Report Number: WUCS-90-39

1990-11-02

Segment Streaming for Efficient Pipelined Televisualization

Fengmin Gong

The importance of scientific visualization for both science and engineering endeavors has been well recognized. Televisualization becomes necessary because of the physical distribution of data, computation resources, and users involved in the visualization process. However, televisualization poses a number of challenges to the designers of communication protocols. A pipelined televisualization (PTV) model is proposed for efficient implementation of a class of visualization applications. Central to the proposed research is the development of a segment of streaming IPC (interprocess communication) mechanism in support of efficient pipelining across very high speed internetworks. This requires exploration of special issues arising from extending... **Read complete abstract on page 2.**

Segment Streaming for Efficient Pipelined Televisualization

Fengmin Gong

Complete Abstract:

The importance of scientific visualization for both science and engineering endeavors has been well recognized. Televisualization becomes necessary because of the physical distribution of data, computation resources, and users involved in the visualization process. However, televisualization poses a number of challenges to the designers of communication protocols. A pipelined televisualization (PTV) model is proposed for efficient implementation of a class of visualization applications. Central to the proposed research is the development of a segment of streaming IPC (interprocess communication) mechanism in support of efficient pipelining across very high speed internetworks. This requires exploration of special issues arising from extending a pipeline across networks with errors and high latency, determination of alternative solutions, and evaluation of such solutions. The novel aspects of the proposed segment streaming mechanism include a two-level flow control method and an intelligent error control mechanism.

SEGMENT STREAMING FOR EFFICIENT PIPELINED TELEVISUALIZATION

Dissertation Proposal

Fengmin Gong

WUCS-90-39

November 1990

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Segment Streaming for Efficient Pipelined Televisualization

(Dissertation Proposal)

Fengmin Gong
lfg@wucs1.wustl.edu
(314) 726 4163

November 2, 1990
WUCS-90-39

Abstract

The importance of scientific visualization for both science and engineering endeavors has been well recognized. Televisualization becomes necessary because of the physical distribution of data, computation resources, and users involved in the visualization process. However, televisualization poses a number of challenges to the designers of communication protocols.

A pipelined televisualization (PTV) model is proposed for efficient implementation of a class of visualization applications. Central to the proposed research is the development of a segment streaming IPC (interprocess communication) mechanism in support of efficient pipelining across very high speed internetworks. This requires exploration of special issues arising from extending a pipeline across networks with errors and high latency, determination of alternative solutions, and evaluation of such solutions. The novel aspects of the proposed segment streaming mechanism include a two-level flow control method and an intelligent error control mechanism.

Segment Streaming for Efficient Pipelined Televisualization

(Dissertation Proposal)

Fengmin Gong
lfg@wucs1.wustl.edu
(314) 726 4163

1. INTRODUCTION

1.1. Motivation

The 1987 NSF report *Visualization in Scientific Computing* defines visualization as a method of computing which transforms symbolic information into geometric information, enabling the researchers to gain better insight into their simulations and computations [39]. It has been further pointed out that visualization is a critical tool for discovery and understanding, as well as a tool for communication and teaching [16].

Scientific visualization has emerged as a major computer-based field of study due to the following three key factors: (1) large-scale computations and high bandwidth data sources (e.g. medical scanners and satellite imaging) generate huge amounts of data that scientists cannot analyze in numerical form in reasonable time [25, 49]; (2) it is believed that the human vision system has a data bandwidth on the order of gigabits per second, but this bandwidth is not exploited by text-oriented data presentation methods which limit the data bandwidth made available for interpretation [61]; and (3) images can convey information far more effectively than numerical data [39].

The use of visualization for many scientific applications has been well documented [62, 29, 40]. New visualization applications have been rapidly emerging with developments in visualization methodology and underlying technologies. In short, visualization is beginning to revolutionize the way research is done in various disciplines of science and engineering.

Televisualization is visualization that utilizes data and computing resources that are physically distributed. There are three components in the visualization process, namely data, computation, and user interaction. As long as the locations of these components are not all the same, the need for televisualization arises. For example, in a three-tiered computing facility environment such as the one at NCSA [25] and as envisioned in the NSF report, a large simulation is run on a supercomputer while a scientist views the visual output and steers the simulation at a workstation. Furthermore, the scientist may wish to do parts of the visualization computation on separate machines in order to distribute the computation load and achieve better performance. Therefore, we believe a significant fraction of practical visualization will be televisualization.

It is interesting and important to note that many visualization applications fit into a pipeline model [25]. For example, steps of a visualization process such as data preparation, mapping of raw data to attributes of visual data, and graphics rendering are readily identifiable stages of a pipeline. It is this class of applications that will be considered in this research effort. Since televisualization is important, and a pipeline is an appropriate computation model for many televisualization applications, a natural question is:

Can we successfully use the pipeline model across high speed networks to make demanding televisualization applications possible?

I believe the answer to this question is yes and therefore propose to demonstrate the viability of the pipelined televisualization model through this research.

1.2. Support for Pipelined Televisualization

Efficient pipelined televisualization (PTV) requires support far beyond what is needed for visualization on a single computer. It requires networks with high bandwidth and low latency, an efficient interstage communication mechanism on the networks, and proper adaptation and partitioning of the visualization computation. We elaborate on the networking and pipelining aspects of PTV in the following paragraphs.

Networking

Suitable network support is critical to the success of televisualization. Trends in networking suggest that the necessary support will be possible. Fast packet switching networks are being developed in many research laboratories (e.g. BPN at Washington University [59], Sunshine at Bellcore [22], and ATOM at NEC [56]). These networks can support communication channels with high transmission rates and low queueing delay. They will have some resource allocation capability and provide performance guarantees. In order to make this guaranteed network bandwidth available to applications, the host-network interface must be designed to support the high data rates. A number of research groups have also begun addressing these issues [3, 11, 30, 54].

However, it is important to note that host-to-host connections with high bandwidth and performance guarantees are necessary but not sufficient conditions for successful deployment of demanding distributed applications. As a result of decreased queueing delay in newer networks, the speed of light propagation delay will increasingly dominate the communication latency in wide area networks. The networks will continue to have more packet errors and losses than a local environment. Moreover, the new networks have much higher bandwidth-delay products which affect flow and error control strategies at the application and transport levels. It is the responsibility of applications and the transport protocols to cope with these conditions and to convert the high bandwidth into high performance for the applications. It has been recognized that suitable solutions can be found by developing deep understanding of the communication requirements of various classes of applications [42]. In the case of PTV, the pipelinedprocessing and high communication latency coupled with the demand for high performance present a unique set of interprocess communication (IPC) requirements that have not been adequately addressed by existing research on networking or visualization.

Pipelining

The pipeline principle has been successfully used in many ways and at different levels of abstraction to achieve high performance. For example, instruction pipelines are used in most new processor designs to obtain high overall processing speed. Arithmetic pipelines are extensively used to achieve high performance for floating point and vector operations. Pipelining is also one of the major principles used in systolic array and wavefront array processors [33, 35]. A pipeline with parallel processes at each stage has been used as a parallel computation model on a multiprocessor system for certain scientific computations [20]. Many methodologies for the design and operation of hardware pipelines have been developed [32, 27, 15]. The proposed use of the pipeline model across networks for televisualization, however, is a new idea and requires significant research. Intrastage computation, interstage communication, and their interaction become important factors affecting PTV performance.

There are two major requirements for efficient pipelining: engineering the pipeline to achieve minimum and equal cycle time for all stages; and operating the pipeline with minimum delay or interruption. To satisfy the first requirement in televisualization pipelines, visualization computation has to be partitioned with respect to the communication overhead and allowed to overlap as much as possible with the communication process. Appropriate flow and error control mechanisms are required to minimize the effect of the network latency and errors on the operation of the pipeline.

Open Issues

To summarize, specific issues concerned with this research for supporting pipelined televisualization are the following:

- How can visualization algorithms be adapted for efficient pipelining?
- How is the visualization computation partitioned and stages of the pipeline mapped to distributed computers?
- What IPC paradigm should be used for interstage communication to achieve maximum overlap between the communication and intrastage computation?
- What is an effective and efficient interstage flow control method for the pipeline?
- How can application-oriented error control be used to achieve better performance for PTV?

The rest of this proposal is organized as follows: Section 2 describes the pipelined televisualization model in more detail; Section 3 discusses in depth a set of research problems in pipelined televisualization and provides the focus for this research; Section 4 explains the proposed solutions; Section 5 outlines the plan for action; Section 6 reviews other relevant work; and Section 7 summarizes the expected contributions.

2. PIPELINED TELEVISUALIZATION MODEL

This section is a more detailed discussion of the pipelined televisualization model. First, pipelining is discussed in the context of concurrent computing, and a descriptive definition of the pipeline as a computational model is presented. Then, the pipelined televisualization model is defined and the applicability of a televisualization for pipelined processing is shown.

2.1. Pipeline and Concurrency

Concurrency, in which multiple operations of a larger computation are carried out within a given time interval, is one of the most important goals for computer science research. Two well-recognized techniques for achieving concurrency by simultaneous operation on multiple computing elements are *parallelism* and *pipelining*. While parallelism achieves concurrency by replicating a hardware structure many times and having each replication execute in parallel different parts of the large computation, pipelining splits the hardware structure into a sequence of substructures (called stages) corresponding to phases of the computation task and achieves concurrency by allowing all stages to operate simultaneously on different parts of the computation [32, Chapter 1].

It has been pointed out that pipelining is the most distinctive, powerful and ubiquitous means for achieving concurrency in computer architectures [15, 18]. Pipelining has also been recognized as an important computation model at the programming level [20, 9]. Instruction pipelining and arithmetic pipelining in modern computers are good examples of pipelining for achieving *fine-grained parallelism*, ie. the basic operations being overlapped are relatively simple. On the other hand, the application of pipeline principles to obtain *large-grained parallelism* are also numerous. For example, a road-following algorithm for robot vehicles has been implemented on the CMU Warp in an 8-stage pipeline [34]. A general architecture called a pipeline net has been proposed for performing vector compound functions (VCF's)[1] [28]. The pipeline net is constructed from multiple functional pipelines that can be arbitrarily interconnected through a crossbar network, which is itself pipelined.

In summary, the success of pipelining principles can be attributed to three important factors. First pipelining supports concurrency; a pipeline with m stages can potentially offer an m-fold speedup over a nonpipelined computation. Second, many computations can be pipelined at various levels of granularity. Third, pipelining avoids many difficulties associated with the parallel approach. For example, there is no need for broadcast communication, and the sequentiality in the pipeline's data flow makes synchronization much easier.

It is the pipeline as a computation model that is of interest to this research. A descriptive definition of the pipeline that emphasizes this modeling aspect will now be presented.

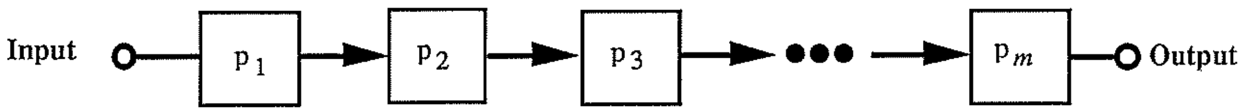

Figure 1: An m-stage Pipeline

Assume we need to perform a computation C on n data items $d_1, d_2, \dots, d_n$ independently. Instead of designating one processor P for the computation C, we first partition C into m subcomputations $c_1, c_2, \dots, c_m$, such that a complete computation on each data item d_i is accomplished by applying $c_1, c_2, \dots, c_m$ in sequence. Next, we designate one processor for each subcomputation, so that we get m corresponding processors $p_1, p_2, \dots, p_m$. Finally, we create a link between each pair of neighbor processors $(p_i,\ p_{i+1},\ (1 \leq i \leq m-1))$, such that when a data item is finished computing at p_i it can be passed to p_{i+1}. We have thus defined a *pipeline* for the computation C. Figure 1 is a graphical representation of such a pipeline. Data are fed into the pipeline one at a time from the input end. Each data item undergoes computation c_i as it passes through processor p_i. By the time a data item reaches the output end, it will have finished the complete computation C. These

[1] A VCF is a collection of linked scalar operations to be executed repeatedly many times in a looping structure.

processors performing subcomputations will be called *stages* of the pipeline in subsequent discussions.

