3-D VISUALIZATION OF MESSAGE PASSING IN DISTRIBUTED PROGRAMS

by

Cyril Gobrecht, Colin Ware and Virendrakumar C. Bhavsar

TR96-103, January 1996

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

3-D Visualization of Message Passing in Distributed Programs

Cyril Gobrecht, Colin Ware and Virendrakumar C. Bhavsar
Faculty of Computer Science
University of New Bunswick
Box 4400, Fredericton, New Brunswick
CANADA E3B 5A3

phone: (506) 453-4566
fax: (506) 453-3566

email: cware@unb.ca
Contact person: Colin Ware

Abstract

This paper describes PVMtrace, a software system for understanding and debugging message passing in distributed programs. In this system each process is represented as a node in a graph and the arcs represent the potential communication channels. The transfer of a packet of information from one process to another is shown by a bead-like object moving along an arc. A queue is a string of beads lined up waiting to be processed. The flexible control over time is found to be essential for the system to be useful and it allows the animation to be played forward and backwards in time both by the direct manipulation of the time line and by an animation rate controller. In addition there is an automatic time control method that rapidly moves the animation through intervals of low activity while slowing down in periods of high activity. The system was initially developed as a PVM debugging tool but it has also been found to be useful in other areas. In particular it has been used to visualize communications in a cellular phone system.

1. 0 Introduction

Distributed and parallel programs, running on different processors and exchanging or sharing data, are notoriously difficult to understand and therefore difficult to debug. Apart from understanding the sequential parts of the distributed program, the most critical issue is gaining

an overall understanding of its dynamic behaviour, in particular, the interaction between distributed processes with respect to synchronization and communication. One approach that is widely used to help understand these systems is to use visualization wherein various kinds of charts and graphs are used to represent different aspects of program execution. There are three main aspects to the visualization of distributed programs: (a) the graphical representation of the set of processors and processes, (b) the visualization of message passing between processors and processes, and (c) the visualization of system utilization. We deal primarily with (a) and (b) here, and in the following sections we describe some of the novel animation techniques that we have developed.

Many commercial and research tools use graphs for the representation of the set of processors and processes. In processor graphs nodes are used to denote processors and arcs for physical connections between processors; in process graphs, nodes are used to represent processes and arcs are used to represent call relationships and temporal orderings.

Most of the systems reported in literature use a *two-dimensional* (2D) display of process and processor graphs and therefore they are limited in terms of the display of large graphs [1]. For example, ParaGraph provides a 2D view where the processes are displayed either in a circle or in process-time diagram. However there is evidence that 3D viewing may be beneficial; Ware and Franck [11] found that more than 300 nodes and 500 arcs could be perceived if the structures were rotated and displayed in stereo with a 3D layout; this is approximately three times as many as could be displayed in 2D. Moreover, the architectures used for parallel/distributed machines and programs have 3D and higher dimensional structures (e.g. cube, hypercube, toroid, pyramid and cube-connected cycles). Thus 3D display of process/processor graphs is much more desirable than a 2D display.

The process-time diagram is the most frequently used graphical method of presenting message passing between processes (this is also known as a message sequence chart). In such a diagram, time is displayed on the x-axis and the different processes are each represented by a horizontal bar[1,7]. Figure 1 illustrates this kind of diagram. Three dimensional versions of this kind of diagram have also been used[4,8]. In these systems, two of the dimensions to represent the structure of the graph laid out on a plane and the third is used to represent the trace history in the form of a process-time diagram. The advantage of the process-time diagram is that it shows an extended period of time in a single representation and it is easy to see the relative times of arrival and departure at a particular node. However, there are a number of disadvantages of both the two-dimensional and three dimensional versions. Such diagrams

can easily become cluttered if the number of nodes and messages is large; the many crossing lines become confusing and ambiguous. Moreover, the addition of information about the behavior of structures such as queues and message types is difficult or impossible with this type of diagram.

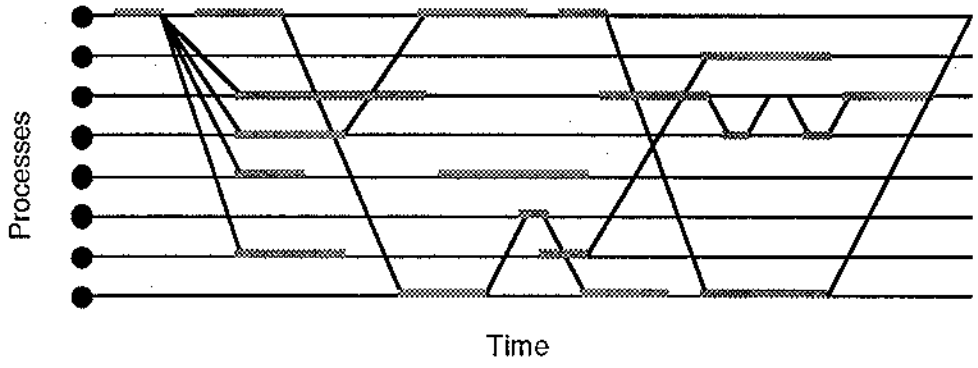


Figure 1. The process-time diagram wherein a horizontal line represents a process and the messages are illustrated by the oblique lines connecting the processes. The thick lines represent processor activity. Time is represented on the x-axis.

Some other systems have relied on animation to show message transmission. The exchange of data in Paragraph [3] is represented by lines connecting the processes. When the representation is running, these lines appear and dissappear to show where and when data exchanges are occurring in the program. The dynamic change of colors of lines has also been used to show traffic flow [3]. Another example of animation is a system described by Kahn and Saraswat [5] in which message passing is represented by placing a line "between the message and the head of a channel." This "eventually shrinks to zero length bringing the message and its recipient together by either moving the message, its recipient, or both".

