

VisuaLinda: A Framework and a System for Visualizing Parallel Linda Programs

Hideki Koike*

Tetsuji Takada

Graduate School of Information Systems
University of Electro-Communications
1-5-1, Chofugaoka, Chofu, Tokyo 182, JAPAN
Tel: +81-424-83-2161
Email: koike@cas.uec.ac.jp

Department of Communications and Systems
University of Electro-Communications
1-5-1, Chofugaoka, Chofu, Tokyo 182, JAPAN
Tel: +81-424-83-2161
E-mail: zetaka@qr.cas.uec.ac.jp

ABSTRACT

This paper describes a framework and a system for visualizing parallel Linda programs. The system is called VisuaLinda, which is a Linda server itself as well as a visualization tool. Since the visualization module is built-in the Linda server, programmers do not need to insert/delete additional sentences into/from their client programs in order to obtain visualization. This framework extremely reduces the programmers' work in debugging parallel programs and helps prevent the probe effect, which is a main concern in monitoring parallel programs. Secondly, the VisuaLinda uses three-dimensional space to display both the relation between Linda server and clients, and their execution. This framework allows us to display much larger number of processes than in 2D, to see two relations simultaneously, to improve the visibility of communication lines, and to see each process's state as well as an overview of the execution.

Keywords: Algorithm animation, debugging tool, information visualization, Linda, parallel programming, performance tuning, three-dimensional graphics, visualization.

INTRODUCTION

Debugging parallel programs is much more difficult than debugging sequential programs. The first reason is that the breakpoint approach used in sequential programs does not work because many processes run in parallel. The second reason is that programmers have to worry about the correctness of each process as well as the correctness of the communication between processes. The third reason is that it is often difficult to repeat the bug because of the non-deterministic behavior of the parallel programs.

Visualization has been playing an important role for debugging such parallel programs. By visualizing the execution pattern of complex programs as a diagram, it helps programmers to understand the program faster and more clearly.

Most of the visualization systems developed until now are for special parallel hardwares such as transputer, connection machine, and so on. Since they have their own trace tools, it is easy to obtain trace data including the information about the processes' state and their execution time.

On the other hand, another parallel computing using general purpose workstations on local area network is getting popular and popular. PVM (parallel virtual machine) is one such example. However, it is more difficult to visualize this kind of parallel computing because of the reasons described in the next section.

This paper describes the *VisuaLinda*, which is a Linda server itself as well as a visualization tool for parallel Linda programs. There are two major contributions in the VisuaLinda system. The first is the build-in design of a visualization module. Programmers do not have to insert/delete additional procedures into/from their programs in order to obtain the visualization. The second is a use of three-dimensional space. As a result, programmers can see both a relation between processes and a time flow of processes simultaneously.

VISUALIZATION OF PARALLEL PROGRAMS

Visualization Framework

McDowell wrote that visualization approach is an effective method for debugging parallel programs, and pointed out the importance of the following approaches[11]:

- Time-process diagram approach
One axis represents processes and another repre-

*Current address: 481 Minor Hall, School of Optometry, University of California, Berkeley, CA 94720

sents time. This diagram can display an execution pattern of processes for a certain period of time. It, however, has little information about each process at one time.

- Animation approach

Processes are placed in two-dimensional space to represent their relations. Then, the state of each process or communication between them are displayed from time to time. This diagram can display more information about each processes at one time. We, however, cannot obtain a global view of the execution pattern of the whole system.

Then, he concluded that an ideal debugger for parallel systems should have an ability to display both of them.

Software Probe is Dangerous

To display such a process-time diagram or an animation, it is necessary to obtain information about each process and communication between them. Using this information, the visualization system makes a real time animation or replays them later.

In case of sequential programs, to make such visualizations is relatively easier. For example, algorithm animation systems established by Brown[1] and Stasko[13] display the execution of the program or algorithms by inserting additional procedures called *software probes* just before or after the *interesting events*[1].

We, however, have to be very careful to apply such techniques to parallel programs. The reasons are as follows:

- Lack of a global clock

Parallel processes run on two or more computers (processors). Since each computer has its own clock, there is no standard time which should be used to obtain the animation or the process-time diagram.

- Probe effect

Probe effect is one of main concerns in discussion of parallel systems. If we put additional procedures into parallel programs to monitor the system some information or to get visualization, the system's behavior might be completely different between before and after the insertion.