Notice that this model does not exclude parallelism or pipelining within each stage of the pipeline. For example, the second stage processor (p_2) can be a multiprocesor itself, executing the corresponding stage function c_2 in parallel on several (sub)processors; this is one level below the pipeline abstraction referred to here. In principle, the pipeline can either operate in lock-step fashion by a global clock or by having each stage processor synchronize only with its neighbors through a handshaking protocol. In the first case, the pipeline is called a synchronous pipeline; the second case corresponds to an asynchronous pipeline. While a synchronous pipeline is simple and efficient in many cases, an asynchronous pipeline is necessary when stages of the pipeline are physically separated or have unequal processing times.

2.2. Pipelined Televisualization Model

The pipelined televisualization model is the application of the pipeline model (defined in Section 2.1) to a class of televisualization computations. In particular, stages of the televisualization pipeline will reside on computers distributed across networks. This subsection first discusses the computation model for pipelinable visualization and then presents a high level communication model for the televisualization pipeline.

2.2.1. Computation Model.

We start by considering a visualization process that is very useful for study of cell development in biology. In this study a researcher takes a number of cells, marks them using some coloring techniques, and then allows the cells grow under controlled conditions. At certain time intervals snapshots are taken of the cells in the form of a set of digitized sectional images; these 2-dimensional images describe the state of cells in 3-dimensional (3D) space for that instance of time. For the time period of study, a set of volume images will have been collected. One desirable way of visualizing the data is to create a sequence of 3D scenes of the cells using polygons, each from one volume image of the development process. The researcher can then interactively view each scene by rotation, zooming and so on. Furthermore, animations showing the development of the cells can be created which will allow observation of the dynamics and serve to communicate with other researchers. This visualization process involves five steps: deblurring of each sectional image, image segmentation and contour generation, surface tiling, rendering of the surfaces, and user interaction. Each of these steps is now discussed in more detail.

Deblurring. Because of limited focusing accuracy of the microscope, sectional images are often blurred. Filtering algorithms are applied to reduce the blur.

Segmentation and contour generation. Areas corresponding to each cell on sectional images have to be identified and their boundary contours generated.

Surface tiling. Once sets of planar contours are generated, a polygon-based surface tiling algorithm is used to generate polygons connecting corresponding contours on adjacent planes.

Surface rendering. The list of polygons, together with additional user input (such as lighting and viewing information), is used to produce images on display devices.

User interaction. Since there may be processing parameters with each of above steps that can be fine tuned for better visualization (e.g. use of different deblurring algorithms), the user should be able to modify these parameters and observe the effect. Interpreting user requests and interfacing with other steps is the function of user interaction.

These steps form a natural partition of the computation. Sectional images are collected under the control of a laboratory computer and saved in mass storage. This computer could also conveniently work on the deblurring task. Another computer may serve as the stage for segmentation and contour generation. The biologist also has access to a supercomputer for the intensive computation required by the surface tiling. Visual output needs to be presented on a graphics workstation that is remote from both the supercomputer and the lab. We therefore assign rendering and user interaction to the graphics workstation. This results in a televisualization pipeline. For high efficiency, we choose a sectional image (2D) as the granularity of data processing in the pipeline instead of the entire 3D volume.

It should be pointed out that, for this application, it is possible to perform all the computation on the laboratory computer and ship image frames to the workstation, or to ship blocks of raw data to the workstation which performs all the computation. However, with PTV model, it is possible to utilize four appropriate computers in a 4-stage pipeline. Sectional image granularity is used to achieve high pipelining efficiency. The speedup due to pipelining provides better visualization performance.

There is a significant class of visualization applications whose computation contains several well-defined stages [37, 60, 26, 25]. We can generalize them into the following five-stage description:

Data generation. This is the step in which the original data is generated. Numerical simulation and image scanning are two examples.

Data preparation. This stage derives visualization data from the observational, experimental, or simulation data. For example, for visualization of a simulation, additional data points may be interpolated to transform an irregular grid into a regular one.

Visualization mapping. This stage constructs abstract visualization objects (AVO [25]) from the data derived at the previous stage. Specifically, it maps the derived data (e.g. pressure, temperature, intensity) to the attributes of the AVO (e.g. geometry, color, opacity) for graphics rendering.

Graphics rendering. Abstract visualization objects generated in the last step are rendered into images for display at this step. The actual operations may include image processing, surface rendering, and volume rendering.

User interaction. The user interacts with the visualization process through the image display and the input devices such as a keyboard, joystick, and dials. Interpreting user input and interfacing with rest of the computation stages are functions of this stage.

These stages can be readily mapped onto a pipeline. In addition, computation at some stages may be further partitioned, thus creating more stages for the pipeline. In the cell development example, deblurring, segmentation, and contour generation are all data preparation functions, but they were mapped onto two pipeline stages.

2.2.2. Communication Model.

Applying the pipeline model to these televisualization applications leads naturally to a PTV implementation. Figure 2 illustrates, at a high level, the communication model for a visualization process (VP) at stage k to pass data to another visualization process at stage $k+1$ which is separated by an internetwork. Interprocess communication is provided through a set of communication protocols, which logically consists of the transport protocol (TP), the internetwork access protocol (IAP), and the network access protocol (NAP). The direct link shown between the two stages is an abstraction of the underlying internetwork. In practice, performance of the underlying internetwork hardware and the communication protocols will have direct impact on the performance of pipelined televisualization. The emphasis of the proposed research is on the interface area between the visualization application and the communication protocols (shown as shaded).

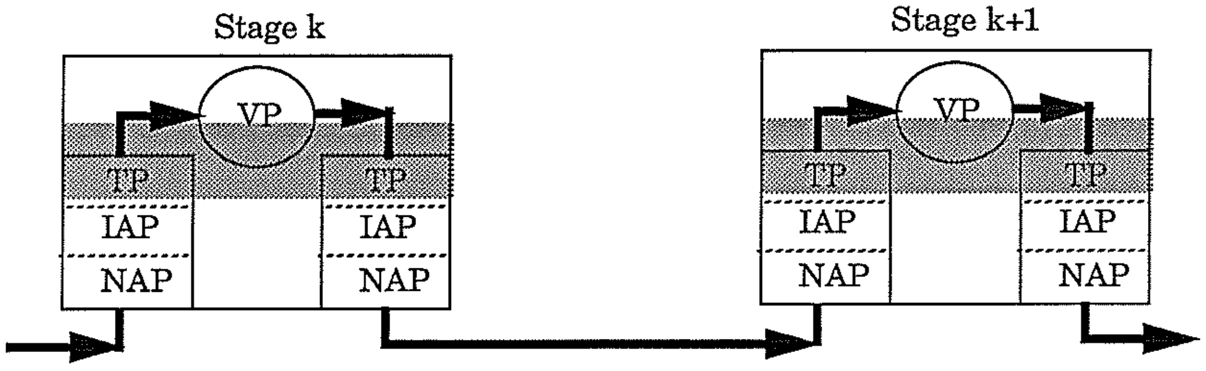

Figure 2: Interstage Data Communication Path

It should be noted that having the abstract PTV model does not guarantee acceptable performance for televisualization applications. Efficient application and communication support is needed for practical implementations. Therefore, our next task is to identify the important issues in supporting the PTV model.

3. RESEARCH ISSUES IN DETAIL

This section first reviews the operation and performance metrics of an ideal pipeline. Then, televisualization pipelines are examined to identify important research issues involved. The focus for this proposed effort will be detailed.

3.1. The Ideal Pipeline

Space-time diagrams[2] have been widely used [10, 15, 24, 27] to depict the processing of tasks in a pipeline and to conduct timing analysis. A space-time diagram is a two-dimensional diagram with time as the horizontal axis; it contains as many rows as there are stages in the pipeline and as many columns as there are distinct intervals of time that need to be represented. Entries in a row show tasks that are processed by the corresponding stage over successive time intervals. We define the *cycle time* of a pipeline stage to be the minimum delay between two sucessive outputs of data

[2]Space-time diagrams are also called Gantt charts. The term space-time diagram is more widely used in the context of pipelines and is also self explanatory.

items from the stage. The cycle time of a pipeline is defined as the maximum cycle times of all the stages. The treatment of ideal pipelines in this section is based on the treatments given in [15, 24, 27].

A pipeline is *ideal* if it satisfies the following four conditions:

- all stages take an equal amount of time in processing all data items
- the time for passing data from one stage to the next is negligible
- the pipeline is synchronized by a global clock
- there is no other delay in the data flow in the pipeline

Assume that we have to process n data items, and it takes T time to process one data item without pipelining. We can design an ideal pipeline by partitioning the processing into m subtasks each taking T/m time, which becomes the cycle time of the pipeline. Figure 3 shows the space-time diagram for an ideal pipeline with $m = 3, T = 3$, and $n \gg 3$. Numbered circles are used to represent data items being processed. The n data items enter the pipeline at one per unit time from the first stage, moving down the pipeline at the same rate, and finally exit at stage 3. It is worth noticing that after 2 units of time, the pipe is fully loaded, and the three stages of the pipeline are operating simultaneously on three data items. All data are processed by time $3+n-1$. It is easy to see that $(m+n-1)(T/m)$ units of time will be taken for a general m-stage ideal pipeline.

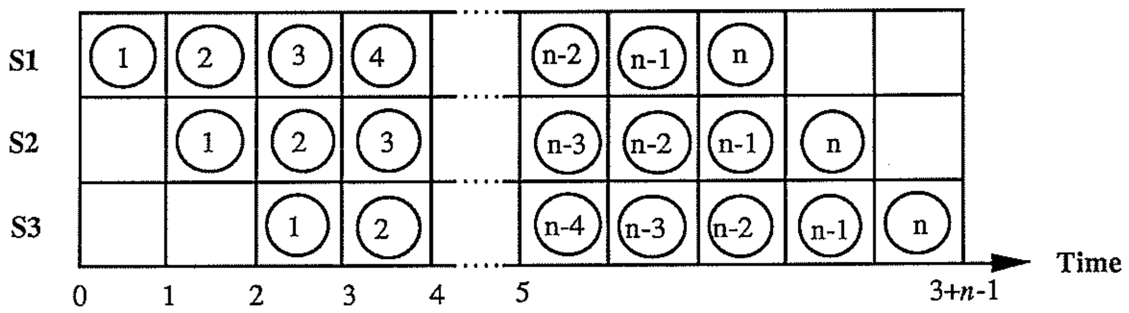

Figure 3: Space-Time Diagram of an 3-Stage Ideal Pipeline

The four most useful metrics for pipeline performance are the ***throughput*** (TH), ***speedup*** (SU), ***stage efficiency*** (SE), and ***pipeline efficiency*** (PE). The throughput of a pipeline is the number of data items completely processed by the pipeline per unit time. For the m-stage ideal pipeline:

$$TH(n,m) = \frac{n}{(m+n-1)\frac{T}{m}} \quad \text{data items/unit time} \tag{1}$$

As $n \to \infty$, $TH(n,m)$ approaches m/T (the reciprocal of the pipeline cycle time) which is the maximum throughput of the pipeline.

The speedup is the ratio between the time it takes to process n data items without pipelining and the time taken by the m-stage pipeline to process the same amount of data. For the m-stage ideal pipeline, this becomes:

$$SU(n,m) = \frac{nm}{m+n-1} \tag{2}$$

$SU(n,m)$ approaches m as $n \to \infty$, resulting in the ideal speedup that one can expect from an m-stage pipeline.

The efficiency of a pipeline stage is the ratio between the number of time units for which the stage is busy and the total number of time units for the pipeline to process n data items. For the ideal pipeline all stages have equal efficiency:

$$SE(n,m) = \frac{n}{m+n-1} \tag{3}$$

The limit of $SE(n,m)$ is clearly unity with $n \to \infty$.

The efficiency of the whole pipeline is the average of the efficiencies for all the stages. Thus, for an m-stage ideal pipeline it is the same as $SE(n,m)$:

$$PE(n,m) = \frac{n}{m+n-1} \tag{4}$$

The limit of $PE(n,m)$ is also unity as $n \to \infty$.

The speedup $SU(n,m)$ and the efficiency $PE(n,m)$ are related by:

$$PE(n,m) = \frac{SU(n,m)}{m} \tag{5}$$

With finite n, performance of an ideal pipeline is solely determined by three parameters: processing time per data item in a nonpipelined system T, number of pipeline stages m, and total number of data items n. The state of the pipeline when $n \to \infty$ is called the *steady state*. The limits as $n \to \infty$ of the performance metrics defined above naturally characterize the steady state performance of the ideal pipeline. In reality many pipelines are not ideal. Televisualization pipelines are such examples.