A recent system that uses the animation of discrete messages is the Conch system described in Topol et al. [10]. In this system processes are arranged around the circumference of a circle while messages are represented by dots that migrate first towards the center of the circle and then towards their target process. One difficulty with this system is the difficulty of seeing which processes are communicating with each other, because the path is broken in the middle. This method clearly shows when messages are not read. However because of the broken path it does not allow the user to easily follow individual messages. The animation of individual messages is an idea which we have developed further in the PVMtrace system.

In this paper we present PVMtrace, a system that makes use of animation to reveal more details of message passing. This system has a number of novel features. We use three dimensions to layout the process graph, allowing for the representation of large and complex systems. We use animated tokens passing between processes to represent messages. Thus unlike the process-time diagram that maps time to space we map time to time. However, since message animation can only give an instantaneous, albeit evolving picture of the system, dynamic control of the animation speed is provided in our system, indeed it is essential. This allows the user to gain an overview by playing the message sequence rapidly or to scrutinize detailed behaviour by skipping to an interesting episode and slowing down the animation. In an automatic mode the rate of animation is controlled by means of an algorithm that is designed to emphasize the interesting episodes.

We have chosen the Parallel Virtual Machine (PVM) as a vehicle for investigating ideas relating to distributed program visualization, both because it is a robust system and because it provides an effective method for instrumenting a running PVM program [2].

The remainder of this paper is organized as follows: First we briefly describe how PVM programs have been instrumented and discuss some of the problems of temporal ordering of messages. Next, we describe the methods we use for the visualization and animation of process graphs and message passing. Following this we describe methods used for animation speed control. We conclude with a description of how PVMtrace has been used in practice and a summary of the relative benefits of this approach.

2. Instrumentation

The first step in visualizing a distributed program is to obtain and store the appropriate information about the program. This section describes how we instrument a PVM-program and how the data received from the program is then organized in a structural manner. Although we specifically describe the methodology in the context of PVM it is applicable to other distributed programs.

A PVM-program is composed of multiple processes which may run on different processors. Usually, a first process is run, which then starts other processes. Once a process is started, it has to join the PVM machine. This is done by using the command pvm_mytid, which creates a communication link between the process and PVM. The child process can then get some

information about its parent (the process id which started it), the other processes and the configuration of the PVM machine. It can also send messages, check the queue of messages and of course receive messages.

Sending and receiving messages is probably the most important part of the system. Sending messages is done by grouping data in packets before sending them. It is possible to send a message to either one or many processes. Each one of these messages has a tag and the sender tid which can help the receiver select messages in its incoming queue. In other words, the queue is not strictly first in first out, look ahead and fetching messages out of order are both allowed.

2.1 Data Retrieval Method

PVMtrace, the visualizing tool, is a PVM-process. When it is started, it joins PVM by using the pvm_mytid command. Then, it starts the first process of the distributed program by using the pvm_spawn command.

A PVM process has the option to get information about its children and all their descendants. Before PVMtrace starts the first process, options are set to require this process and all its descendants to send a record of all message passing and process creation activities to PVMtrace. Once this is done, each time one of the processes in the distributed program issues a command (that we chose to monitor) it will send a message to PVMtrace describing which process issued the command, the name of the command, the time it has been sent , the arguments used and in some cases the resulting values. This data is then stored inside PVMtrace as events in order to do the visualization. In addition, PVMtrace also asks the distributed program to send messages containing the output normally directed to the screen. This allows PVMtrace to output the messages to the local screen instead of having them displayed on each individual computer where the processes are running.

There are four main states of a PVM process: running, waiting, not yet started, and terminated. When it is running it is either in the state of sending or receiving a message or it is doing some computation. When it is waiting it is because a certain message has not yet arrived. When it is idle, it is either because it has not yet started, but it virtually exists because it has been spawned. When it is terminated it is finished either by quitting PVM normally or because it has

been killed. PVMtrace can determine this state information because the processes can be programmed to send it a message any time their state changes.

2.2 Time Related Problems

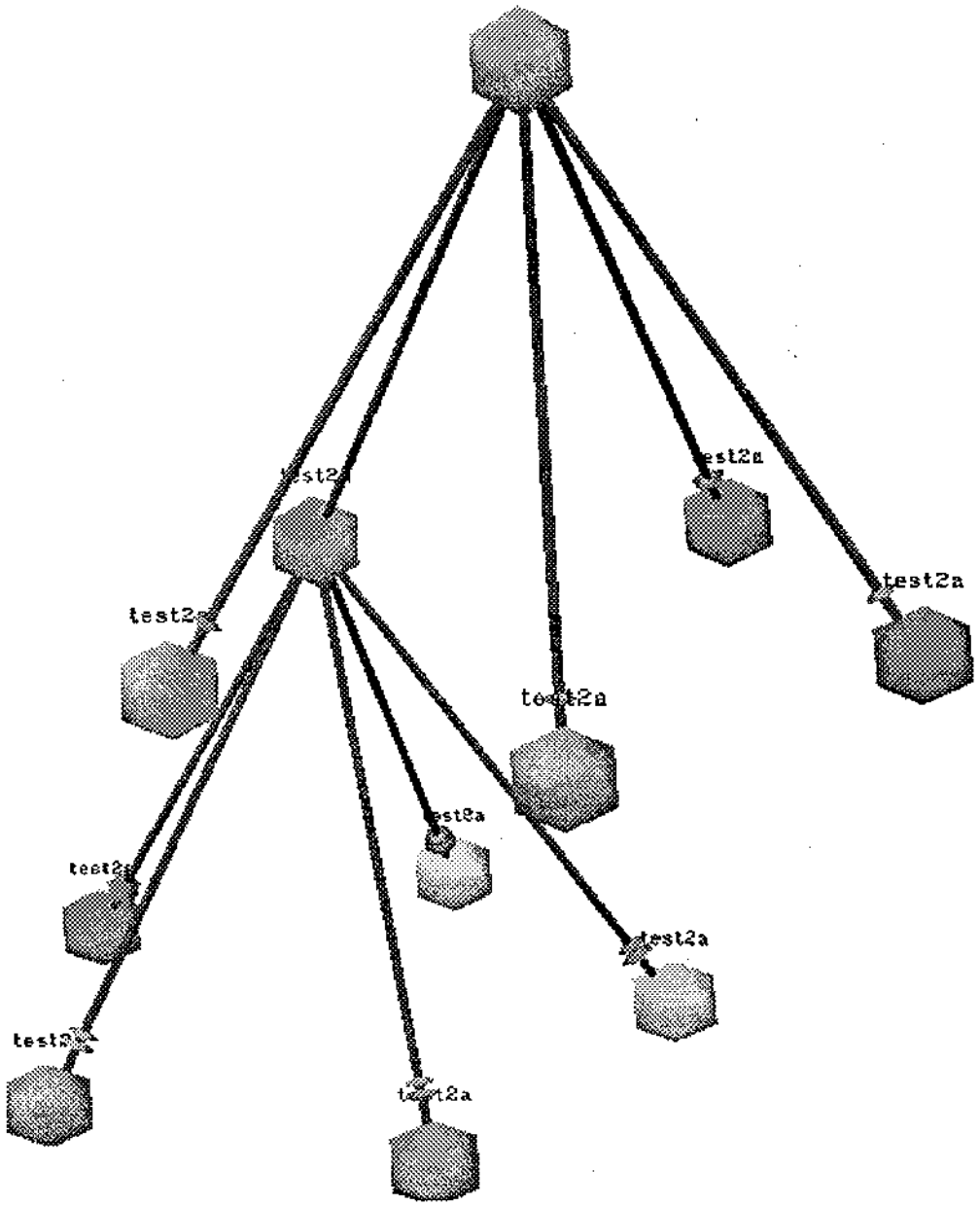
There are two major problems involved in the active monitoring of parallel programs . The first of these is that the monitoring program itself tends to distort the temporal behavior of the program. Not only will it slow execution on the host machine, but it will also slow down the network by adding messages. While this problem is real the effect of it may be no greater than the effect of an additional user logging on the host system and we believe that our programs are robust against such occurrences.