Moreover, the software probe approach requires programmers to insert/delete additional procedures at the corresponding point. Once the bug is found, these procedures have to be deleted. However, if another bug is found, programmers have to insert them again and also delete again after debugging. Most of the programmers have the similar experience to insert/delete lots of output sentences (such as "printf") into/from their programs when they debug their programs.

VISUALINDA

On the basis of analysis described in the previous section, we developed the VisuaLinda which is a Linda server as well as a visualization system for parallel Linda programs.

Linda

Linda[3], proposed by Carriero and Gelernter, is a concept of parallel programming rather than a parallel programming language itself. By adding a couple of primitive procedures for interprocess communication to existing sequential programming languages such as C or FORTRAN, these languages are extended to parallel programming languages.

To do interprocess communication, Linda has a shared space and primitive procedures to access the shared space. The shared space is called *tuple space* and is used to send/receive data called *tuple*. There are a couple of primitive procedures for communication¹.

out Output a tuple to the tuple space.

in Input a tuple from the tuple space. The tuple is deleted from the tuple space.

rd Read a tuple from the tuple space. The tuple is not deleted from the tuple space.

Interprocess communication using tuples and a tuple space is illustrated in Figure 1.

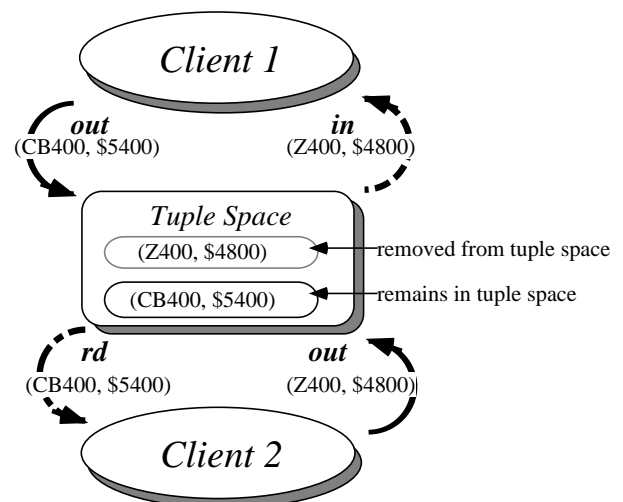


Figure 1: Interprocess communication using tuples in Linda.

¹Eval has not yet implemented in the VisuaLinda.

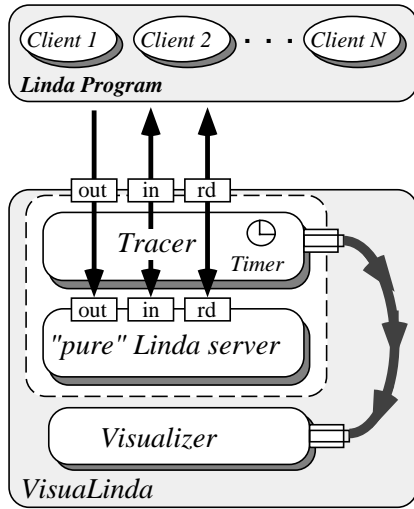


Figure 2: VisuaLinda System Overview

Built-in Visualization Module

As is described above, all communications are done through the tuple space maintained by the Linda server. We focused on this and decided to embed a visualization module in the server. When the server receives a request from a client, the server hands it over to the visualization module with a time when the server started to process the request as shown in Figure 2.

By adopting this built-in visualization module approach, the problems described in the previous section are minimized as follows:

- global clock;

Although clients run in parallel, the server processes requests from the clients sequentially. Therefore, the time which the request is processed can be regarded as a standard time. We can use this to make an animation or to draw a process-time diagram.
- probe effect;

From the programmer's point of view, the VisuaLinda is a Linda server itself. And client programs are not affected whether or not the server makes visualization. Since no additional procedure is inserted to the client programs, the probe effect is minimized with the VisuaLinda.
- no change to client program;

At the same time, programmers do not need to insert/delete any procedures into/from their client programs. Programmers can concentrate their attention on writing their programs. Visualization is automatically obtained whether or not they want.