3.2. Televisualization Pipelines

In this subsection, we discuss in more detail the important research issues in televisualization pipelines. We do so by properly extending the treatment of the ideal pipeline. First of all, it is important to note that a televisualization pipeline is different from an ideal pipeline in several ways.

- The very complex computation of visualization makes it almost impossible to obtain perfect partitioning.
- A televisualization pipeline must operate in an asynchronous mode, because the pipeline stages are distributed. Furthermore, asynchronous operation will be necessary for achieving high efficiency with less than perfect partitioning of computation. This means that adjacent stages of the pipeline have to accomplish the exchange of data through explicit message communications.

- The overhead for interstage communication is no longer negligible because of the delays and errors in the underlying networks, and because of the large volumes of data being transferred. The success of PTV depends critically on how well the pipeline can cope with this overhead.
- User interaction with different stages of the televisualization pipeline is also required. The pipelining mechanism has to support such interaction with minimal additional pipeline delay.

Additionally, it should be noted that data items in televisualization have a much larger granularity than typically has been the case for other pipelines. How these differences will affect the performance of the televisualization pipeline needs to be understood.

3.2.1. Algorithms Adaptation and Pipeline Engineering.

The first step in implementing pipelined televisualization is to organize the visualization computation as a linear array of processes $p_1, p_2, \ldots, p_k$ as shown in Figure 4.

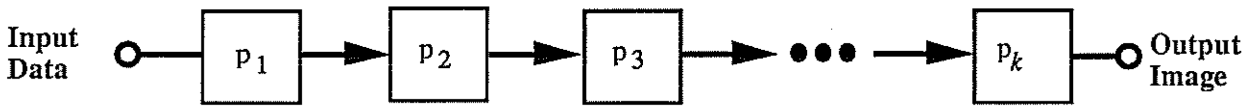

Figure 4: A Pipelinable Computation Form

Each segment of visualization data is first processed by p_1, the result from p_1 is then fed to p_2, and so on until the final result is produced by p_k. In general, process $p_i(1 < i < k)$ receives an input from p_{i-1} and sends the result to p_{i+1}. The objective is to adapt the processes to work with the smallest data granularity. The importance of a smaller data granularity is shown in the following paragraphs.

Assume that we have an m-stage ideal pipeline to process n data items; let T be the time for processing one data item without pipelining and T/m be the cycle time of the pipeline; then the speedup of this pipeline is as given by Equation (2):

$$SU = \frac{nm}{m+n-1}$$

Suppose we can reorganize the computation such that each input data item is further partitioned into q ($q > 1$) parts (new data items). If the overhead due to partitioning is negligible, each stage now only takes $T/(mq)$ time and there are nq new data items. The speedup SU_q is therefore:

$$SU_q = \frac{nmq}{m+nq-1}$$

It can easily be shown that SU_q is a monotonically increasing function of q for $m > 1$. This means that, for a given computation and a fixed number of pipeline stages, use of smaller data granularity leads to larger speedup. In practice, the effectiveness of using smaller data granularity will be limited by the overhead from the partitioning and the communication. Therefore, for high PTV performance it is critical to adapt the visualization algorithms such that they can work with smaller data partitions (e.g., subgrids of a simulation result, CT data slices) in an incremental fashion. The overhead introduced in the adaptation process should be kept minimal.

Once the visualization computation is in a pipelinable form, careful engineering is required to map the sequence of processes onto a set of computers. The objective is to obtain equal cycle time for all stages of the pipeline. To appreciate the impact of unequal partitioning on pipeline performance, we derive a relationship between performance and a measure for unequalness of partitioning. The approach taken is adapted from [27].

Assume an m-stage televisualization pipeline that satisfies all the ideal conditions (Section 3.1) except that not all stages have equal processing time. Let t_i be the cycle time (same as the processing time given the assumption) for stage i $(1 \leq i \leq m)$. Let T be the total processing time for one data item on a single computer which serves as a reference for speedup calculation:

$$T = \sum_{i=1}^{m} t_i$$

Suppose that stage j $(1 \leq j \leq m)$ has the longest cycle time:

$$t_j = \frac{T}{m} + \delta \qquad \delta > 0$$

where δ is the additional time required at stage j above that for the ideal partitioning. Then stage j becomes the bottleneck of the pipeline. Assuming $n \gg m$ for PTV applications, we can approximate the pipeline throughput by its steady state value:

$$TH(n,m) = \frac{1}{t_j} = \frac{m}{T + \delta m} \tag{6}$$

The speedup and efficiency are correspondingly given as:

$$SU(n,m) = \frac{mT}{T + \delta m} \tag{7}$$

$$PE(n,m) = \frac{T}{T + \delta m} \tag{8}$$

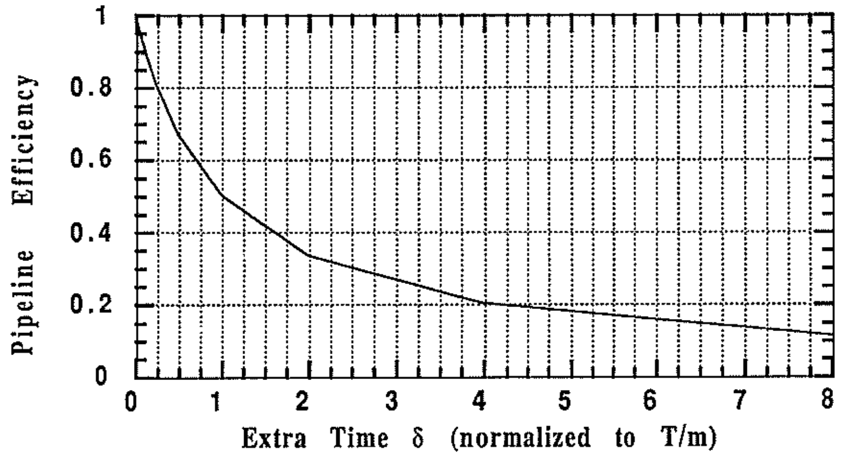

Figure 5: Effect of Unequal Partitioning on Pipeline Efficiency

Figure 5 shows a plot depicting the relationship between the efficiency and extra time δ. The vertical axis represents the pipeline efficiency; the horizontal axis represents δ relative to T/m. The pipeline efficiency shows significant decrease with increased δ. For example, efficiency of the pipeline reduces to 0.5 when δ reaches T/m; the throughput and the speedup of the pipeline are limited to 50% of the maxima possible.

Therefore, an important issue is how to partition a visualization computation to achieve equal cycle times among the pipeline stages.

Algorithm adaptation and partitioning are application dependent issues. These issues overlap with the general problems of computation partitioning, process assignment, and scheduling in a multiprocessor system [21]. But it should be noted that much higher and less predictable communication overhead, as well as the potential variations in execution speed of pipeline stages, make these issues more difficult in televisualization pipelines. These issues will be addressed for two target applications to a degree that will enable us to understand their implication on the issues of interstage communication and synchronization, as well as error control.

3.2.2. Computation and Communication Overlap.

In this subsection, we demonstrate that reducing the interstage communication overhead and overlapping communication with the visualization computation within a pipeline stage are both essential to high performance of a televisualization pipeline.

Consider an m-stage televisualization pipeline that is ideal except for a significant interstage communication overhead. Also assume for now that visualization computation does not overlap with interstage communication. Figure 6 illustrates the timing within a stage k $(1 < k < m)$ under this assumption. For each data item, stage k first receives data from stage $k-1$ using t_r units of time, processes the data in t_p time, and then sends the result to stage $k+1$ which takes t_s units of time.

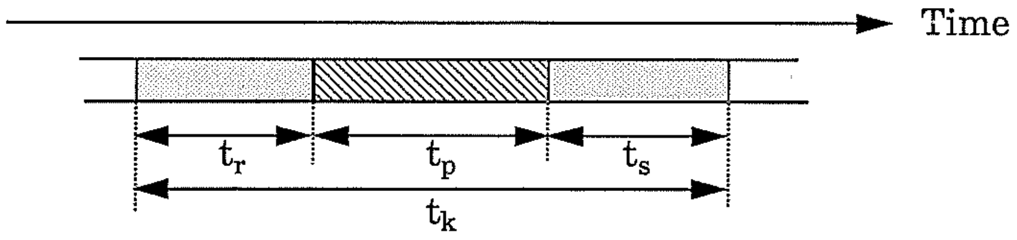

Figure 6: Nonoverlapped Timing within Stage k

The cycle time for stage k is:

$$t_k = t_r + t_p + t_s$$

Let the communication overhead be $t_c = t_r + t_s$, and let R be the ratio of t_c to the intrastage processing time t_p, that is:

$$R = \frac{t_c}{t_p} = \frac{t_r + t_s}{t_p}$$

The pipeline cycle time can thus be rewritten as:

$$t_k = (1 + R)t_p$$

The following expressions for the steady state performance can be derived:

$$TH(n,m) = \frac{1}{(1+R)t_p} \tag{9}$$

$$SU(n,m) = \frac{m}{1+R} \tag{10}$$

$$PE(n,m) = \frac{1}{1+R} \tag{11}$$

Comparing Equations (9) to (11) with (6) to (8), we find that communication overhead affects the pipeline performance in essentially the same way as does unequal partitioning. The plot of R vs. pipeline efficiency as given by Equation (11) is very similar to that for (8) and is not repeated here.

For maximum performance with a given partitioning, it is required that $R = 0$ ($t_c = 0$), but this is impossible to achieve in a real environment. Two approaches can be taken to reduce the impact of the communication overhead. One is to make t_c as small as possible; the second is to overlap t_c with t_p as much as possible.

When there is a partial overlap between the intrastage computation and interstage communication, t_c represents the portion of the communication overhead that is still visible to the intrastage computation. Thus, Equations (9) to (11) are a characterization of the performance of the pipeline for all computation and communication overlaps.

Figure 7 shows an overlapping of computation and communication within the k^{th} stage of a pipeline. The top row represents the receiving of data by the communication interface (data items i and $i+1$ are shown); the middle row corresponds to the processing of the received data ($i-1, i, i+1$); and the bottom row shows the transmission of the data ($i-1, i$).

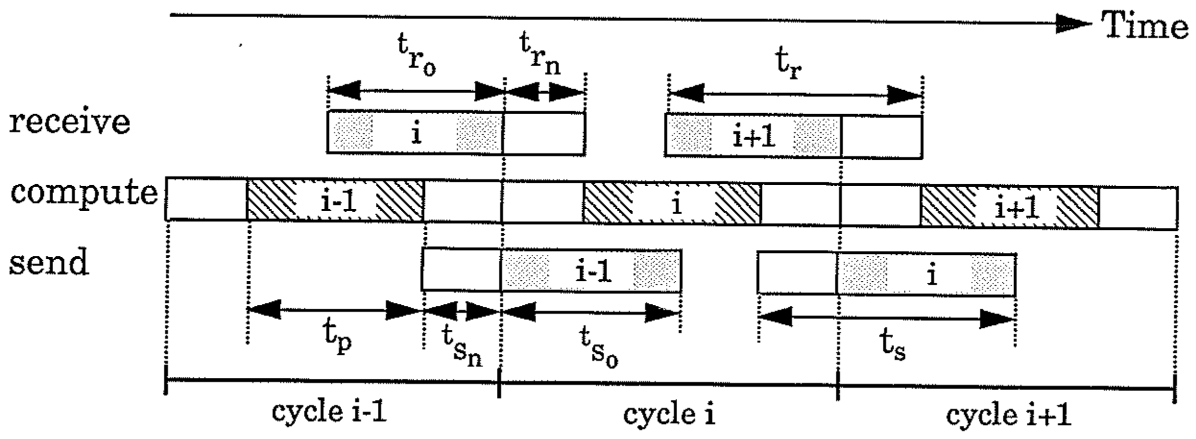

Figure 7: Overlapped Timing within Stage k

The receive time t_r and the send time t_s in Figure 6 are further defined as $t_r = t_{r_n} + t_{r_o}$ and $t_s = t_{s_n} + t_{s_o}$, where t_{r_o} and t_{s_o} represent communication overhead that can overlap with the visualization processing, and t_{r_n} and t_{s_n} represent those parts that cannot; the pure visualization processing takes t_p time per data item, but the local processor will be occupied by either receive or send processing during the periods that cannot overlap. Figure 7 is drawn with implicit assumptions $t_{r_n} = t_{s_n}$ and $t_p = t_{r_o} = t_{s_o}$, the corresponding cycle time is:

$$t_k = t_{r_n} + t_{s_n} + t_p$$

In general, the cycle time for stage k is determined as:

$$t_k = \max(t_{s_n} + \max(t_p + t_{r_n}, t_{s_o}), t_{r_n} + \max(t_p + t_{s_n}, t_{r_o})) \tag{12}$$

Achieving maximum overlap is a difficult issue. First of all, data prefetching is necessary to allow overlap. This raises standard issues regarding the effectiveness of prefetching: how much to prefetch, when to prefetch, and where to store the prefetched data. These issues can only be properly addressed with thorough understanding of the application's data requirement and communication overhead. A high speed host-network interface is required to support parallelism between the data transmission and the local visualization processing. The part of the communication overhead that cannot overlap with the local processing may include time for context switching between application processes and the communication processes, and for moving data between communication buffers and the application space. This overhead must be minimized by appropriate design and implementation of the interface between the visualization application and the communication protocols as well as appropriate buffer management.