A more critical problem is that it is not possible to exactly determine the arrival times of messages when they are running on different machines and where only time of departure information is available. Moreover, it is not possible to know with certainty which of two events occurred first between two processors because it is not possible to precisely estimate the differences between the clocks on different machines.

Because of these time problems which insert an uncertainty in the order the events, the events are put in different ordered lists, one for each processor. Within each one of these lists, we are certain of the order of the events and we also are sure that no events have occurred concurrently. For each list, a pointer is kept on the next event that has to be handled in the animation. PVMtrace also stores a time correction value for each processor. For an extensive discussion of time related issues see Kraemner and Stasko [6,7].

To select which command has to be handled first, PVMtrace compares the times of the next command on each host adjusted by the correction factor. To initially set the values of the time difference between the PVMtrace host and another, the PVMtrace host asks the other, what time it is. This time is then compared to the time the request was sent and the time the answer was received. However, because the communication lags are variable this only gives a lower value and an upper value of the time difference. By repeating this process, we can expect to reduce the difference between the lower and upper values. Nevertheless, in some cases, because of the errors in the time correction events may be out

of order. To correct this problem, PVMtrace does automatic adjustment of the time correction values. Each time it encounters a message read that occurs before the message is in the queue, it adjusts the time differences of the processor on which the reading occurred. For example, PVM trace may receive information that process A on processor 1 has read a message from process B at time t while there is no corresponding message (correct sender, size and tag) in the input queue for process A. In this case, PVMtrace looks ahead to find the correct message and uses this to adjust the time correction constant.



Figurer 2. A cone tree layout of rocesses, communication channels and messages in PVMtrace.

PVMtrace can be run either on-line or off-line. When it is run on line, unless the number of messages is very small relative to the amount of computation, PVM will be far ahead of the animation and thus all the data relating to message passing and program state will have been stored. When it is run off line, the data is read from a trace file. From the point of view of the animation, time is incremented at some rate that is convenient to the viewer. In both cases we allow the user to go backward and forward in time using the stored data.

3. Visualization

This section describes how the graphical 3D representation of the PVM-program is done. It is important to note that in distinction to most other systems, the visualization is done by mapping time to time and space to space and not time to one space-dimension and the processors to a second dimension as in the case of the process-time diagram. The implications of this will be explored in the following sub-sections

The implementation of PVMtrace has been done on Silicon Graphics workstations so as to be able to use the fast 3D graphics available on these types of machines. PVMtrace is built using the GL graphics library for the 3D graphics with Motif for the menus, the widgets and the mouse events. For programming in Motif, the ViewKit library has been used. This library has been written in C++ and considers the widgets or group of widgets as C++ objects.

3.1 Processes

Each PVM program is represented by a graph where the processes are the nodes and potential communication channels are the arcs. The nodes which represent the processes are drawn in 3D as sphere-shaped objects composed of 20 triangles (Figure 2). There are two options for color coding the processes. In one mode color is used represent the state of the processes: waiting processes are represented by a red color, terminated processes are represented in black while not yet started processes are in white. When a process is sending or receiving it is yellow, when it is running (but not sending or receiving) it is green.

In another mode which can be selected from the menu, the processes are colored according to the processor they are running on. Each processor gets a color assigned (or more precisely a

hue in the HSV color system). By modifying the value and saturation, each process on the same processor gets a slightly different color, which can be related to the colors in the time-line (described in a later section). For example, all the processes from one processor will be in blue, with different saturation values. Each node is labelled with the name of the program it is running. When the user clicks on the node, the PVM process id appears under the name. When the whole scene is rotating, the name still appears facing the screen and on top of the node.

3.2 Process Layout

The layout method used in PVMtrace is the cone tree [9]. Cone trees are a way to display a tree structure in 3D. The parent node forms the apex of the cone and the children are displayed in a circle underneath it forming the base. In cone trees, the further the nodes are from the root, the smaller the base of the cone will be. This is used to avoid intersection between cones. We also make the size of the base depend on the number of child nodes. Figure 2 gives an example.