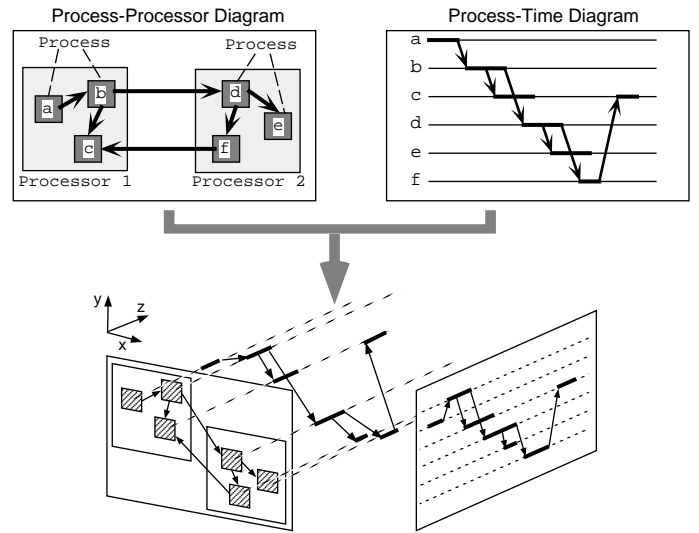


Figure 3: Three-dimensional framework for visualizing parallel programs. Processes are laid out in XY-plane, and Z-axis indicates time. Therefore, process relation diagram and process-time diagram, which are represented as two separated diagrams, are visualized in one 3D diagram.

(If they do not want to see the visualization, they may iconify the VisuaLinda window.)

3D Framework

In [6], Koike proposed a framework for visualizing massively parallel processes. In this framework, processes are laid out in XY-plane, and Z-axis indicates time as shown in Figure 3. Therefore, a process relation diagram and a process-time diagram, which are normally drawn in separated diagrams, are visualized in one 3D diagram. It is noteworthy that two approaches, process-time diagram and animation, which McDowell listed as requirements for ideal debugger, are satisfied with this one 3D diagram.

The main features of this framework are:

- ability to display large amounts of processes;

We can use 2D space to lay out processes instead of 1D.
- simultaneous understanding of a process relation and a time flow;

It will reduce human's cognitive load. Detailed discussions are described in [6]
- improvements of visibility of communication lines;

Communication lines are generally not crossing in 3D space. By changing viewpoints, users can get a good position to see the diagram.

- integration of detailed information of processes and overview of the graph;
As ParspectiveWall[9] demonstrated, perspective view of 3D graphics allows us to see both details (process's state, name, etc.) and global context (overview of the graph).

Implementation

The VisuaLinda was implemented on Silicon Graphics IRIS Indigo/XS with C++(GNU C++) and 3D software visualization tool VOGUE[6]. Figure 4 is an example visualization of a Linda program with six clients. A server process is placed at the top of XY-plane and client processes are placed under the server. Z-axis indicates time. In the figure, a sphere represents an existence of a client process. As it runs, the "bar" grows along the time axis. The bar is displayed in transparent when the process is waiting a tuple, and otherwise it is displayed in solid. Moreover, large transparent polygons represent workstations. In this example, four workstations are used.

Currently, the following functions are implemented in VisuaLinda.

- changing viewpoint
By using GUI sliders or keyboard, it is possible to change the 3D viewpoint with animation.
- displaying each tuple
By clicking a communication line, corresponding tuple data is displayed (as shown in Figure 4).
- displaying tuple space
By clicking a certain point of server process, all tuples in the tuple space at that time are displayed.
- displaying state of each process
The process waiting for a tuple is displayed in transparent. Otherwise it is displayed in no transparent.
- displaying hostname
By clicking a sphere representing a process, host name of the computer in which the process is running is displayed.

VISUALINDA IN PRACTICAL USE

This section describes how the VisuaLinda is used for such applications.

Displaying a Large Number of Processes

First, we start to compare the VisuaLinda's 3D framework and traditional 2D framework. We visualized Linda programs with 14 clients. The results are shown in Figure 5 and Figure 6. As you can see, the vertical axis overflows in the 2D diagram and communication lines reduces the visibility of the diagram. On the other

hand, the 3D framework has much more space to display processes, and we can understand communication between processes more clearly.

Used in Debugging

It is possible to classify bugs of parallel programs into two categories, one is of each program and another is of communication. Debugging techniques for sequential programs are useful for the former but not for the latter.

Figure 7 are visualizations of the programs with bug (A) and without bug (B). In the (A), processe A is ended in transparent without a cube which represents the end of its execution. Thus, programmers can understand that this process is waiting (forever!) for tuples and that process B should output more tuples to tuple space. Figure 7 (B) shows the result after the programmer correct this bug.