3.2.3. Interstage Synchronization and Error Control.

Consider stage S_k $(1 < k < m)$ of a typical m-stage hardware pipeline as shown in Figure 8. There are latches L_k and L_{k+1} between S_k and its two neighboring stages S_{k-1} and S_{k+1} Each input data item has to be fetched from latch L_k, processed, and then loaded to latch L_{k+1}. Access to these latches has to be implemented in such a way that data fetched from L_k is always valid output of stage $k-1$, and data in L_{k+1} will not be overwritten before being used by stage $k+1$. This is the basic function of ***pipeline synchronization***.

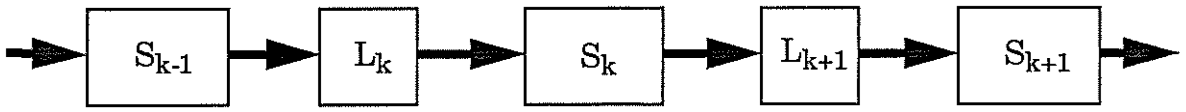

Figure 8: Interstage Synchronization

Since it is infeasible for the televisualization pipeline to use hardware circuitry to directly implement this function, interstage synchronization has to be realized through explicit message exchanges. The interstage synchronization problem then becomes a problem of interstage flow control which will be addressed at the transport level. The suitability of several existing flow control methods (combined with error control) to the televisualization pipeline is examined next. More detailed discussions of these methods can be found in textbooks on computer networks [51, 57]. We only present and discuss the most relevant results applicable to pipeline flow and error control.

Stop and Wait Protocol

The simplest form of flow and error control is *stop and wait*, in which the sender sends only one data segment for each request. The request message can be an acknowledgement (ACK) which causes the sender to send the next data segment, or a negative acknowledgement (NAK) which causes a retransmission of the segment associated with the NAK.

Figure 9 depicts two stages exchanging data segments using the stop and wait protocol. t_{xmt} is the data segment transmitting time. It is clear that only one segment can be sent with every roundtrip delay (t_d) assuming no errors. Stop and wait will be an acceptable solution if arbitrarily large segments can be sent without errors for each ACK/NAK. In practice the use of very large segment sizes is not possible, thus stop and wait leads to very inefficient flow control. This problem is even more significant for a televisualization pipeline which utilizes high data bandwidth in presence of high latency.

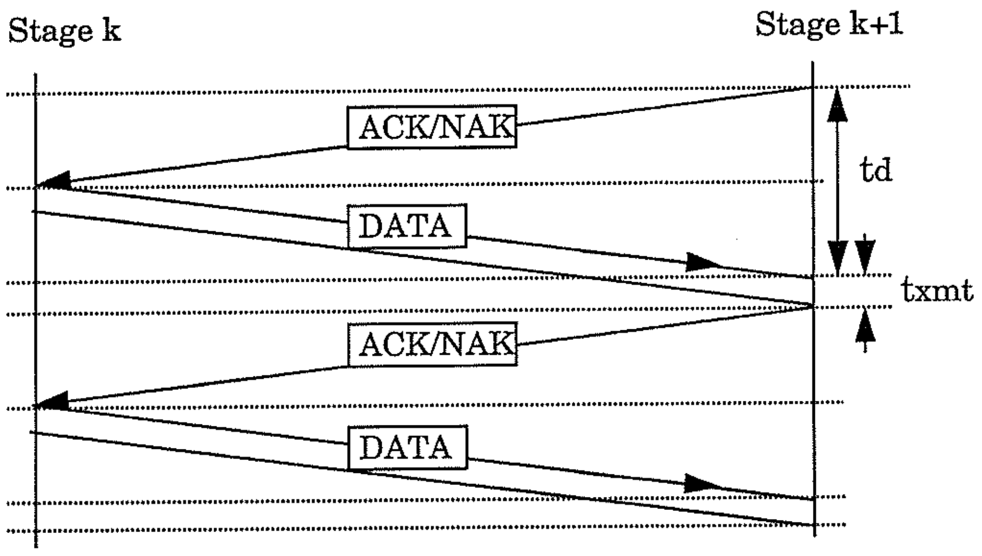

Figure 9: Stop and Wait Protocol

Sliding Window with Cumulative Acknowledgement

In this protocol, a sender can send a series of data segments determined by a window size $\hat{W}$ without receiving an acknowledgement. Each segment has a sequence number corresponding to a window slot. Upon detection of an error, the receiver sends a NAK to the sender and discards all future incoming segments until the segment in error is correctly received. The sender must retransmit the segment in error plus all succeeding segments after the NAK is received.

This protocol improves on the stop and wait by allowing multiple data segments to flow without waiting, thus overcoming the large communication delay.

In case of errors, cumulative acknowledgement will result in the retransmission of multiple segments, some of which are duplicates of good packets, thus bandwidth is wasted. Since the receiver only accepts data segments in sequence, out of sequence data will be discarded and subsequently have to be retransmitted. The cumulative acknowledgement flow control has been shown to limit the effective utilization of the communication link. Combining flow control and error control within one sliding window mechanism leads to their mutual interference. Furthermore, sliding window flow control involves a tradeoff between tight control versus high channel utilization, i.e. while a large window size is required to overcome latency in long haul networks, a smaller window is needed for effective flow control [11, 14].

Most importantly, the fact that the sliding window protocol uses acknowledgements to advance the window makes it ineffective in unreliable networks. A receiver cannot stop the incoming flow by withholding acknowledgements because a sender cannot tell whether an acknowledgement is lost or withheld by the receiver.

Therefore, we believe that the sliding window with cumulative acknowledgement is not a good solution for general high speed network environments, especially for the more stringent performance requirement of PTV applications.

Sliding Window with Selective Acknowledgement

Sliding window with selective acknowledgement improves on connection utilization by selectively acknowledging and retransmitting only data segments in error.

As a potential scheme for flow and error control in high speed network environments, the sliding window with selective acknowledgement still has several disadvantages if considered in its simplest form:

- Receivers and transmitters need more complex control logic. To maintain the sequence, receivers must save all the data segments following the one in error until it is retransmitted; senders need to be able to send out segments in arbitrary order to respond to retransmission requests.
- These mechanisms provide completely reliable communication regardless of application requirements. Even though correctly received data is saved at the receiver, the application is not allowed to access the partially received data. Every data segment lost or corrupted is retransmitted, even if applications can tolerate some lost segments.
- Although selective acknowledgement is more appropriate than cumulative acknowledgement for high speed networks, this scheme has not been widely implemented and still has the disadvantages of interference between flow and error control.

There are two major reasons for considering application-oriented error control in PTV:

- Error retransmission introduces significant communication delay which in turn reduces pipeline performance.
- Not all visualization applications require retransmission of every packet in error.

Thus, simply insisting on providing error-free communication does not guarantee an effective visualization. In addition, the large and variable communication latency makes accurate determination of timer values difficult. Furthermore, bursts of errors together with the high data rates tend to produce less predictable error patterns and loss of large data segments. These are all difficulties facing a PTV error control mechanism. We need to understand application tolerance for errors and design an intelligent error control mechanism.

In summary, we believe that the flow and error control requirements for PTV cannot be satisfied by existing mechanisms. However, the following guidelines can be drawn for the design of the PTV flow and error control:

- The flow and error control functions should not adversely interact with each other.
- The flow control should work well over networks with high bandwidth-delay products and with different classes of applications.
- The error control should integrate the selective retransmission scheme with the application-oriented approach and be able to optimize for different classes of applications.

4. PROPOSED SOLUTIONS

The issues outlined in the previous section can be addressed in the framework of network IPC support for pipelined televisualization. This IPC support should introduce minimum overhead in exchanging data segments between the computation process and the communication system. It should include an effective flow control mechanism to fulfill the synchronization requirement of the televisualization pipeline. Finally, it should have intelligent error control that attempts to satisfy visualization application error tolerance with minimum retransmissions. The proposed solution contains three important parts, namely use of the Axon segment streaming paradigm, a two-level flow control method, and the appropriate application-oriented error control. They will be discussed in this section.

4.1. Axon Communication Architecture

Axon is a high performance communication architecture for distributed systems proposed by Sterbenz and Parulkar [54]. The primary goal of the Axon architecture is to support a high performance data path delivering high (inter)network bandwidth directly to applications. The significant features of Axon are: (1) an integrated design of host and network interface architecture, operating systems, and communication protocols; (2) a network virtual storage system (NVS) which includes support for virtual shared memory across networks [53]; (3) application-oriented lightweight transport protocols for different classes of applications; (4) a network interface which can provide a high bandwidth low latency path directly between the network and host memory [52].

Figure 10 shows a block diagram of the Axon host-network interface architecture [52]. The direct path from CMM (communication memory module) to CMM, consisting of the data path and per packet processing as enclosed by the dashed line, is implemented in hardware as part of the CMP (communications processor). The CMP consists of datapath ($\mathrm{CMP_d}$) and control ($\mathrm{CMP_c}$) portions. The CMP datapath interfaces to the VHSI (very high speed internetwork) optical links and the serial ports of the CMM, and performs such functions as encryption/decryption and data format conversion. The CMP control functions are those directly related to the datapath such

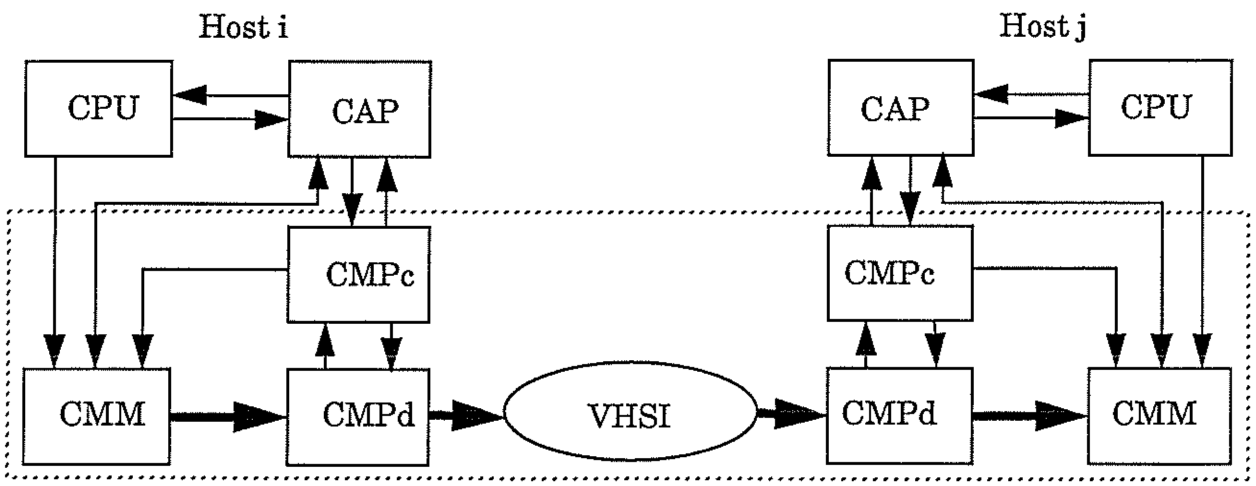

Figure 10: Axon Host-Network Interface

as header build/decode, checksum generate/compare, rate specification timing, as well as the per packet congram[3] multiplexing and control. The CMM is a multiported memory, with serial ports connected to the CMP transmitting and receiving data paths, and random access port available to the host CPU for program execution.

A high performance microprocessor, the CMP assist processor (CAP), performs functions that are not part of the critical path, but require high performance that would be inadequately provided by the host CPU and would adversely impact the performance of other host processes. Examples include packet arrival to page presence mapping and packet retransmission timer management functions. The host CPU is responsible for link/segment/page fault handling (for network virtual storage) and per congram functions.

