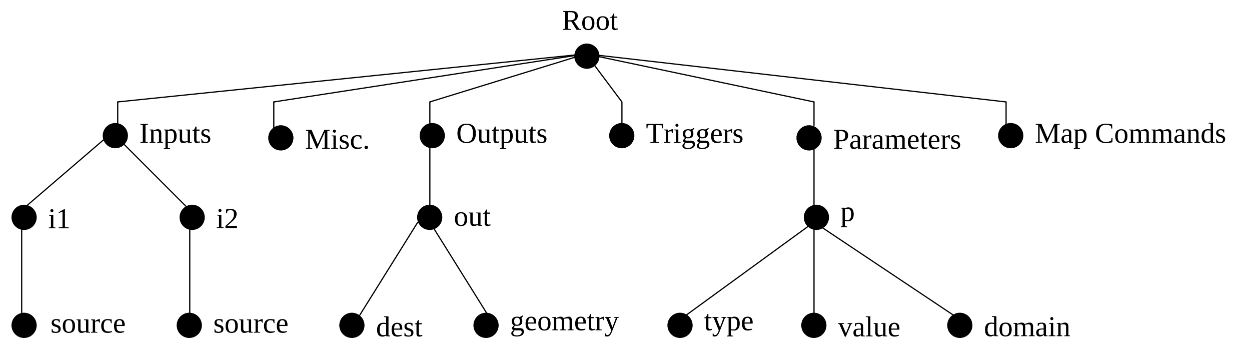

Figure 8: Control Namespace for Cross-Dissolve

container is constructed under the *inputs* container and labeled with the name of the input. For example, Figure 8 illustrates the namespace representing a cross-dissolve effect with two inputs labeled *i1* and *i2*. For each attribute of an input, a child container is constructed under the container associated with that input and labeled with the name of the attribute. In Figure 8, the inputs *i1* and *i2* have an attribute called *source* which is represented by child containers of the *i1* and *i2* containers. Outputs and parameters are handled similarly. In Figure 8, the output of the cross-dissolve effect is represented by a child of the *outputs* container and is labeled *out*. Attributes of the *out* output are represented by children of this container and are labeled with the attributes names (e.g., *dest* and *geometry*). The parameter *p* in our example is represented by a child container of the *parameters* container and has attributes *type*, *domain*, and *value*. The *triggers* container is used for trigger commands and completion tokens. The *map commands* container is used to implement control features described in Section 7, and the *misc* container is used for debugging and miscellaneous messages.

Participants in a control session can be one of two types: controlling agents or implementation agents. Applications like user interfaces and automated video production processes are controlling agents that require control over a video effect. Single processor implementations of a video effect are implementation agents. Parallel processing mechanisms that manage the use of temporal, spatial, and functional parallelism act as both implementation agents and controlling agents. To higher levels of parallelism and applications, these mechanisms are implementation agents. To lower levels of parallelism and single processor implementations of effect graphs, these mechanisms act as controlling agents.

Each level of the effect implementation is associated with a different multicast control session. Parallel processing mechanisms participate in multiple control sessions. They participate in the control session for their own level as well as the control sessions for each of the next lower levels that the mechanisms are coordinating. For example, in Figure 7 which shows a 9 processor implementation of an effect, there are 8 different control sessions with one for each level of the effect (i.e., *G*, *G1*, *G2*, *G1a*, *G1b*, *G2a*, *G2b*, and *G2c*). The temporal mechanisms at level *G* also participate as controlling agents for *G1* and *G2*. The spatial combiner at level *G1* also participates as a controlling agent for *G1a* and *G1b*. The functional controller at level *G2* also participates as a controlling agent for *G2a*, *G2b*, and *G2c*.

Implementation agents communicate the structure of an effect (i.e., number of inputs, number of outputs, parameters, etc.), by creating container nodes as children of the *inputs*, *outputs*, and *parameters* nodes. Various attributes of inputs, outputs, and parameters are described by creating a subcontainer for each attribute under the associated container.