This seems to be a tiny example. But this example shows us the difficulty of finding such bug. Such bugs are never found with a debugging tool for sequential programs. With the VisuaLinda, programmers can find the bug just by seeing the visualization.

It is important to notice that this bug might not be occurred if the access order to a tuple space is different. Consider two processes, one of which wants to input a tuple with **in** and another wants to read in the tuple with **rd**. If the former access to the tuple space earlier, the latter can also read in the tuple because the tuple is kept existing in the tuple space. On the other hand, if the latter access earlier, the former will be blocked as shown in Figure 7. If we put software probes in order to obtain visualization, the access order might change and therefore the bug might not be occurred.

Used in Performance Tuning

A primary motivation for using parallel programs is the increase in computational speed as compared to using only sequential programs. However, if the design of parallel programs is not correct, the increase in speed may not meet expectations. On the contrary, it might be slower because of the overhead of communication. In Linda, each process is allotted to computers. The processes allotted to the same computer run in concurrent, not in parallel. Therefore, the process which has heavy computation should be assigned to the faster computer. The light processes may be assigned to the slowest computers or some of them may be assigned to the same computer.

Figure 8 represents an example of the performance tuning. In figure 8 (A), programmers recognize that many parts of clients are transparent. This is because processes are always waiting for tuples. Figure 8 (B) is the visualization after improving the programs. There are

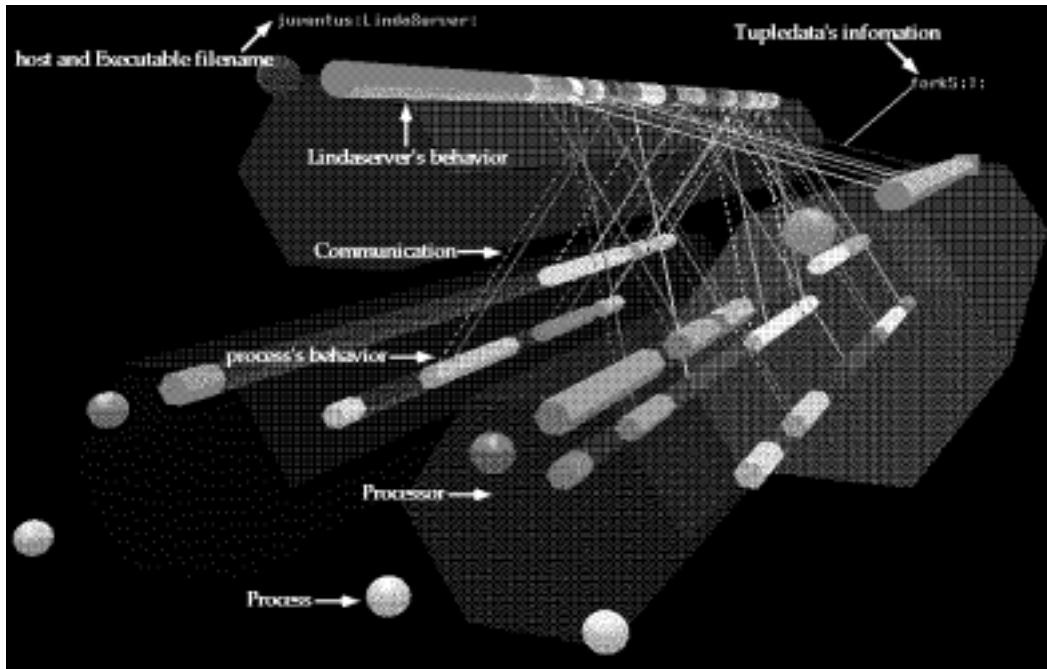


Figure 4: An example visualization of Linda program with the VisuaLinda. A server process is placed at the top of XY-plane and client processes are placed under the server. The “bar” grows along the time axis as each process runs. Large transparent polygons represent workstations in which the processes are running.

less transparent parts. It means that computation is done more effectively.

With the VisuaLinda, programmers can easily know which process reduces the performance of the whole system. Then, they can redesign the program and separate this process to some processes, or they just reassign this heavy process to a faster computer.

DISCUSSION

Limitations of VisuaLinda

There are some limitations in the VisuaLinda framework. The first is a computational efficiency problem. Since the VisuaLinda has to do additional work for visualization as is compared to the pure Linda server, which does not have visualization capability, the VisuaLinda is relatively slower than the pure Linda server.