In PVMtrace, the tree structure that determines the layout represents process ancestry. The top node represents the process that has been started by PVMtrace. All the nodes that are then spawned by this process will form the base of the first cone. These processes might spawn other processes which will form new cones underneath. This automatic way of laying out the nodes seems to be effective in many cases. In so far as communication are mostly between parents and children this layout tends to reduce the number of arc crossings which tend to confuse the overall stucture. Clearly some other ways of displaying the nodes might be more suitable for certain parallel programs which have a particular architecture. For instance hypercube algorithms would probably be better displayed with a layout specifically designed to reflect this structure.

3.3 Links

A link between two processes is only defined if and when a message is sent between them. If no messages are sent, no link is defined. The purpose of connections defined in this way is solely to show where there is an exchange of data and where not. It does not relate to the physical network structure. By keeping track of the traffic in both directions, PVMtrace can show how much data is exchanged between the processes.

The links are represented as solid rods connecting the nodes (Figure 2). Because PVMtrace uses a lighting model this makes the link orientation easier to see. The links are very helpful to understand the structure of the tree. When there is more than one level in the cone tree, it is hard to find which nodes belong to which cone without the visible links.

When using PVMtrace, there are two optional color coding schmes for the links . In both the color changes with the number of messages going along the link but in different ways.

With one option, the color shows the accumulated number of messages through a color code. The color varies from purple to red (from a cold to a warm color) and is related to the number of messages through a logarithmic function, to make colors change a lot with very few messages and still change for thousands of messages. Each time a message is sent between two nodes, the color of the link between them is modified.

With the other option, color is used to show the current activity level. Each message passing along the link heats it up. The link then slowly cools. The colors in this mode vary between black (which is the same color as the background) and red. The 'warmer' the link is, the more red is used in its representation. This mode shows the activity of a certain link, at a certain time of the animation. This mode of viewing has the advantage that it tends to declutter areas of the diagram where there is little activity because most of the dormant arcs will have faded into the background.

3.4 Messages

The messages are represented as rings around the links and resemble beads. They remain visible from the time they are sent to the time they are read. The messages can be in one of three states at the time of drawing. Initially a message will be in a moving state, when it is travelling from one node to another. Then it may stay in a queued state waiting to be read, finally, it will be in the state of being read. Although these states may represent very short time intervals in terms of the actual PVM program the animation of the states and state transitions over an extended time is essential to program comprehension. It is only because the link takes a significant amount of time to travel between two nodes (in the animation) that makes it possible for the user to see that a message has passed and to visualize the

direction of communication. We have put a considerable effort into making this animation effective from a number of standpoints.

PVMtrace allows for program animation at various rates from very slow to quite fast. It is important that communication events can be seen at these different rates. The number of frames PVMtrace uses to represent a transmitted message depends on the speed of the animation, not on the time the event took in PVM (which we do not know). When the messages are moving, they are represented as a bead with the trailing edge stretched out into a tail giving the impression of speed. The faster the message is moving, the longer the tail is drawn, however the message is drawn in fewer frames (Figure 3). When the speed of the animation is very fast the message may be represented only by a single very long bead that stretches the entire length of the communication link for a single frame. In fact this single bead may represent one or more messages in this interval. All you see in this case is quick flashes traveling from one point to the other, however the direction is still clear. When the animation is slowed down, the messages will take more time to go from one point to another and each individual message is clearly distinguishable.

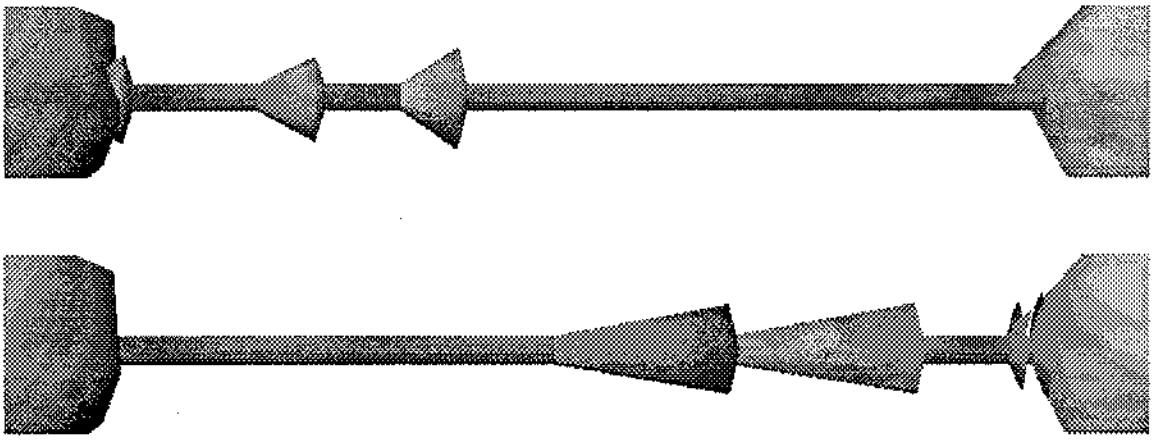


Figure 3: Messages moving from left to right along a link (slow and fast)

3.5 Message Contents

The more data a message contains, the bigger it is drawn. To limit the size of the message and to make differences between one byte and two bytes messages visible, the relation between the size of the drawn message and the size of the real message is chosen to be logarithmic. The color used to draw a message also gives some information about the

content. The main problem is that the content may be heterogeneous. The data inside can be composed of different types (depending on what data was packed in before sending). For the color choice, the data is divided in three categories. The first category represented in blue is for values of type char and strings. The second category (green) contains values of type integer, short or long as well as unsigned integer and unsigned short. Finally, the third category (red) contains float, double and complex types. The three colors are mixed depending on the amount of data of each type contained in the message. For example a message with as many chars as floats will be drawn in purple (blue and red). In most cases this color scheme gives a good idea of the type of message it is because most messages contain only one type of data, not several.