Controlling agents set a particular attribute by transmitting its value as a packet in the attribute's container. When the packet is received by other participants, the container information indicates which attribute of which input, output, or parameter is being set and the data in the packet provides the new value. Receivers tune the selective reliability mechanisms for these attribute containers to reliably receive only the last transmitted packet. To illustrate this point, we refer to the example namespace for the cross-dissolve effect shown in Figure 8. If an implementation agent receives packets with sequence numbers 1 and 3 for the *source* attribute of the input *i1*, it will be notified that packet 2 has been lost. The agent will not issue a repair request because the information sent in packet 2 is known to be old since packet 3 has already been received. If the SNAP mechanisms discover a tail-loss of packets 4, 5, and 6, only packet 6 will be recovered since it represents the most up to date and current value of the attribute.

The control messages sent in the *triggers* container include trigger commands and completion tokens described earlier. These messages have limited temporal value and no reliability is associated with this container. The *misc* and *map commands* containers are used for a variety of different messages including control mapping messages which are described in the next section. Because these control messages need to be sent reliably, receivers invoke the recovery mechanisms for all losses.

Using this naming scheme with SNAP and SRM, we can achieve our primary design goals outlined in the previous section. Our first design goal of hiding the number and location of both the controlling agents as well as the implementing agents is achieved by using multicast. Our second design goal of associating different reliability semantics with different types of control messages is achieved by creating a namespace structure in SNAP that groups related control messages together. Our last design goal of being able to recover the current state of an effect is supported by SNAP's ability to reconstruct the current namespace combined with mechanisms for recovering relevant messages in each of the containers.

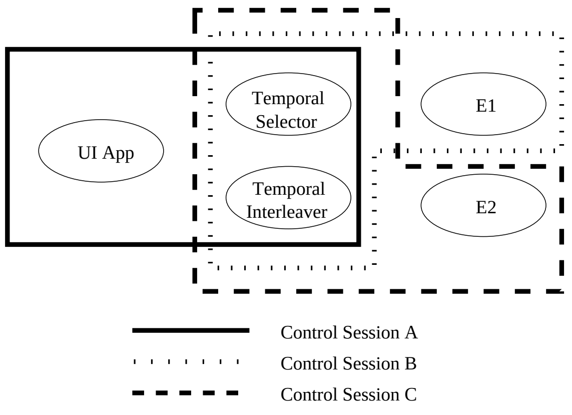

Figure 9: Example Control Session Organization with Temporal Parallelism

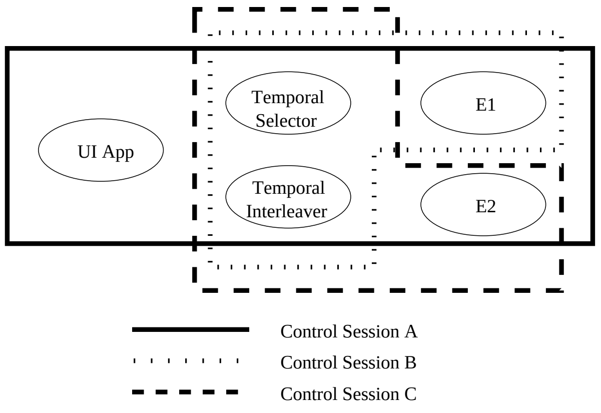

Figure 10: Example Control Session Organization After Map Command

7 Control Mapping Feature

The responsibilities of the temporal, spatial, and functional parallelism mechanisms give these entities dual roles with respect to control. They participate as implementing agents for higher levels of parallelism as well as controlling agents for the lower levels of parallelism that they coordinate. A consequence of this dual role is the need to participate in more than one control session and translate control messages from one session into another. Often control messages from higher levels can be passed directly to lower levels. To avoid this forwarding latency, we developed control mapping commands that enable portions of one control session to be mapped into another. This section describes this capability.

Figure 9 shows an example using temporal parallelism. Pictured on the left is a UI application that controls the effect. The effect is implemented with temporal parallelism and involves two processes for controlling the parallelism. These two processes are labeled as the "Temporal Selector" and the "Temporal Interleaver." The selector and interleaver processes coordinate the actions of E1 and E2 which are independent implementations of the effect. E1 and E2 may be single processor implementations, or they may be further parallelized. The UI application participates in control session A along with the selector and interleaver. The selector and interleaver processes also participate in control session B to direct the actions of E1 and control session C to direct the actions of E2. These control sessions are depicted in Figure 9 as boxes that group the participating processes.

As described in the previous section, E1 and E2 create containers in their SNAP namespace to indicate the existence of inputs, outputs, and parameters and subcontainers to indicate attributes. In our example, suppose E1 and E2 are cross-dissolve effects that use the control namespace shown in Figure 8. The temporal selector and interleaver processes detect these containers in control sessions B and C and reflect the structure of the effect by constructing a congruent namespace in control session A. In this way, information from lower levels is exposed to higher levels and eventually to the controlling application.