Second, our framework is not applied as it is to the multiple server Linda, which is currently being studied in Linda community. Since there are two or more servers, there are the same number of standard times as the number of servers. Therefore, VisuaLinda would need to be extended to multiple timelines for multiple servers.

Related Work

ParVis[8] is a visualization system for MultiLisp, and displays its parallel processes in 2D process-time diagram. PIE[7] is a visualization system for Mach operating system and is used for performance tuning of the system's kernel. JED[10] is a visualization system for parallel programs running on Cedar multi processor environment. These work use 2D process-time diagram. Therefore, they cannot display much more processes nor represent a process-processor relation as the VisuaLinda can.

As three-dimensional visualization work was done by SemNet[5] and Information Visualizer[2, 12, 9]. These systems can display only one relation at a time. Although users change their viewpoints, the obtained information is the same. For example, Cone Tree[12] shows one hierarchical structure whichever viewpoints users choose. On the other hand, our 3D framework makes it possible to display two different relations without disturbing each other.

We first applied this 3D framework to parallel processes of control software at Tokyo Electric Power Company[6] and also experimented virtual reality interfaces[4]. However, it depends on the special hardware and debugging tools for it. Therefore, we want to apply our framework to more general parallel computing and to show

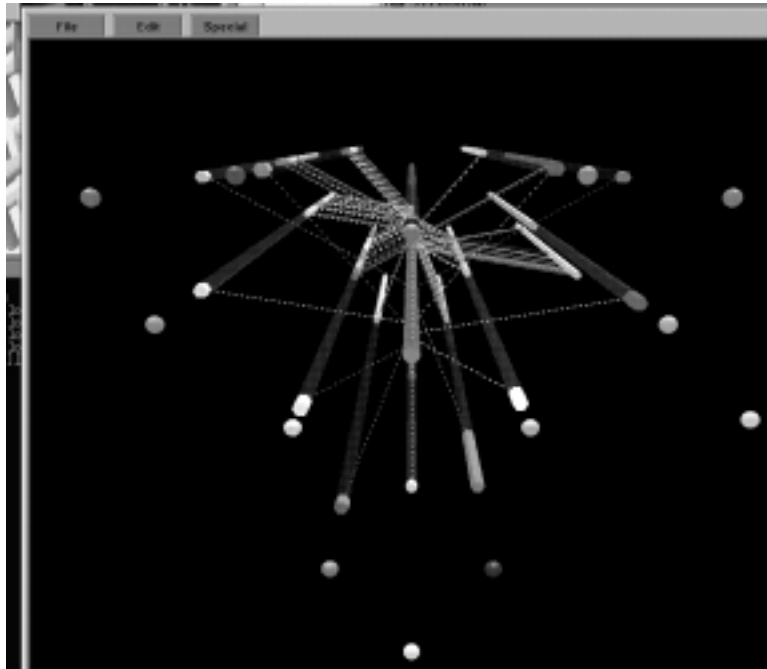


Figure 5: The 3D visualization obtained by the VisuaLinda. There is enough space to lay out processes, and it is easier to recognize communication between processes.