The Axon host-network interface provides hardware support for high data rates at the host-network interface and for maximum parallelism between the host application processing and the communication processing. With the high bandwidth CMM-to-CMM path available, it is the responsibility of the distributed application and its transport protocol to make efficient use of the path for high application performance. The next three subsections provide a systematic description of the proposed solution for PTV applications.

4.2. Segment Streaming

Segment streaming [53] is an IPC paradigm proposed for supporting exchange of a stream of segments among processes with high bandwidth and low latency. Segment streaming is provided through two transport level operations: *send-stream* which is used to send a segment stream to a remote host, and *get-stream* which is used to retrieve a segment stream from a remote host. Application processes invoke these operations by making corresponding system calls. Segment streaming supports both *repetitive* transmission of a segment and *sequential* transfer of segments defined as a segment group. Initiation of the transmission of each segment (repetitive or sequential) can be based on a specified

[3]A congram combines the desirable features of a datagram with those of a (soft) connection. It can be thought as a connection with the added attributes of rapid setup and survivability in presence of network failures [52].

interval (*interval synchronized*) or the execution of a specified program call (*program synchronized*).

The combination of repetitive transmission with program synchronization within the get-stream operation is the most useful option for PTV applications. A segment buffer will be repetitively transmitted in a stream, with initiation of each transmission determined by the flow control mechanism. Error control is provided by a separate error control mechanism.

The essence of segment streaming is that a single get-stream call (by an application process) performs the request for all of the segments. Each segment will be transmitted when ready across a VHSI connection without the latency of request or setup. Therefore, it supports segment prefetching and allows overlapping between the visualization computation and the communication processing. Provisions will also be made to allow the communication system direct access to the application data space. Overall, with respect to the cycle time of the pipeline (Equation (12) in Section 3.2.2), segment streaming allows packet transmitting time (length/rate) to be part of t_{r_o} or t_{s_o}, and allows t_{r_n} and t_{s_n} to be minimized.

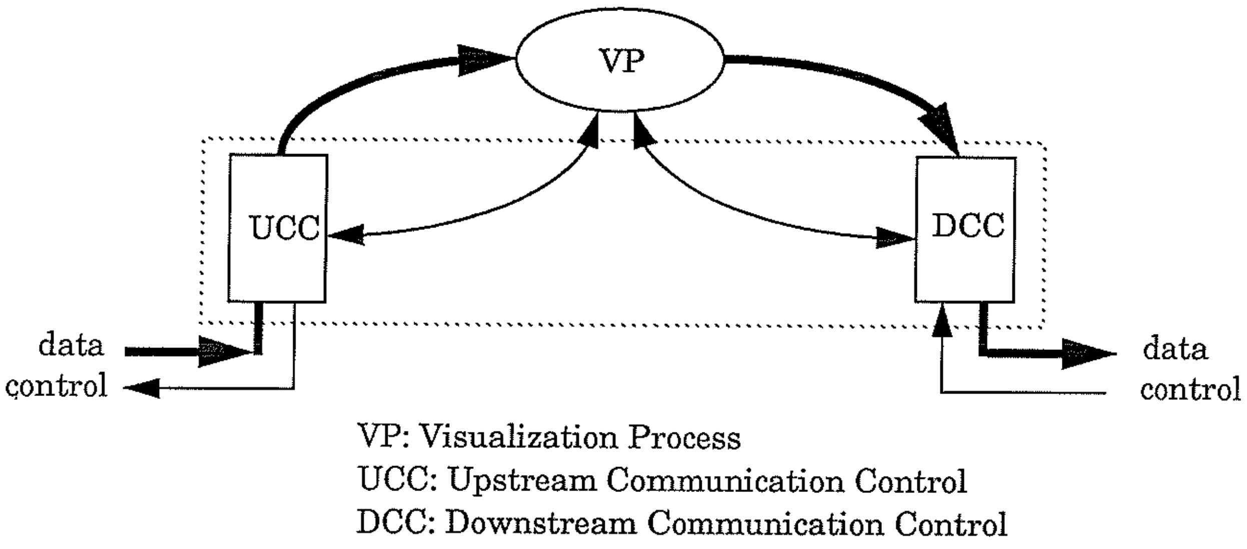
VP
UCC
DCC
data
control
data
control
VP: Visualization Process
UCC: Upstream Communication Control
DCC: Downstream Communication Control

Figure 11: Block Diagram of the Segment Streaming Mechanism

A high level block diagram of the proposed segment streaming for PTV applications is given in Figure 11. The mechanism consists of two major parts: the upstream communication control (UCC) and the downstream communication control (DCC). The upstream control logic is responsible for receiving data segments from the upstream neighbor and the downstream control logic is responsible for sending result segments to the downstream neighbor. Both receiving and sending operations are subject to proper flow control and error control. The data path is shown in thick lines and the control information flows in thin lines. Referring to the Axon host-network interface in Figure 10, visualization process (VP) and high level streaming control will be executed by the host CPU. The UCC and DCC functions will be partitioned among the CAP and CMP. CMP will be assigned those normal per packet processing functions, e.g. rate specification timing and packet encapsulation/decapsulation. CAP will take on functions such as packet retransmission timer management, packet to segment presence mapping, and the simple window control.

Petri nets have been widely used for describing and analyzing concurrent systems [44]. We chose Petri nets for presenting the proposed segment streaming. Informally, a Petri net is a directed bipartite graph with nodes in one partition represented by bars and nodes in the other partition represented by circles. The bars are referred to as transitions and the circles are referred to as places. The input places of a transition are those with an arrow to the transition and the output places are those having an arrow from the transition. A transition will fire when a precondition on the presence of tokens in its input and output places holds. In a traditional Petri net, the precondition may be that all the input places contains at least one token and all the output places contain no more than certain number of tokens. The firing of a transition removes one token from each of the input places and produces a new token to each of its output places. The Petri net model we will use is an extended one based on [41]. This model allows tokens to have attributes and allows a procedure to be associated with a transition.

In a segment streaming representation, places represent buffers. A place may contain tokens which can represent data, control or status information. The transitions represent processing of data and decision making processes. A macro transition, represented by a rectangular box, is recursively defined as a Petri net which may contain simple or macro transitions. Macro transitions will be used to represent some logic at a higher level of abstraction.

Figure 12 shows a further decomposition of the UCC presented in the extended Petri net model. For ease of presentation transition names will be in caps of font SANS SERIF and place names will be in roman **boldfaces**. Three threads of processing can be identified with the three columns in the diagram:

- The receive path on the left contains three simple transitions, the upstream rate control (URC_r), the upstream error control (UEC_r) and the upstream window control (UWC_r). These transitions represent the data path consisting of normal per packet processing for incoming data segments from the upstream neighbor.
- The send path on the right contains correspondingly, three simple transitions UWC_s, UEC_s and URC_s. The control packets (pmt for window control and NAK for error control) normally follow this path to the upstream neighbor.
- The middle column (shaded) consists of three macro transitions for window control, error control, and rate control respectively. This more complex logic is activated only in response to exceptional flow and error conditions.

Places in the diagram consist of two types. The first type includes the "internal" places which indicate intermediate states of data and control information within UCC. For example, **pkt_recv** signals the arrival of a packet from the rate control and **pkt_ok** indicates a packet that has passed error checking; the second type of places are interface points between UCC and other control logic (e.g. **pkt_in** for packet from the network, **VP_st** for the status of visualization process).

The downstream part (DCC) of the segment streaming can be similarly decomposed. This is not shown due to the lack of space. It should be noted that simplifying the data paths and increasing concurrency among the three threads will make segment streaming more efficient.

4.3. Two-Level Flow Control

The purpose of the pipeline synchronization is to guarantee orderly and efficient data flow in the pipeline. This problem is addressed by the design of an interstage flow control mechanism. The objective of the flow control is to allow processed data segments to be sent to downstream neighbors as

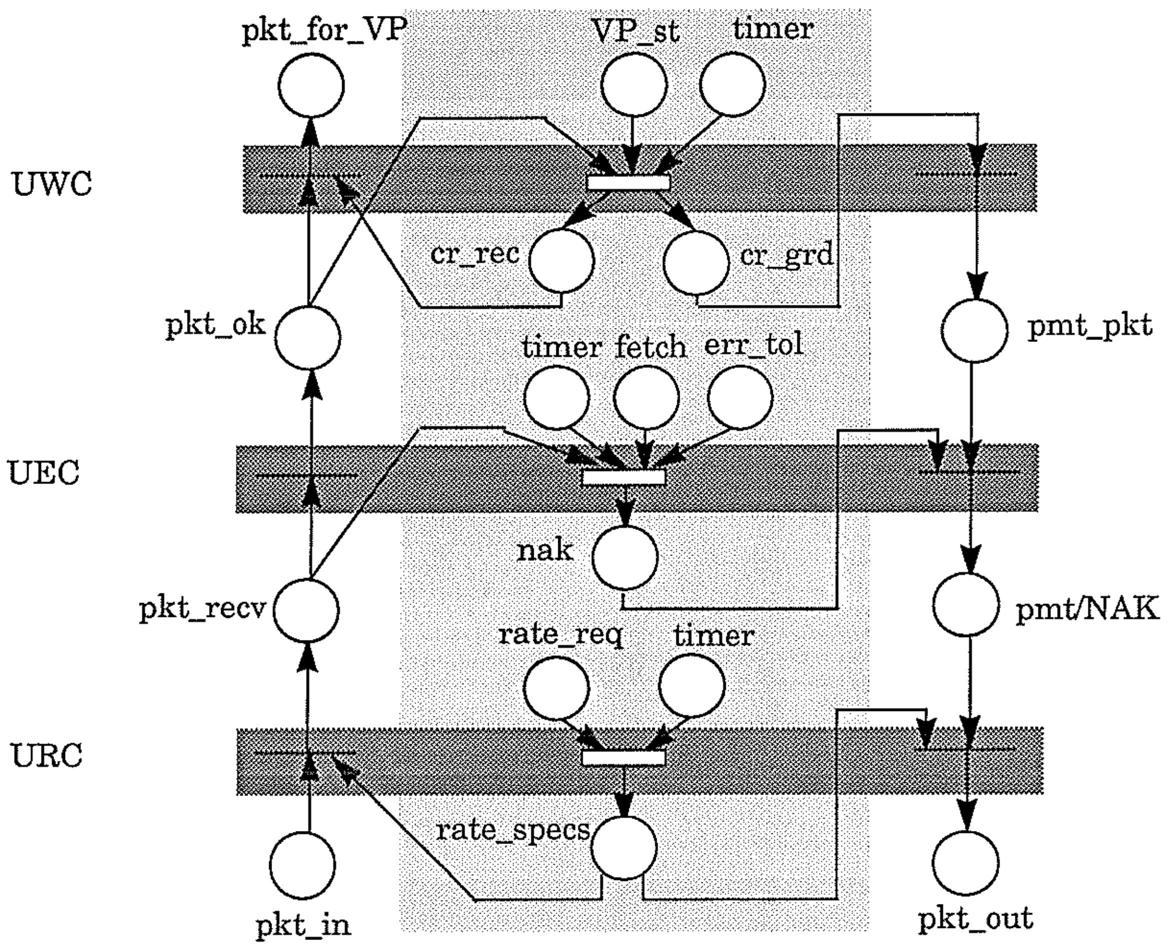

Figure 12: Petri Net Model for UCC

soon as possible but without causing overflow. This objective will be achieved through the proposed two-level flow control method at the transport level.

The two-level method consists of a simple window flow control on top of the rate control mechanism. The simple window control directly interfaces with the visualization process within each stage by receiving and sending segments for the process. The rate control mechanism enforces data transmission and reception rates which are derived from the flow requirement of the simple window control and agreed upon by the underlying network at the time of connection setup.

The specifics of the rate control will be adopted from a related research [1, 8]. The enforcement of rate will be based on derivatives of the leaky bucket model [1]. The following discussion will concentrate on the window control portion. The window control is exercised between two neighbor stages of the pipeline. The receiver starts by sending a control packet (a permit) to the sender specifying the next expected segment number N and the credit limit W (the maximum number of segments the sender can send without further permit). The sender can send a segment only if there are available window slots $(N, N+1, \ldots, N+W-1)$. A segment is sent as a sequence of packets, each carrying a segment number, a packet number within the segment, and a credit confirmation number (which is the concatenation of one toggle bit with the most recently received credit limit). During pipeline operation, the sender keeps track of the consumption of credits and the newly issued credits. The receiver accepts only packets with valid credit and makes decisions as to when to grant new credit to the sender and how much credit to grant.