The UI application sends messages in particular containers to set attribute values for inputs, outputs, and parameters, issue trigger commands, and, in general, control the effect. The commands are translated by the temporal selector and temporal interleaver into the appropriate commands for E1 and E2. For example, if the UI application issues a command to set the *source* attribute of *i1* (i.e., specify which stream should be used as that input), the temporal selector and interleaver receive this message and take appropriate action. The temporal selector which is in charge of temporally dividing input streams among E1 and E2 translates this command to set the inputs of E1 and E2. The command is not simply forwarded to E1 and E2 because they do not receive the input stream directly, but instead, will receive streams that have been temporally divided by the temporal selector. The temporal interleaver, however, takes no action because it is only responsible for output streams and not input streams. Consequently, the interleaver has no interest in control messages that involve inputs. The interleaver tunes the SRM/SNAP reliability mechanisms to avoid repairing any lost control messages that involve inputs. Similarly, the temporal selector is optimized to deal only with control messages that involve inputs and does not participate in output control messages. One advantage of using SRM and SNAP is the ability to tune the reliability semantics for only portions of the namespace. This example highlights how we capitalize on this advantage.

Some control messages do not need translation by either the temporal selector or interleaver. For example, messages setting the value of a parameter are not translated. These messages need to be forwarded to E1 and E2. Either the selector or the interleaver can be responsible for forwarding these messages. Forwarding messages, however, can create problems with message latency. In our example, E1 and E2 may be further parallelized in different ways. If E1 involves 1 additional level of parallelization and E2 involves 10 addition levels of parallelization, messages that are forwarded to E1 and E2 will experience vastly different latencies. Reducing the latency of control messages improves the responsiveness of the system.

To optimize the control mechanism and avoid forwarding latencies, we added map commands. A map command instructs processes that implement an effect to join and participate in other control sessions for a limited portion of the control namespace. Table 2 lists and describes the map commands we have implemented. These commands are issued in the *map commands* container. Receivers fully recover lost map commands.

With map commands, mechanisms that manage paral-

Map Command	Description
map_session *addr port*	Map the control session specified by *addr* and *port* in its entirety into this session. All messages in all containers from the specified session are processed.
map_inputs *addr port*	Map the "inputs" container from the control session specified by *addr* and *port*. Any subcontainers are also mapped into this session.
map_outputs *addr port*	Map the "outputs" container from the control session specified by *addr* and *port*. Any subcontainers are also mapped into this session.
map_parameters *addr port*	Map the "parameters" container from the control session specified by *addr* and *port*. Any subcontainers are also mapped into this session.
map_input *addr port input_name ?alias?*	Map the subcontainer of the "inputs" container associated with *input_name* into this control session. The *alias* is an optional parameter which if given indicates the name the mapped container should be aliased to in this session.
map_output *addr port output_name ?alias?*	Map the subcontainer of the "outputs" container associated with *output_name* into this control session. The *alias* is an optional parameter which if given indicates the name the mapped container should be aliased to in this session.
map_parameter *addr port parameter_name ?alias?*	Map the subcontainer of the "parameters" container associated with *parameter_name* into this control session. The *alias* is an optional parameter which if given indicates the name the mapped container should be aliased to in this session.
map_triggers *addr port*	Map the "triggers" container from the control session specified by *addr* and *port*.
map_misc *addr port*	Map the "misc" container from the control session specified by *addr* and *port*.
map_map_cmds *addr port*	Map the "map commands" container from the control session specified by *addr* and *port*.

Table 2: Description of map commands currently implemented.

lelism like the temporal selector and interleaver can map portions of the higher level control session that need to be forwarded directly into the lower level control sessions. In our example, the temporal selector issues map commands to E1 and E2 to map the *parameters* container and all subcontainers from session A into sessions B and C. Figure 10 shows which processes participate in each control session after these map commands are executed. E1 and E2 join and participate in session A as well as their original sessions, but only for the parameters portion of the session A namespace. When the UI application sends control messages for a parameter, these messages are now directly received by E1 and E2 with no forwarding by the temporal parallelism mechanisms. If E1 and E2 contain further levels of parallelism, the original map command is properly translated and/or forwarded to each level. Only processors that require parameter control messages map the *parameter* container of control session A into their own control session. All non-parameter control messages in session A are ignored by E1 and E2 and losses of non-parameter control messages are not repaired.