PVMtrace allows the user to get more information about the message by clicking on it. When this is done little flag appears linked to the message. This flag shows the detailed structure of the message using color bars (using the same color coding as for the message itself). Each line in the flag represents an array and the length of the line represents the size of the array. To avoid too long lines, arrays are changed into squared blocks when the array has more than ten elements. The components of the message are shown in the data flag in the same order as defined in the message. For example, Figure 4 shows a node, a link and a message in the waiting queue. The flag attached to the message shows (on a color display) that the message contains a floating point value and an array of seven integers.

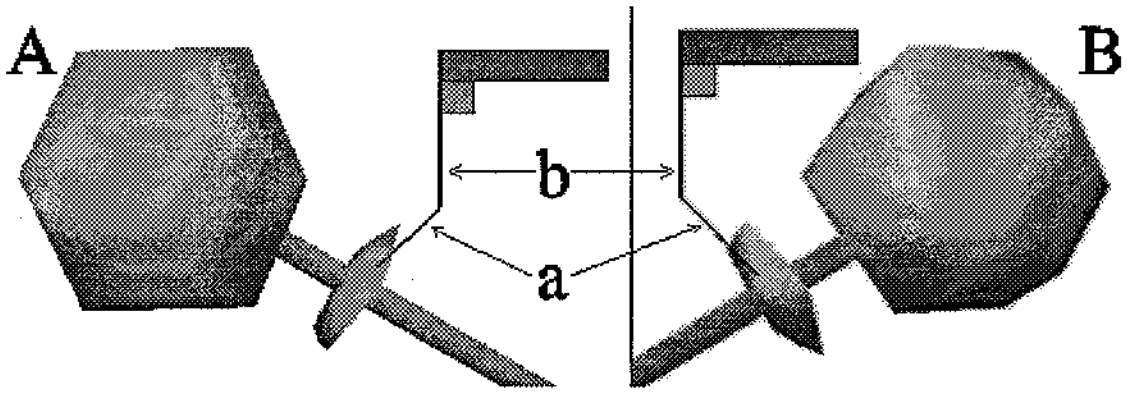


Figure 4. The flags used to reveal the structure and content of messages.

The data flag always appears facing the screen, for any viewing direction so as to make it always clearly visible. The way the flag is drawn depends on the direction of the link.

Figure 4 shows a node A with a link going down on the right side and node B with a link going down to the left. Part a of the flag goes up to the right for node A and to the left for node B. Part b of the flag is unchanged.

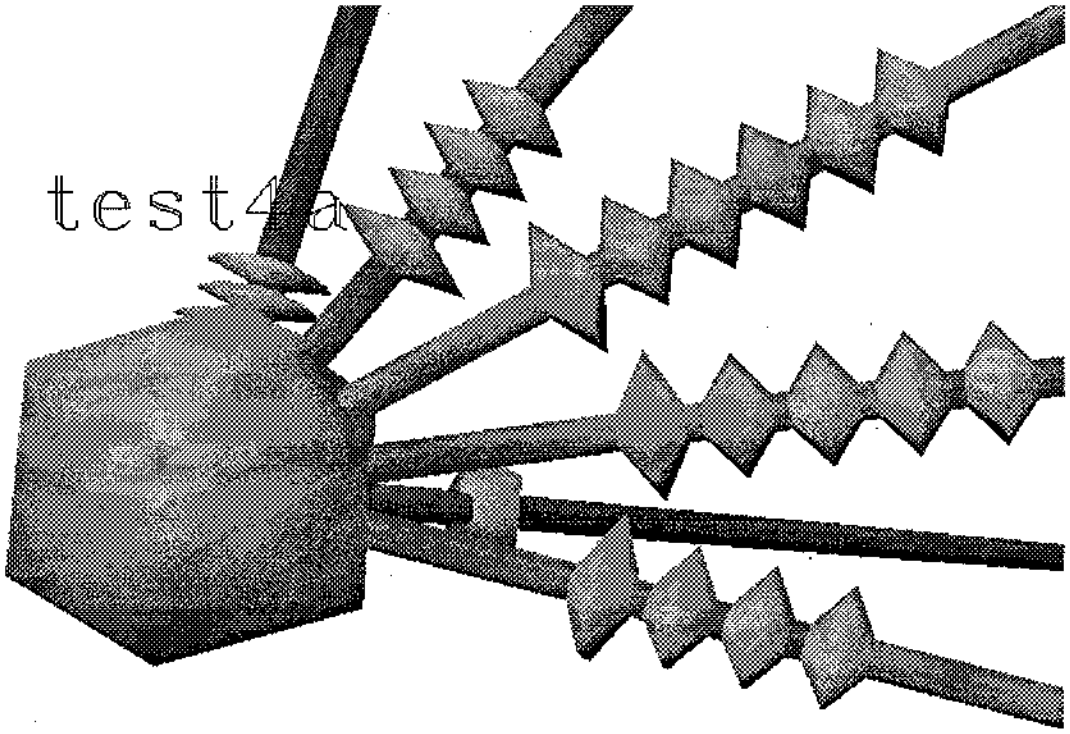


Figure 5. Messages queued up on links waiting to be read

3.6 Message Queues

Once a message has reached the destination process, it is put in a queue before it is read. The waiting queue is not necessarily a first-in first-out queue as any message may be read out of order in PVM.

PVMtrace shows a queue as a string of beads at the receiving end of a link. For each node, there will be as many queues represented as links connecting the node. When a message is read, it shrinks over a period of 5 frames and disappears inside the link. This gives the observer time to see which message was read. Once it has completely disappeared, the messages behind will advance one step to their new position.
This way of representing queues seems very natural because it visually mimics the way PVM handles them. They just seem to be waiting to be absorbed by the receiving process. Figure 5 shows an example of a node with several queues.

3.7 Viewpoint control

Since the process graph is changing over time and is constructed automatically, it is useful to be able to easily change the viewpoint. It is also useful to have the whole graph moving or rotating in time to have a better understanding of the 3D structure.

For viewpoint control, we use widgets which are laid out in the 3D window on the sides. Three widgets can be used for rotation, three others for translation and one for the scaling operation (Figure 6). These widgets can also be set to allow for continuous rotation permitting better understanding of the 3D structure.