A Petri net graph is given in Figure 13 which is an expansion of the upstream window control in Figure 12. This logic accomplishes three major functions: checking input packets for valid credit (the window slot number), dynamically controlling the receiving window, and retransmitting window control permit.

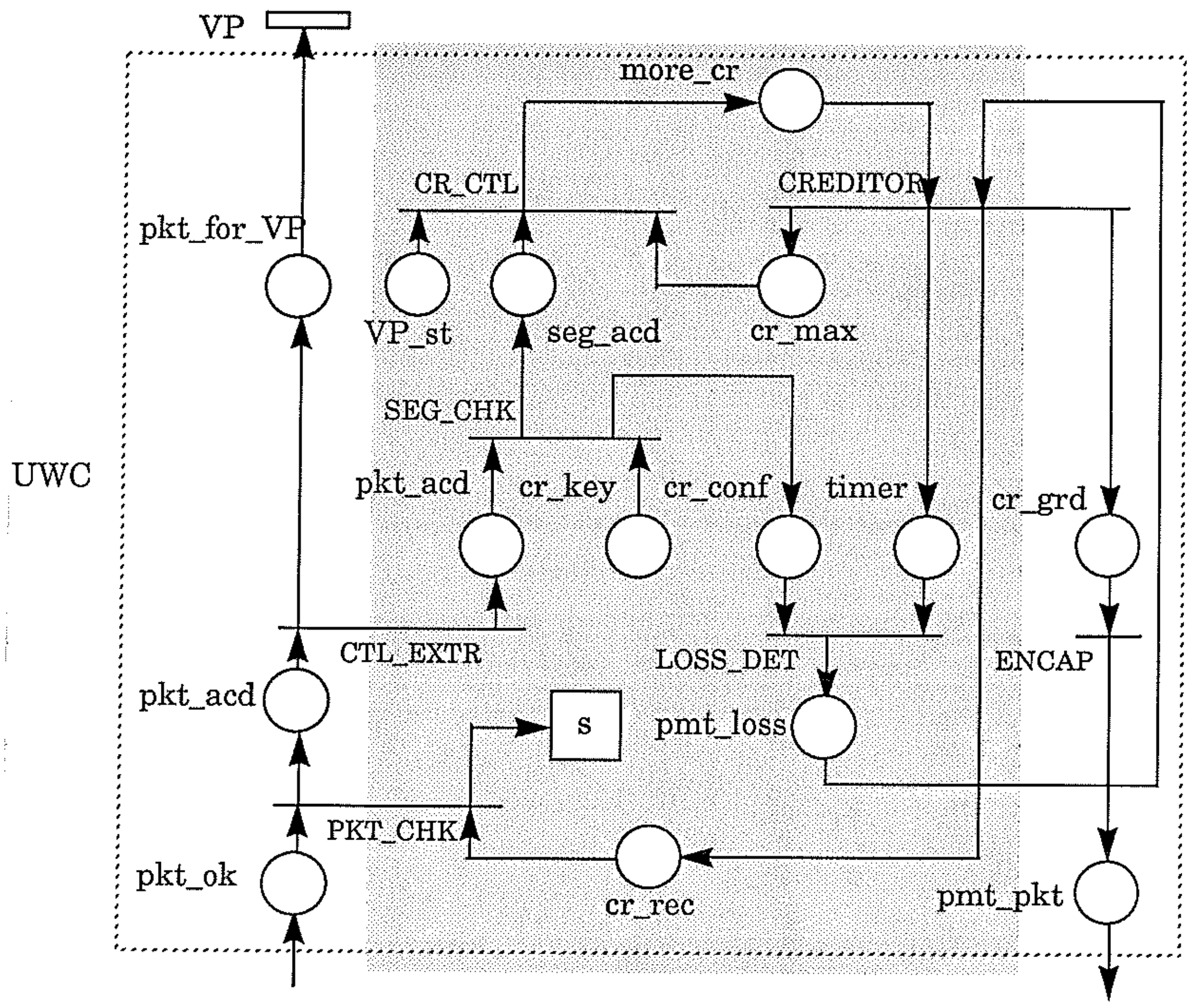

Figure 13: Window Flow Control Logic of UCC

As shown in Figure 13, a packet to the UWC logic from upstream should always be a data packet. The arrival of a data packet corresponds to a token at **pkt_ok** (packet error-checked). This token is subject to credit checking at PKT_CHK. If it belongs to a segment within a valid window slot, it is passed to **pkt_acd** (packet accepted). Otherwise the token is discarded in sink s. Transition

CTL_EXTR (control extraction) passes the data to VP and extracts the window information for credit control. At transition SEG_CHK (segment checking), the presence of packets in **pkt_acd** is mapped into the presence of segments in **seg_acd** (segment accepted) and the credit confirmation flag (in **cr_conf**) is set accordingly for permit retransmission control.

The decision on granting new credit is made at CR_CTL (credit control) according to three factors: the current status of VP in **VP_st**, the previous setting of credit in **cr_max** (maximum credit), and the current usage of the credit from **seg_acd**. If more credit should be issued, transition CREDITOR will update the record in **cr_max**, set the **cr_rec** for input packet checking, set the number of granted credits in **cr_grd** which will be sent to the upstream neighbor in a permit packet, and set the **timer** for permit loss detection.

A permit loss is detected at LOSS_DET if the time-out occurs before the reception of the permit by the upstream neighbor can be confirmed. A permit loss will be indicated in **pmt_loss**, which will trigger another firing of CREDITOR.

The down stream window control is much simpler. It needs only to keep track of the credit consumption and the new credit issued by the downstream neighbor. It will block if the credit runs out. The Petri net graph for the downstream window control is omitted.

The two-level flow control mechanism above sucessfully addresses the flow control issues raised in Section 3.2.3. First of all, the rate control ensures that the sources (VP in case of PTV) do not use more bandwidth than requested during connection setup. The next generation networks (e.g. ATM) require this of data sources in order to manage network resources effectively and to provide guaranteed services to applications. Such a performance guarantee from the network is essential to the success of PTV. However, rate control alone is not sufficient for PTV flow control for two reasons: (1) frequently adjusting the (inter)network connection rates to cope with speed fluctuation of a pipeline stage is not practical, because changing the data rate requires decision making by every packet switch and gateway on the connection path, which incurs significant delay to render the flow control ineffective and add latency to the pipeline; (2) despite its appropriateness for underlying networks, data rate is not a natural application level parameter for televisualization which deals with data in segments. Therefore, the window control on top is used to adjust data flow between two neighbor stages (VPs) to account for changes in the data consumption speed. Large data granularity for each window slot is used to achieve efficiency in networks with high bandwidth-delay product. Window update at the sending end is controlled by a special *permit* from the receiver, not by the ACKs as in a traditional sliding window protocol. Therefore, the flow control can be quickly activated by withholding new permits. The credit-based window control together with buffers distributed among pipeline stages prevents data overflow and minimizes delay to the pipeline in case of flow control activation. Since the window mechanism serves only the purpose of stage-to-stage flow control, it is simple and free of the interferences from both error and congestion conditions in the underlying network.

4.4. Application-Oriented Error Control

Error control has two aspects: (1) the detection of packet corruption, loss, and duplication; (2) the compensation for these error conditions. The goal for error control should be to satisfy the error tolerance of applications with minimum error control overhead. We approach this problem by using an application-oriented design. The application-oriented error control scheme for pipelined televisualization has four important aspects:

Timer at receiver. The timer for detecting data losses is located at the receiving end of a connection. The time-out interval is easier to determine for a timer at the receiving end because a more accurate estimate of one-way delay can be obtained than of a roundtrip delay [14]. More importantly, the receiver is best qualified to make retransmission decisions and thus should have control over the timer.

Selective retransmission. Segment streaming error control uses a selective acknowledgement scheme rather than the cumulative acknowledgement. A receiver saves correctly received data packets and requests retransmission for only the missing or corrupted parts. Therefore application processes can be allowed earlier access to partially received data, and extraneous retransmissions are avoided.

Retransmission strategies. In order to perform intelligent error control, we need to define a set of retransmission strategies. Retransmission strategy can be defined in three dimensions [55]: (1) granularity, which refers to how many missing packet events are accumulated before a request for retransmission is made; (2) fetch policy, which determines whether to always request for retransmission of a packet (*anticipatory retransmission*), or only to request retransmission when the data corresponding to the packet is referenced (*demand retransmission*); (3) preemption specifies whether to allow retransmissions to preempt the primary data stream (*preemptive*) or to make the retransmissions wait until all the data flowing in the primary stream is transmitted (*nonpreemptive*).

Application error tolerance. Since the main objective of the application-oriented error control is to satisfy an application's requirement using minimum retransmission, a set of application error tolerances needs to be defined which can be used by the error control mechanism to determine the optimal control strategy. Application error tolerance can also be described in a three dimensional space: (1) bit error tolerance (bit error rate, the minimum distance between two error bits); (2) packet loss tolerance (packet loss rate, minimum distance between two lost packets); (3) real time requirement (e.g. most recent data preferred, minimum animation speed).

The error control logic is described by two Petri net graphs in Figures 14–15.

The function of the upstream error control is to detect errors (bit error, packet duplicate, and packet loss) associated with the packet stream from the upstream stage and take appropriate actions (e.g. ignore, request for retransmission) according to the given error tolerance of the application. A Petri net graph for this logic is given in Figure 14.

An arriving packet is represented as a token in **pkt_recv** (packet received). Transition RECV_CTL (receive control) will pass the packet directly to **pkt_ok** if no bit error is detected. Duplicate packets are discarded. The **timer** is set at the stream setup according to the data and retransmission granularities. A packet loss is detected when time-out occurs before the corresponding packet is received. In case of bit error or packet loss, the retransmission request decision is made with respect to the error condition and the given error tolerance. If retransmission is needed, a token is produced to **nak** (negative acknowledgement). Transition MUX combines the NAK and the window permit into one token stream and passes them to the URC logic.

The downstream error control serves two purposes: detecting and discarding the corrupted packet (either a NAK or a window permit) from the downstream stage and processing retransmission requests according to the given application error tolerance. Figure 15 depicts this logic in a Petri net.

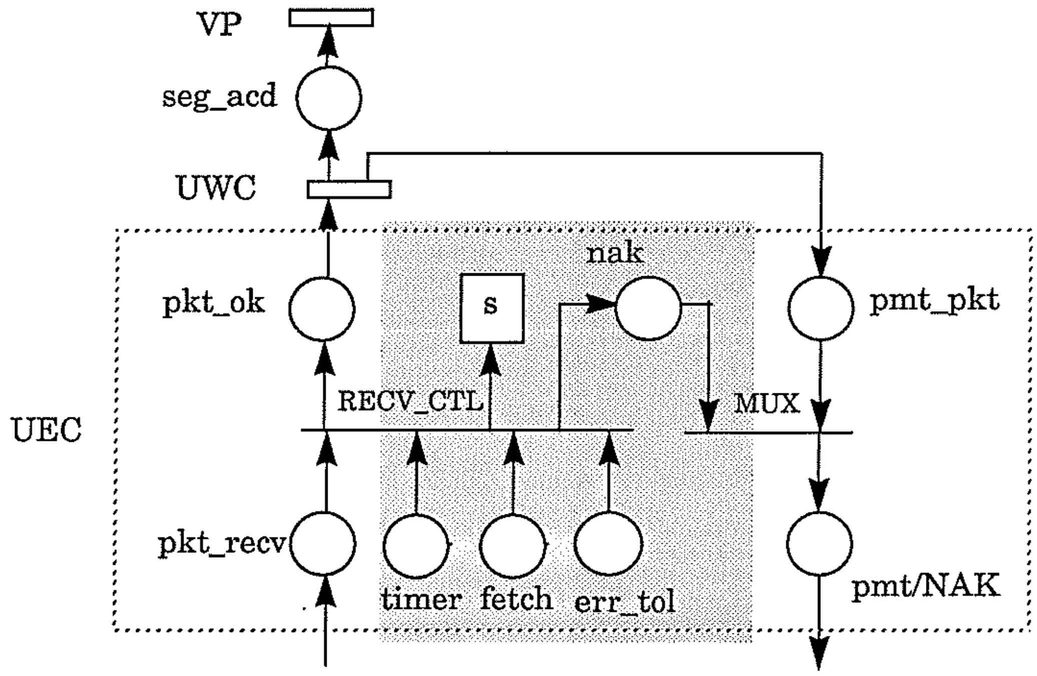

Figure 14: Error Control Logic of UCC

For a data packet that arrives from downstream, a token is generated in **pkt_recv**. Error checking is performed at RECV_CTL (receive control). If a bit error occurred, the packet is discarded into the sink s; if the packet contains a NAK but the tolerance indicates no need for retransmission the packet is also discarded; otherwise the packet is accepted into **pkt_acd**. Next, transition DEMUX produces a token to **nak** in case of a NAK packet and one to **pmt_pkt** if it is a permit packet. A NAK token causes one retransmission of the packet in error. The permit token will be interpreted by the DWC logic as described in Section 4.3.