By using map commands, mechanisms that implement the three types of parallelism avoid the responsibility of forwarding control messages that do not have to be handled or translated. These messages are directly received by whatever processes require them at any level of parallelism.

An advanced feature of map commands is the ability to aggregate and compose control elements. For example, when temporal parallelism is exploited, the temporal interleaver can provide controlling agents (i.e., mechanisms at higher levels of the implementation hierarchy or the controlling application) a parameter to govern how much buffering latency should be allowed when constructing the interleaved output stream. This parameter is not part of the effect itself but is specific to the temporal interleaver mechanism and only exists when temporal parallelism is exploited. The interleaver can "add" this parameter to the effect implementation by constructing the appropriate subcontainer in the parameter portion of its control namespace. Controlling agents higher in the hierarchy than the temporal interleaver treat the new parameter as it would any other. Implementation agents lower in the hierarchy than the temporal interleaver are unaware of the extra parameter and are unaffected by its presence.

An additional advanced feature of map commands is mapping control messages with aliasing. Aliasing is when a container of one control session namespace is mapped into another control session with a different name. This feature

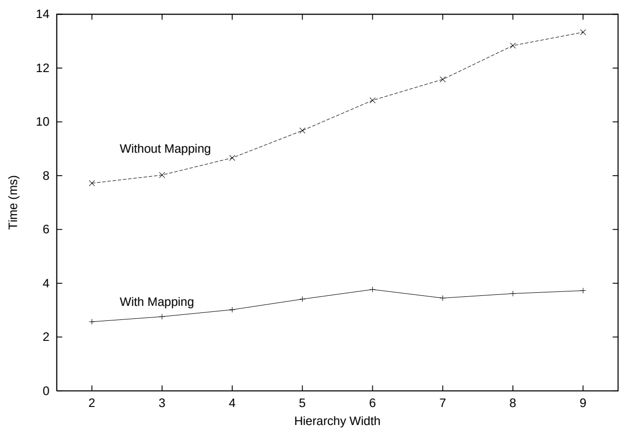

Figure 11: Time required to distribute control messages to a shallow hierarchy of varying width.

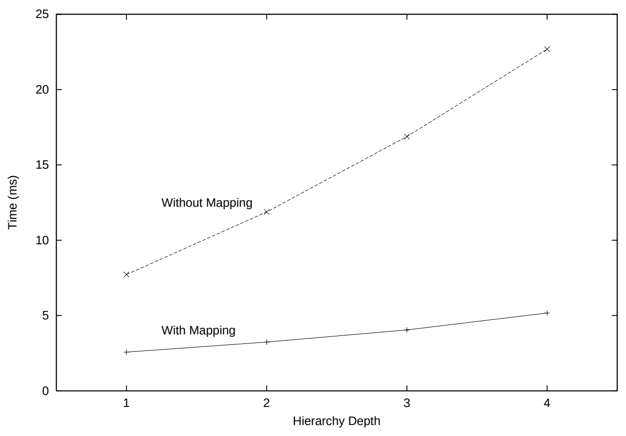

Figure 12: Time required to distribute control messages to a binary hierarchy of varying depth.

enables a variety of flexible and interesting control structures. For example, if an application is controlling two different effects, Effect A and Effect B, and the application needs parameter "theta" of Effect A to be equal to parameter "alpha" of Effect B, the container describing parameter theta can be mapped and aliased into the control session of Effect B with the name of parameter alpha. Messages controlling theta for Effect A will be interpreted by Effect B as messages controlling alpha.

To measure the effectiveness of the mapping optimization, we constructed hierarchies of distributed processes and measured the time required to distribute a control message to the leaves of the hierarchy with and without using the mapping optimization. The experiments were conducted on the Berkeley NOW which is comprised of Ultra Sparc-1 workstations connected by a 10 Mb/s switched Ethernet network. Figure 11 shows the results using a shallow hierarchy comprised of one root node and between two and nine children. Using the mapping optimization, the leaves of the hierarchy all participate in the topmost control session and receive control messages directly. Thus, even as the number of leaves grows, the time for distributing control messages remains relatively constant and small (i.e., around 2-3 milliseconds). Without the mapping optimization, the root node must unicast each control message to each child. Thus, the time for delivering control messages grows with the number of children. Figure 12 shows similar results with binary tree hierarchies of varying heights.