4. Animation Speed Control

The main problem encountered when using a unique view that evolves over time comes from the fact that you get only the view of the data at one point in time. You have no trace of the past events and no image of the future. Therefore it can be difficult to place the event in the temporal context. In PVMtrace we provide this sense of context mainly by providing very flexible control over time so that a sequence of events can be replayed fast or slow, forward or backward, and in this way the user may build a mental model at various temporal resolutions. Also, specific queries about a local sequence of events can be answered by replaying that sequence many times if necessary, or very slowly. If the query concerns the pattern of events at the macro level of detail then a fast playback may be more useful.

We also provide a visual record of the event sequence with the interactive time-line. The time-line is like a musical score that represents for each processor, the number of PVM commands issued over time. It is shown in Figure 6. In the middle of the line, a cursor consisting of two vertical lines shows current moment in the animation. A single horizontal line is provided to represent each processor and on this line vertical histogram bars represent the number of commands issued by that processor. Thus, a bar on the right of the cursor of 3 pixels high represents three PVM commands issued during the next time interval.

If the system is set to the state in which processes are color coded, the colors of the bars on the time lines match the colors of the processes. Usually, the time-intervals are short enough so that only one process could have issued a command in a given time-interval. The

size of the whole time-line is limited in height, and the more processors are part of the PVM machine, the less height is available for each one of them. This limits the time line capacity to forty processors if we keep at least two pixels of height per processor line. The interval in time represented by one histogram bar width can be changed. If it is increased, the user will get some information about the activity over a longer time period. If it is decreased, the user will have more detailed graphs representing shorter time periods. A small bar is drawn vertically across the entire time line plot at one second intervals to give a feedback to the user on how much time is passing.

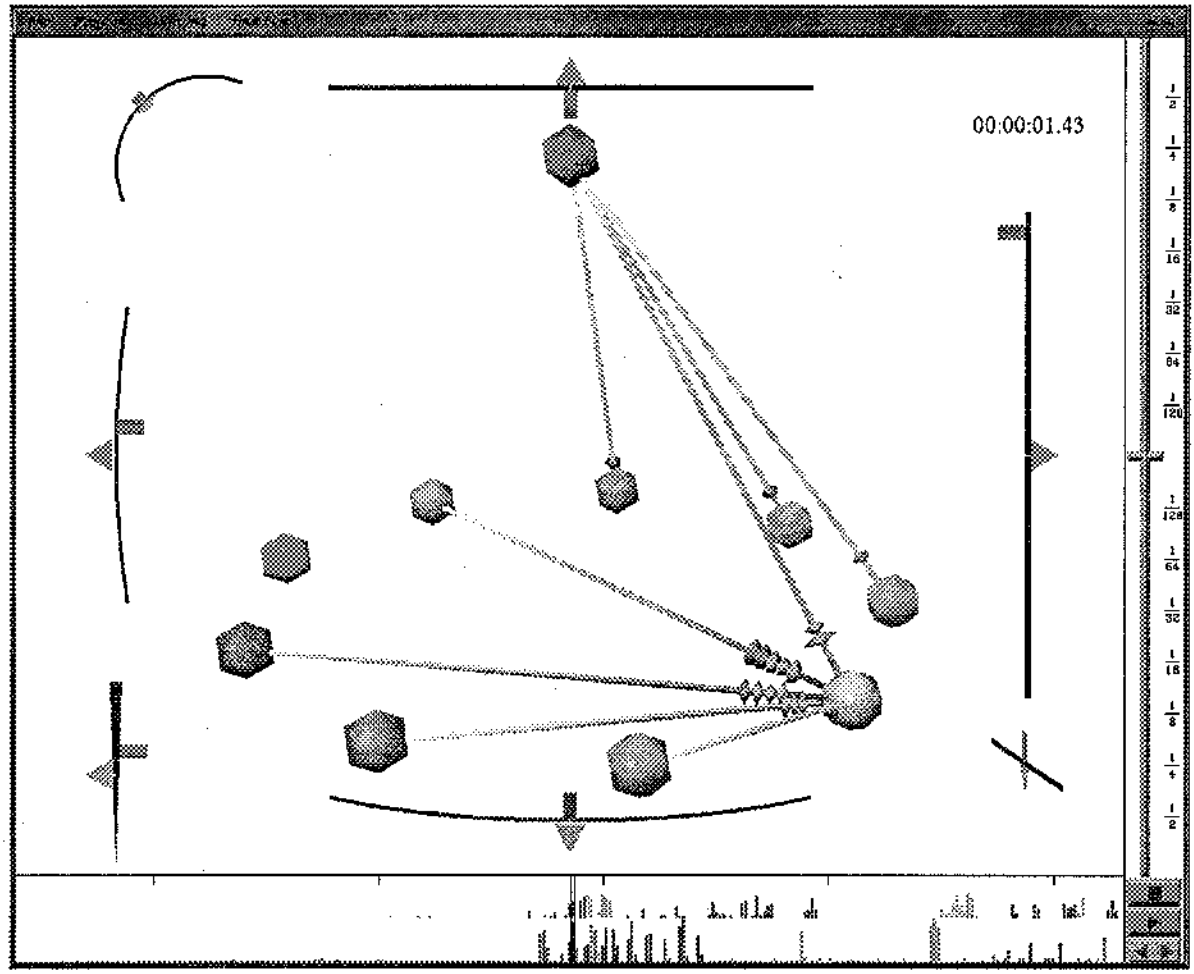


Figure 6 The user interface consistes of a main display region in the center with a number of control devices. The 3D widgets surrounding the 3D graph allow for rotations and translations of the data. The plot at the bottom of the screen shows the sequence of messages. This scrolls across as the animation proceeds with the virtical line acting as a cursor. The speed control is provided at the right hand side. Some of the arcs can be seen fading because they have little recent activity.

As we discussed earlier, time adjustments may sometimes be made automatically when a misordering of events is detected. This results in the events of one processor being suddenly shifted compared to the others on the time-line. It is sometimes disturbing to see these sudden changes, however they are relatively rare.