The most significant feature of the proposed error control mechanism is its support for selecting different error control strategies according to application requirements. Therefore, error control can be optimized for the performance requirements of different applications (e.g. animation, real-time process monitoring).

It should be noted that only by separating error control function from flow control is it possible to use different error control strategies without adversely affecting the flow control function. Furthermore, by allowing the receivers to control timers and retransmissions, the proposed error control is less reliant on the accuracy of roundtrip delay estimate.

5. PLAN OF ACTION

The pipelined segment streaming is intended to be part of the Axon system. It also assumes support from Axon's host-network interface design, but a complete implementation of Axon cannot be expected in the time frame of this proposed work. In order to develop the ideas of segment

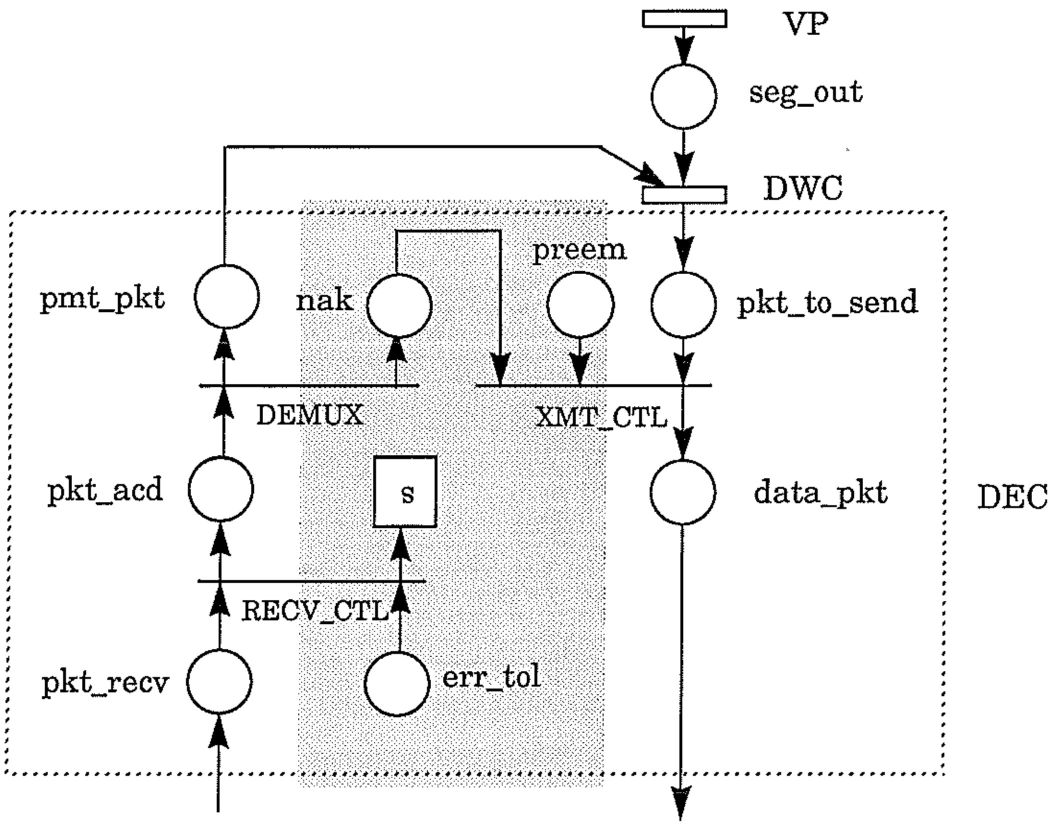

Figure 15: Error Control Logic of DCC

streaming with two-level flow control and intelligent error control and to demonstrate the viability of the PTV model, the following steps will be taken to conduct this research:

Identification and adaptation of target applications. Two target applications have been chosen for this study. The first, interactive visualization of cell development process, was introduced in Section 2.2.1. The second application is visualization of a 3-D scalar data set using the volume rendering technique [37, 58]. These applications have been selected because they represent applications with challenging computation and communication requirements, and they cover the two major rendering techniques (surface rendering and volume rendering) used for visualizing natural 3-D data.

Determination of the optimal error control strategies for target applications. This involves careful evaluation of the applications error tolerance with respect to different visualization objectives.

Design and simulation study. In the design of the segment streaming mechanism, two specific issues will have to be addressed: (1) determining the conditions under which two-level flow control and intelligent error control will be effective; (2) comparing the effectiveness of different flow control policies and error control strategies. This requires understanding the dynamics of the visualization application, flow control, and error control, as well as their interactions. Such complex and dynamic behavior makes it very difficult to apply an analytical approach. We therefore resort to discrete-event simulation for the design and analysis of the segment streaming mechanism [36, 31]. A simulator will be built on top of the Axon simulator which is under development in a related effort. The Petri net specification of segment streaming will be simulated to help understand these issues.

Experimental implementation of the segment streaming mechanism. The segment streaming (with two-level flow control and the application-oriented error control) will be implemented on top of existing IP (internet protocol). The campus network will provide the underlying network support. A model for the implementation is shown in Figure 16.

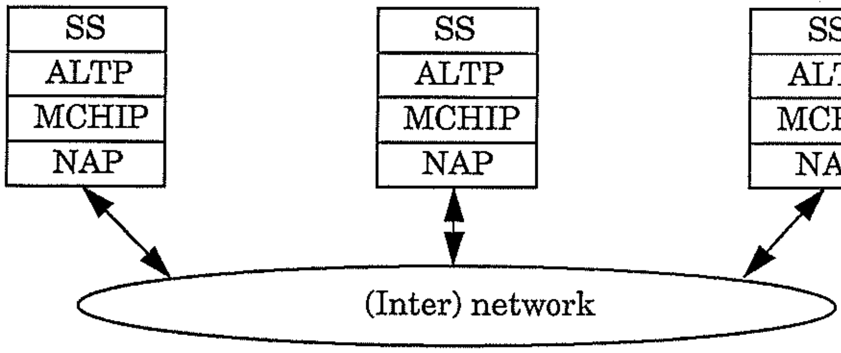

Figure 16: Segment Streaming Implementation Model

In this logical hierarchy of the protocols, the network access protocol (NAP) for Ethernet already exists. The internetwork protocol is a multi-point congram-oriented high performance internet protocol (MCHIP) which will be provided by a related research effort [43, 38]. A subset of the ALTP (application-oriented light weight transport protocol) will be implemented, which supports only the segment streaming operation with a two-level flow control mechanism and the appropriate error control mechanism. The high level segment streaming (SS) control includes functions such as prediction of the local processor load condition for flow control and a simple application interface to allow the invocation of the segment streaming operation.

Computing resources to be utilized in this experiment include several workstations, a Pixar image computer [46, 47], and a Titan graphics supercomputer [7, 17].

Implementation of the target applications. The two target applications will be implemented as PTV using the segment streaming support. Their performance will be studied using real measurements. The results will be used to verify those of the simulation study.

Characterization of suitable PTV applications. A set of rules will be developed which can be used to determine whether a visualization application is suitable for PTV implementation, given some parameters of the data and computation requirements as well as the available network support.

To better appreciate the issue of the viability of the PTV model (also the feasibility of pipelining across networks), we consider a visualization example using the volume rendering technique proposed in [37]. Assume the input data consists of 256 slices of 512 x 512 each. Each voxel is represented by 2 bytes. Simple analysis shows that about 100 arithmetic operations (addition or multiplication) are needed to project one voxel onto an image plane from an orthographic direction. On a single 50Mflops computer, projection of one slice will take $100 \times 512 \times 512/(50\text{Mflops}) \approx 500\text{ms}$. To apply the PTV model, we have partitioned the computation into 6 stages, with the longest stage requiring

23 operations/voxel. The per slice projection time, which is now determined by the longest stage, becomes approximately 115ms. This means that over 4-fold (100/23) speedup is possible with adequate IPC support. So what are the requirements of the IPC?

Let our goal be a modest 4-fold speedup. Then the cycle time of the pipeline should be no more than $25 \times 512 \times 512$flops/(50Mflops) $\approx$ 131ms. Due to data expansion, an equivalent of 5 data slices have to be passed between some pipeline stages. From Equation (12) in Section 3.2.2, we can show that a sustained effective data rate of 159Mbps is required, even with the assumption that communication completely overlap with intrastage computation (i.e. $t_{r_n} = t_{s_n} = 0$) and communication is 100% error-free. Without complete overlap between communication and computation, both t_{r_n} and t_{s_n} must be no more than $131 - 115 = 16$ms and a higher data rate is required. Moreover, networks are typically not 100% error-free and the error control overhead reduces the effective data rate. Finally, the pipeline throughput can be significantly less than 1 image update/131ms unless there is an effective flow control mechanism to avoid overflowing in the pipeline.

It is our belief that existing transport protocols and IPC mechanisms cannot meet such requirements of the application. It is my goal to develop an adequate IPC support at the transport level that will be able to satisfy these requirements of the PTV pipeline, therefore establishing the feasibility of pipelining across networks and the viability of the PTV model.

6. RELATED WORK

We know of no other efforts that attempt to develop special IPC mechanisms for efficient support of pipelined televisualization. There are, however, some efforts that are related to specific aspects of the proposed research. These are described in this section.

6.1. Televisualization Support

Existing network support for televisualization consists mostly of an efficient file transfer facility, typically used in batch or background mode. That is, the data generation step is performed first, then the data is shipped to a graphics workstation or minisupercomputer for visualization. This process does not support interactive visualization and often takes longer than necessary due to required human intervention.

Two approaches have been taken to make this process faster and more interactive. With the first approach, all visualization processing is performed on a supercomputer and images are then sent to display devices either as pixels or as NTSC (National Television Standards Committee) video signals [45, 4]. The transmission of pixels generally requires very high bandwidth, but results in higher image fidelity. The second approach transfers raw data between the supercomputer and the visualization facilities and requires higher bandwidth connections than NTSC analog signals. For example, data sets of a simulation can be transferred at high speed to a minisupercomputer that performs visualization computations [25, 62]. Neither of these are visualization solutions that every scientist can afford.

The Rivers (Research on Interactive Visual Environments) Project at NCSA (National Center for Supercomputer Applications) is developing hardware and software systems for supporting interactive steering of supercomputing by visualization [25]. This work would bring significant improvement to the NCSA supercomputing environment.

6.2. IPC Paradigms

There are two widely used IPC paradigms: remote procedure call (RPC) and message passing [2, 5, 19]. RPC derives its major advantage from its similarity to a conventional procedure call in programming languages. With RPC, programmers of distributed applications need not concern themselves with the details of managing communications with another address space or another machine, or with the details of the communication system in use. This forms a clear contrast with the message passing paradigm, in which application programmers are exposed to some of the lower level communication details. Message passing, however, allows applications to define more powerful IPC semantics to suit their needs.

It has been pointed out by many that RPC is not adequate for parallel distributed computing because its call semantics prevent concurrent communication [42]. There have been attempts to overcome this weakness by introducing more parallelism into the RPC semantics. Gifford and Glasser introduced a channel model which includes a pipe extension to RPC for passing back incremental results without call return, a provision for passing remote procedures as first-class values, and a synchronization scheme for maintaining relative sequencing of calls on pipes and procedures [23]. Satyanarayanan and Siegel extended RPC by allowing a process to make multiple RPC calls without being blocked [50]. These extensions, however, do not directly address the concurrent communication requirement in a televisualization pipeline.

Message passing supports only the basic message sending and receiving semantics. Any more complex semantics such as flow control will have to be supported by building additional mechanisms on the basic message passing mechanism. Furthermore, most existing message passing support is not designed to deal efficiently with large granularity data as needed in PTV applications.

6.3. Flow Control and Error Control

The two most significant flow control mechanisms are sliding window flow control [6, 57] and rate-based flow control [14, 55, 43]. They have been discussed in Section 3.2.3 and Section 4.3 respectively. The proposed two-level flow control attempts to combine the strong points of the window-based and the rate-based flow control approaches to best serve the need of pipelined televisualization.