8 Related Work

In this section we describe alternative approaches to distributing control information used in distributed and parallel systems. The most traditional communication primitive for distributed systems is some form of *remote procedure call* (RPC). The Remote Method Invocation (RMI) mechanism in Java is an example of an RPC-like service. RPC mechanisms, however, are fundamentally location specific. A client process must be able to specifically address the server process to invoke the RPC. In addition, RPC mechanisms generally do not allow for relaxed reliability requirements.

Horus [2] is a protocol toolkit specifically designed to support flexible communication requirements among groups of processes. Horus supports dynamically changing process groups and a variety of flexible reliability and consistency models. The key feature of SNAP leveraged by our system is the ability to dynamically construct a hierarchical namespace within which to address our control messages, transcending the limitations of a single numeric transport-level sequence number namespace. This feature is not immediately supported by Horus. Horus is flexible enough as a protocol framework to implement an equivalent solution, but doing so would essentially entail writing new protocol modules for Horus that provided the services entailed in SNAP and using existing Horus modules to provide the functionality of SRM.

9 Summary

This paper describes the control mechanisms built with SRM and SNAP for software-only parallel video effects processing. The PSVP system uses a recursive multi-level mapping strategy to parallelize video effects. As a consequence of this strategy, several requirements are made of any mechanism used to distribute and translate control messages. These requirements include efficient delivery of messages to all processors, tunable reliability semantics on a per control message basis, and recoverable state.

Traditional distributed system control mechanisms, do not meet these requirements. Our approach to the problem uses IP-Multicast to provide efficient delivery of messages along with SRM and SNAP to provide tunable reliability semantics and recoverable state. We achieve this objective by organizing control messages into a namespace that reflects application level semantics and groups related control messages. This organization was described and its use illustrated by several examples.

We also describe an optimization of the control mechanism to avoid unnecessary forwarding of control messages through each layer of parallelism. The optimization allows portions of one control session to be mapped into another. We extended this optimization with aliasing which allows the mapped portion of the control namespace to be renamed automatically. With aliasing, we can construct flexible control mechanisms that relate control attributes of different effects.

PSVP is implemented using the Berkeley MASH toolkit [10] and Cornell's Dali video manipulation language [12] in C++ and OTcl. We have used PSVP to generate real-time effects for RTP streams with H.261 and M-JPEG video data as part of the Berkeley Multimedia, Interfaces, and Graphics seminar series.

References

[1] Elan Amir, Steven McCanne, and Randy Katz. An active service framework and its application to real-time multimedia transcoding. *Computer Communication Review*, 28(4):178–189, October 1998.

[2] Kenneth P. Birman, Robber van Renesse, and Silvano Maffeis. Horus: a flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[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] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[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 1995.

[6] M. Litzkow, M. Livny, and M.W. Mutka. Condor - a hunter of idle workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.

[7] Ketan Mayer-Patel and Lawrence A. Rowe. Exploiting temporal parallelism for software-only video effects processing. *Proceeding of ACM Multimedia '98*, pages 161–170, 1998.

[8] Ketan Mayer-Patel and Lawrence A. Rowe. Exploiting spatial parallelism for software-only video effects processing. *Proceeding of SPIE Multimedia Computing and Networking 1999*, pages 252–263, 1999.

[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] Wei-Tsang Ooi et al. The dali multimedia software library. *Proceedings of SPIE Multimedia Computing and Networking 1999*, pages 264–275, 1999.

[13] International Standards Organization. *Coded Representation of Picture, Audio, and Multimedia/Hypermedia Information*, December 1991. Committee Draft of Standard ISO/IEC 11172.

[14] International Standards Organization. *Digital Compression and Coding of Continuous Tone Still Images*, February 1991. JTCI Committee Draft of Standard ISO/IEC 10918.

[15] Suchitra Raman and Steve McCanne. Scalable data naming for application-level framing in reliable multicast. *Proceeding of ACM Multimedia '98*, pages 391–400, 1998.

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

[17] International Telecommunication Union. *Video codec for audiovisual services at p*64kb/s*, March 1993. ITU-T Recommendataion H.261.

[18] International Telecommunication Union. *Video coding for low bit rate communication*, February 1998. ITU-T Recommendation H.263.