4.1 Manual control

A special slider is provided on the right side of the screen which can be dragged by the mouse to control the rate of animation. The slider has two purposes: It is there to show the speed of the animation and to control it. When the slider is in the middle, the animation is stopped. Above the middle, the animation will go forward in timewhile underneath, it will go backwards in time. The rate of animation is an exponential function of the slider position.

The buttons underneath the slider provide three commands: start, stop and reverse. When the animation is stopped, the speed value is retained in case of a restart, to go back to the previous speed. Each time the buttons are used, the slider moves automatically to the appropriate place. All these commands are also available through the menu.

The time-line is also an input device, enabling the user to navigate in time. There are two different ways to use it. By clicking on a certain place along the line, the animation will continue at this point in time. Of course, this forces PVMtrace to go through all the commands between the two points in time to do the appropriate changes, like creating and deleting messages or changing the states of the nodes. When the user clicks on the current time and drags to a certain position, the region on the time-line will be highlighted and the animation will be drawn up to the point dragged to with increased speed. These features allow the user not only to go forward to places of interest, but also to go back to places on a time-line to view them again.

4.2 Automatic control

PVM works in the UNIX environment where there are multiple processes sharing each machine (not only PVM processes). The consequence is that a process executes bursts of

instructions, which can be clearly seen on the time-line. There are moments where there is nothing happening and others where there is lot. Irregular distribution of message passing may also occur because processes do extended calculations followed by a burst of message passing.

When the speed of the animation is held constant, it is usually too slow for periods where no PVM commands have been issued and too fast when messages are sent or when new processes are created. Therefore, the user has to continuously accelerate and slow down the animation to be able to see the interesting parts in full detail without being bored by the dull periods. Too avoid this, it is possible to put PVMtrace in an automatic speed control mode, where the speed changes automatically depending on how much is currently happening. The average speed is based on the speed when the automatic mode was selected. The speed is increased or decreased depending on the activity in the next twentieth of a second. The time-slider still indicates the animation speed the animation, which is important, because the user is not controlling it anymore.

The equation used to determine the speed of the animation is:

$$S = \frac{S_r}{e^{\left(\frac{n_r - n}{n_{max}}\right)}}$$

where S_r is a reference speed set by the user, n is the number of events during a 100 ms time interval centered around the current time, n_r is a reference value which defines the number of events required to maintain the reference speed and n_{max} is the expected maximum number of events that can occur in that 100 ms interval.

5. Applications

5.1 PVM Programs

We have made PVMtrace available as a debugging tool to graduate students working on other PVM related projects. We have found it to be an effective debugging aid in a number of instances. We also found a non fatal flaw in the Quicksort program that came with PVM as a test example. In this program, each process checks how many elements it has to sort,

and decides whether it should divide the problem in two again according to the Quicksort algorithm (which takes a certain amount of time) or switch to an insertion sort. By selecting the option of color coding the states of the processes it became immediately clear that at the end, some of the processes at the bottom of the tree were still running and waiting for processes (probably their parents) to send them messages. But the parents had already finished and had sent back their results. We discovered on investigation that some nodes were creating two new uneccessary processes which were never used and were not disposed of, while using the insertion sort. The correction to the code was easily made.

5.2 A Cellular Phone System

The approach of using animations to visualize communications has many other applications besides distributed programs. Many applications from truck deliveries to paper processing in offices have as elements communication via discrete packets of information, the use of queues and need to understand complex systems. We have been working with a group at Bell Northern Research to represent the communications occurring in a cellular phone system. There are five different layers of nodes in the BNR model (Figure 7): the bottom layer forms a hexagonal grid with a Base Station unit at the center. Each hexagon represents a zone of a certain area (a city for example). Each one of these receivers is linked to one of the 12 Base Station Controllers of the second layer. They are themselves connected to 4 pairs of Mobile Service Commutators and Visitor Location Registers, which are interconnected and displayed on two other layers. Finally on the top layer are units to store and control information about the users.

Each time an event happens, for example someone turns on a cellular phone in a certain hexagonal zone, messages will be sent from the base unit and transmitted to the control units in order to set up a link. For each event, messages are sent back and forth. The system can get complicated, when concurrent events happen or, for example, when a cellular phone is moving to another zone.

In order to be useful for cellular phone simulation PVMtrace had to be modified so that it was independent of PVM. In addition, the layout algorithm was changed; in PVMtrace, the nodes are laid out automatically, by building the cone tree. For the cell phone problem, the nodes have their own special layout (especially the bottom nodes displayed in a

hexagonal grid). The data came with coordinates supplied for each node. The loading of the data had to be modified so that when a node was created, it already had its position. However, the animation and visualization components of PVMtrace remained the same. This work appears sufficiently promising that it is continuing with a major joint initiative with BNR.