Many methods for error control have been discussed in standard networking textbooks [6, 57]. Major disadvantages with the traditional error control method were pointed out in Section 3.2.3 and the proposed application-oriented error control has been presented in Section 4.4.

6.4. Host-Network Interface

There have been several recent high performance host-network interface architecture projects. The NAB (network adaptor board) [30] is a custom host-interface designed to support VMTP [11]. Another approach to the performance problem is to implement existing transport protocol mechanisms in hardware, as in XTP (express transport protocol) and the PE (protocol engine) [12, 13]. The Nectar CAB (communication accelerator board) [3] provides a workstation-network interface of 10MBps, and avoids a store and forward hop when CAB memory is mapped into the host address space. The CAB is connected to the host through a VME bus port.

Finally, the Axon network interface [52] (described in Section 4.1) emphasizes the integral design of host architecture, protocols, and operating systems, as well as the systematic evaluation of the division of functionality between hardware and software.

It should be noted that an important aspect of new host-network interface designs is to provide a high bandwidth memory-to-memory data path. This proposed research addresses the issue of how to make efficient use of the data path for achieving high application performance.

7. EXPECTED CONTRIBUTIONS

The proposed work will be considered complete once the tasks outlined in Section 5 have been performed.

Major contributions expected of this research can be summarized as follows:

- demonstration of the feasibility and usefulness of extending the pipeline principle to a network environment
- demonstration of the PTV model as a viable approach for implementing some demanding visualization applications
- development and understanding of the two-level flow control concept for PTV applications
- understanding of the target application error tolerances and the determination of optimal error control strategies

Additionally, since this research addresses several issues within the Axon architecture, its results will have implications on the Axon architecture. Through the development of the network support for PTV applications, further understanding will be gained about the general approach of application-oriented protocol design.

References

[1] Akhtar, Shahid, *Congestion Control in a Fast Packet Switching Network*, Wash. U. CS Dept., M.S. thesis, St. Louis, Dec. 1987.

[2] Andrews, Gregory R. and Fred B. Schneider, "Concepts and Notations for Concurrent Programming", in *Concurrent Programming*, Narian Gehani and Andrew D. McGettrick, eds., Addison-Wesley, Wokingham, England, Mass, 1988, pp. 3–69.

[3] Arnould, Emmaneul, et al., "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers", *ASPLOS-III (ACM SIGOPS OS Rev.)*, Vol. 23, ACM, New York, April 1989, pp. 205–216.

[4] Bancroft, Gordon V., et al., "Scientific Visualization in Computational Aerodynamics at NASA Ames Research Center", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 89–95.

[5] Bal, Henri E., et al., "Programming Languages for Distributed Computing Systems", *ACM Computing Surveys: special issue on programming language paradigms*, Vol. 21, No. 3, Sept. 1989, pp. 261–322.

[6] Bertsekas, Dimitri and Robert Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[7] Borden, Bruce S., "Graphics Processing on a Graphics Supercomputer", *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, July 1989, pp. 56–62.

[8] Bovopoulos, Andreas D. and Einir Valdimarsson, "Performance Evaluation Of A User Network Interface For ATM Networks", 1990.

[9] Carriero, Nicholas, and David Gelernter, "How to Write Parallel Programs: A Guide to the Perplexed", *ACM Compt. Surv.*, Vol. 21, No. 3, Sept. 1989, pp.323–358.

[10] Chen, T. C., "Parallelism, Pipelining and Computer Efficiency", *Computer Design*, January 1971, pp. 69–74.

[11] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol. 16, No. 3, ACM, New York, 1986, pp. 406–415.

[12] Chesson, Greg, "Protocol Engine Design", *Proceeding of the Usenix Conference*, 1986.

[13] Chesson, Greg, et al., "XTP Protocol Definition", Revision 3.1, Protocol Engines, Inc., PEI 88–13, Santa Barbara, Calif., 1988.

[14] Clark, David D., Mark L. Lambert, and LiXia Zhang, "NETBLT: A High Throughput Transport Protocol", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol. 17, No. 5, ACM, New York, 1987, pp. 353–359.

[15] Dasgupta, Subrata, *Computer Architecture A modern synthesis, Volume 2: Advanced Topics*, John Wiley & Sons 1989.

[16] DeFanti, Thomas A., et al., "Visualization : Expanding Scientific and Engineering Research Opportunities", *Computer* , Vol. 22, No. 8, Aug. 1989, pp. 12–25.

[17] Diede, T., et al., "The Titan Graphics Supercomputer Architecture ", Vol. 21, No. 9, *Computer*, Sept. 1988, pp.13–30.

[18] Dubey, Pradeep K. and Michael J. Flynn, "Optimal Pipelining", *Journal of Parallel and Distributed Computing*, Vol. 8, No. 1, January, 1990, pp. 10–19.

[19] Filman, Robert E. and Daniel P. Friedman, *Coordinated Computing: Tools and Techniques for Distributed Software*, McGraw-Hill, New York, 1984.

[20] Gajaski, Daniel, et al., "CEDAR", in Tutorial, *Supercomputers: Design and applications*, edited by Kai Hwang, Computer Society Press 1884, pp. 251–275.

[21] Gajaski, D. D., and J-K Peir, "Essential Issues in Multiprocessor Systems", *Computer* , Vol. 18, No. 6, 1985, pp. 9–28.

[22] Giacopelli, J. N., et al., "Sunshine: A High Performance Self-Routing Broadband Packet Switch Architecture", 1989, pp. 1–22.

[23] Gifford, David K., and Nathan Glasser, "Remote Pipes and Procedures for Efficient Distributed Communication", *ACM Trans. Comput. Syst.*, Vol. 6, No. 3, August 1988, pp. 258–283.

[24] Hayes, John P., *Computer Architecture and Organization*, Second Edition, McGraw-Hill 1988.

[25] Haber, Robert B., "Scientific Visualization and the Rivers Project at the Center for Supercomputing Applications", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 84–89.

[26] Helman, James and Lambertas Hesselink, "Representation and Display of Vector Field Topology in Fluid Flow Data Sets", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 27–36.

[27] Hwang, Kai and Faye' A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill 1984.

[28] Hwang, K., and Z. Xu, "Multipipeline Networking for Compound Vector Processing", *IEEE Trans. Computers*, Vol. 37, No. 1, January 1988, pp. 33–47.

[29] Jameson, Antony, "Computational Aerodynamics for Aircraft Design" *Science*, Vol. 245, No. 4916, 28 July 1989, pp. 361–371.

[30] Kanakia, Hemant and David R. Cheriton, "The VMP Network Adaptor Board (NAB): High Performance Network Communication for Multiprocessors", *SIGCOMM'88 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol. 18, No. 4, ACM, New York, 1988, pp. 175–187.

[31] Kobayashi, H., *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*, Addison-Wesley Publishing Company, Inc. 1981.

[32] Kogge, Peter M., *The Architecture of Pipelined Computers*, Hemisphere Publishing Corporation 1981.

[33] Kung, H.T., "Why Systolic Architecture?", *Computer*, Vol. 15, No. 1, January 1982, pp. 37–46.

[34] Kung, H.T., "The CMU Warp Processor", in *Supercomputers Algorithms, Architectures, and Scientific Computation*, (eds.) F.A. Matsen and T. Tajima, University of Texas Press, Austin 1986, pp. 236–247.

[35] Kung, S.Y. et al., "Wavefront Array Processor-Concept to Implementation", *Computer*, Vol. 20, No. 7, July 1987, pp. 18–33.

[36] Law, Averill M. and W. David Kelton, *Simulation Modeling and Analysis*, McGraw-Hill Inc. 1982.

[37] Levoy, Marc, "Display of Surfaces from Volume Data", *Computer Graphics and Applications*, Vol. 8, No. 3, May 1988, pp. 29–37.

[38] Mazraani, Tony Y. and Gurudatta M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, abridged from: Washington University Department of Computer Science, technical report WUCS-89-20, St. Louis, Aug. 1989.

[39] McCormick, Bruce H., et al., (eds.), "Visualization in Scientific Computing", *Computer Graphics*, Vol. 21, No. 6, November 1987.

[40] Nielson, Gregory M. and Bruce D. Shriver (eds.), *Scientific Visualization bringing data into focus*, special issue, Computer, Vol. 22, No. 8, Aug. 1989.

[41] Noe, Jerre D., and Gary J. Nutt, Macro *E*-Nets for Representation of Parallel Systems", *IEEE Transactions on Computers*, Vol. 22, No. 8, August 1973, pp. 718–727.

[42] Partridge, Craig (ed), *Proceedings of Internet Research Steering Group (IRSG) Workshop on Architectures for Very-High-Speed Networks*, January 24–26, 1990, Cambridge, Massachusetts.

[43] Parulkar, Gurudatta M., "The Next Generation of Internetworking" , *Computer Communication Review*, Vol. 20, No. 1, ACM SIGCOMM, New York, Jan. 1990, pp. 18–43, also: Washington University Department of Computer Science, technical report WUCS-89-19, St Louis, May 1989.

[44] Peterson, James L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall 1981, Englewood Cliffs, New Jersey.

[45] Phillips, Richard L., "Distributed Visualization at Los Alamos National Laboratory", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 70–77.

[46] *Pixar Image Computer Programmer's Manual*, Pixar 1987.

[47] *Pixar Image Computer Application SW USER'S GUIDE*, Pixar 1987.

[48] Ramakrishnan, K. K., and Raj Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer", *SIGCOMM '88 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.18, No.4, ACM, New York, 1988, pp.303–313.

[49] Rosenblum, Lawrence J., "Visualization of Experimental Data at the Naval Research Laboratory", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 95–101.

[50] Satyanarayanan, M, and Siegel EH, "Parallel communication in a large distributed environment", *IEEE Trans. Computers*, Vol. 39, No 3, 1990, pp. 328–348.

[51] Stallings, William, *Data and Computer Communications*, second edition, Macmillan Publishing Company 1988.

[52] Sterbenz, James P. G., *Axon: Host-network interface Design*, Washington University Computer Science Department, technical report WUCS-90-7, St. Louis, March 1989.

[53] Sterbenz, James P. G., Gurudatta M. Parulkar, "Axon: Network Virtual Storage Design", *Computer Communication Review*, Vol. 20, No. 2, ACM SIGCOMM, New York, April 1990, pp. 50–56, abridged from: Washington University Computer Science Department, technical report WUCS-89-13, St. Louis, May 1989.

[54] Sterbenz, James P. G., Gurudatta M. Parulkar, "Axon: A High Speed Communication Architecture for Distributed Applications", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, pp. 484–492, also: Washington University Computer Science Department, technical report WUCS-89-36, St. Louis, September 1989.

[55] Sterbenz, James P. G., Gurudatta M. Parulkar, *Axon: Application-oriented Lightweight Transport Protocol Design*, Washington University Computer Science Department, technical report WUCS-89-14, St. Louis, September 1989.

[56] Suzuki, Hiroshi, et al., "Output-buffer Switch Architecture for Asynchronous Transfer Mode", *Conf. Rec. Int. Conf. Commun. Vol. 1 (of 3), IEEE International Conference on Communications (ICC'89)*, Boston, MA, USA, June 11–14 1989, pp. 99–103.

[57] Tanenbaum, Andrew S., *Computer Networks*, Prentice-Hall 1988.

[58] Tiede, Ulf, et al., "Investigation of Medical 3D-Rendering Algorithms", *Computer Graphics and Applications*, Vol. 10, No. 2, March 1990, pp. 41–53.

[59] Turner, Johnathan S., "Design of a Broadcast Packet Switching Network", *IEEE Transactions on Communication*, Vol.36, No. 6, New York, June 1988, pp. 734–743.

[60] Upson, Craig, et al., "The Application Visualization System: A Computational Environment for Scientific Visualization", *Computer Graphics and Applications*, Vol. Vol. 9, No. 4, July 1989, pp. 30–42.

[61] Winkler, Karl-Heinz A., et al., "On the Characteristics of a Numerical Fluid Dynamics Simulator", in *Supercomputers Algorithms, Architectures, and Scientific Computation*, (eds.) F.A. Matsen and T. Tajima, University of Texas Press, Austin 1986, pp. 416–429.

[62] Winkler, Karl-Heinz A., et al., "A Numerical Laboratory", *Physics Today*, Vol. 40, No. 10, October 1987, pp. 28–37.