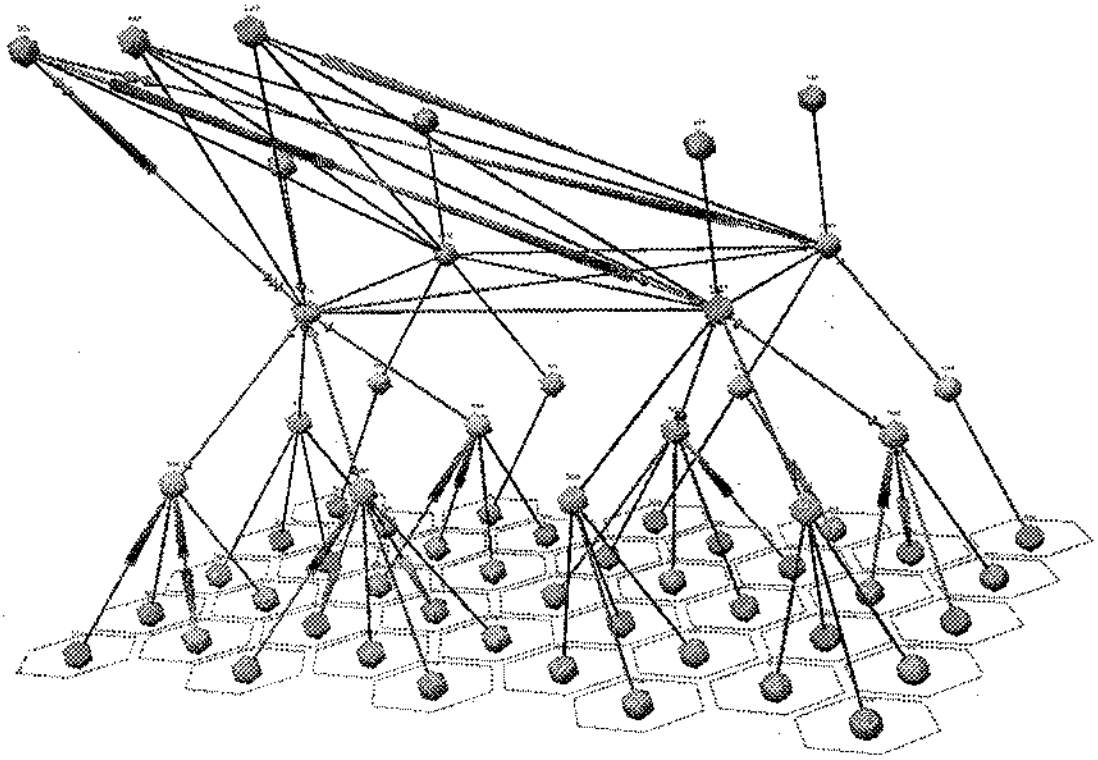


Figure 7. A cellular phone system.

6. Conclusion

We are convinced from our experiences with PVMtrace that animating messages is a viable alternative to the process-time diagram for interpreting message passing behavior. Indeed it has a number of advantages. Because we are no longer using one of the display dimensions to represent time we can represent more complex structures of information. In 3D we can represent structures involving larger numbers of processes with much greater clarity than is possible for the classic process-time diagram. The basic reason for this is that in a process-time diagram only one spatial dimension is available for process layout

whereas in our approach three spatial dimensions are used for process layout. This allows for more nodes and more links to be represented.

There are things that can be done with an animated representation that could not be done with the process-time diagram. For example, in PVMtrace queues are a natural extension of the message passing representation; queued messages simply line up behind the process they are waiting for. However, it is difficult to see how queues could be added to the process-time diagram. Representing messages as individual beads also provides a way of representing more information about the content of messages than can be shown with a line on a process-time diagram.

A possible advantage of the process-time diagram is the ability to see the temporal sequence of events over an extended interval. However, the lack of a visual trace of the kind provided by a process-time diagram does not seem to be a problem when using PVMtrace. This has also been compensated by a flexible control of time. In all, if we count the two ways of controlling animation using the interactive time-line, the speed controller and its forward and reverse buttons and finally the variable rate automatic control, we have six different ways of controlling the animation speed. Moreover we believe all of them to be useful in giving us the necessary control. Although there is duplicate functionality in the sense that is it possible to arrive at the same sequence of events in many different ways, these controllers all have aspects which make them useful. The time-line does give us a partial trace of past events that is undoubtedly useful but its function seems to be more to allow us to return to interesting sequences of events rather than allowing analysis. It functions as a kind of interactive time reference map.

In general one of the main advantages of the approach we have taken is that our approach allows for the integration of many different types of information. Rather than using a multi-view interface, we believe that a rich single representation may be more effective. Thus we can simultaneously view the process state, the messages, the message content and the state of the queues. When information has to be displayed on several graphs as it is for example in the ParaGraph system, it is sometimes difficult to relate what is happening on one node, displayed on one diagram, to something happening on a queue displayed in another diagram.

Acknowlegements

We are grateful to Angelo Bean at BNR for his assistance with the Cell Phone Visualization. Glenn Franck and Mark Paton both assisted with the software development and with advice. This work was funded with an NSERC Strategic Grant.

References

1. Beguelin, A. Dongarra, J. Geist, A. and Sunderam, V (1993) Visualization and Debugging in a Heterogeneous Environment. IEEE Computer, pp. 88-95, June vol 26, #6

2. Geist, A. Beguelin, A, Dongarra, J, Weicheng Jiang, Manchek, R. and Sunderam, V : PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing (The MIT Press, May 1993).

3. Heath, M.T. and Etheridge. (1991) Visualizing the Performance of Parallel Programs. IEEE Software, September, 29-39.

4. Hirose, M., Myoi, T., Amari, H., Inamura, K., and Stark, L. (1990) Development of a visual 3D virtual environment for control software. Human Machine Interfaces for Teleoperators and Virtual Environments, NASA Conf. Publication 10071.

5. Kahn, K.M. and Sarswat, V.A. (1990) Complete Visualizations of Concurrent Programs and their Execution. IEEE Workshop on visual languages, pp. 7-15.

6. Kraemer, E. and J. T. Stasko, J.T. (1993) A Methodology for Building Application Specific Visualizations of Parallel Programs (Journal of Parallel and Distributed Computing 18, pp. 258-264.

7. Kraemer, E and Stasko, J.T. (1994) Issues in Visualization for the Comprehension of Parallel Programs. Proceedings, IEEE Workshop on Program Comprehension.Washington, November. pp. 116-124.

8. Koike, H. (1993) The role of another spatial dimension in software visualization, ACM Transactions on Information Systesms, 11(3), 267-286.

9. Robertson, G.G., Mackinlay, J.D and Card, S. K. (1991) Cone Trees: Animated 3D Visualizations of Hierarchical Information. CHI'91 Proceedings. 189-194.

10. Topol, B. J.T. Stasko, V. Sunderam, (1995) Integrating Visualization Support into Distributed Computing Systems. International Conference on Distributed Computing Systems. Vancouver, May pp 19-26.

11. Ware, C. and Franck, G. (In press) Visualizing Information Nets in Three Dimensions. ACMTransactions on Graphics