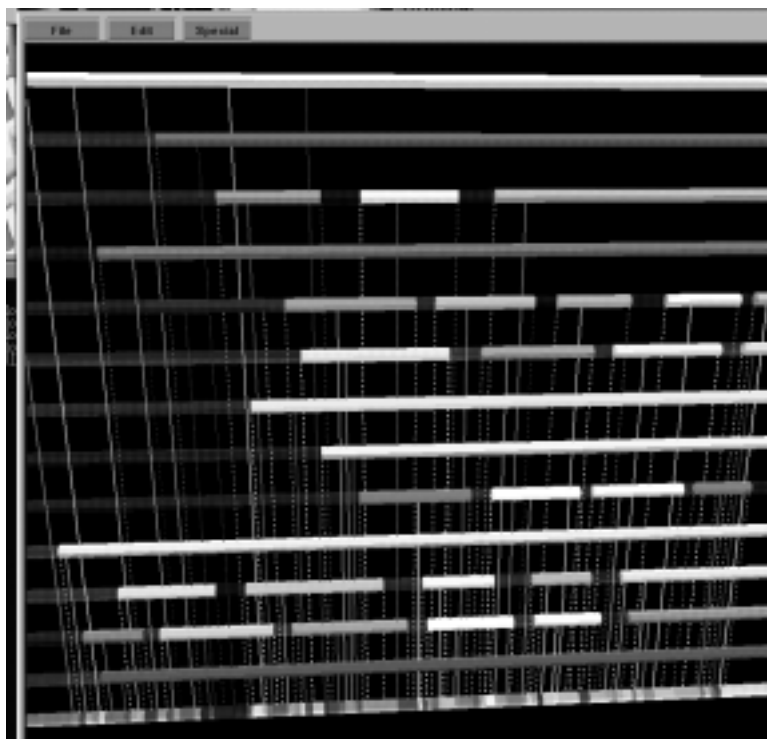
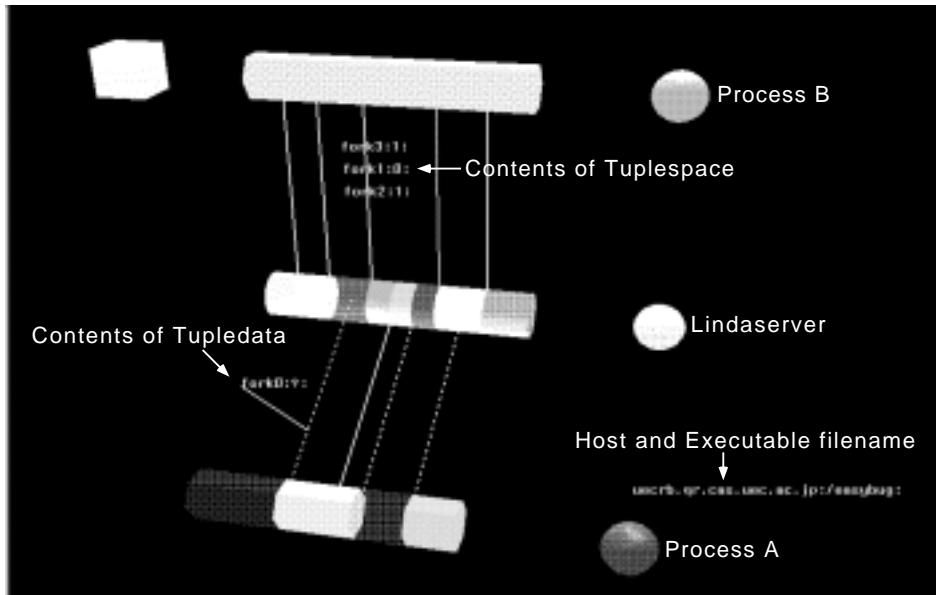
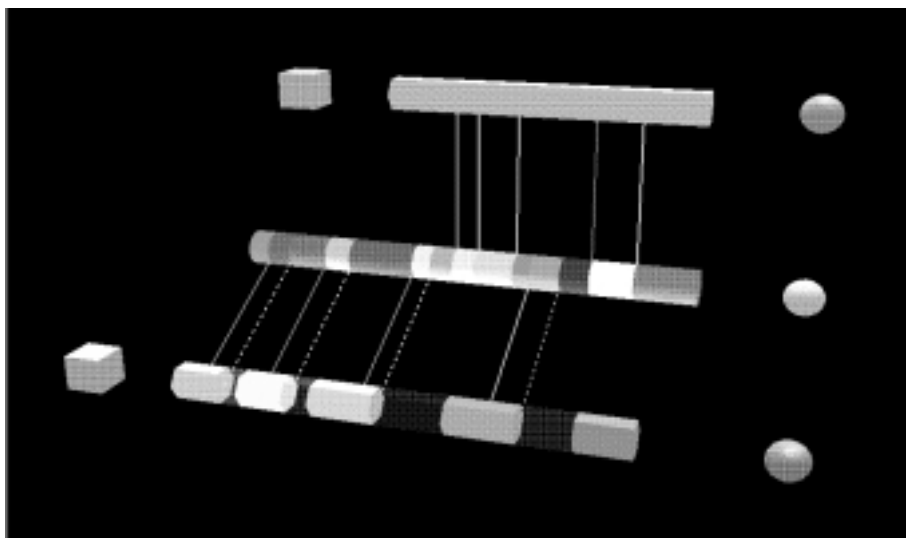


Figure 6: The 2D process-time diagram simulated by the VisuaLinda. The Y axis overflows with just 14 processes.

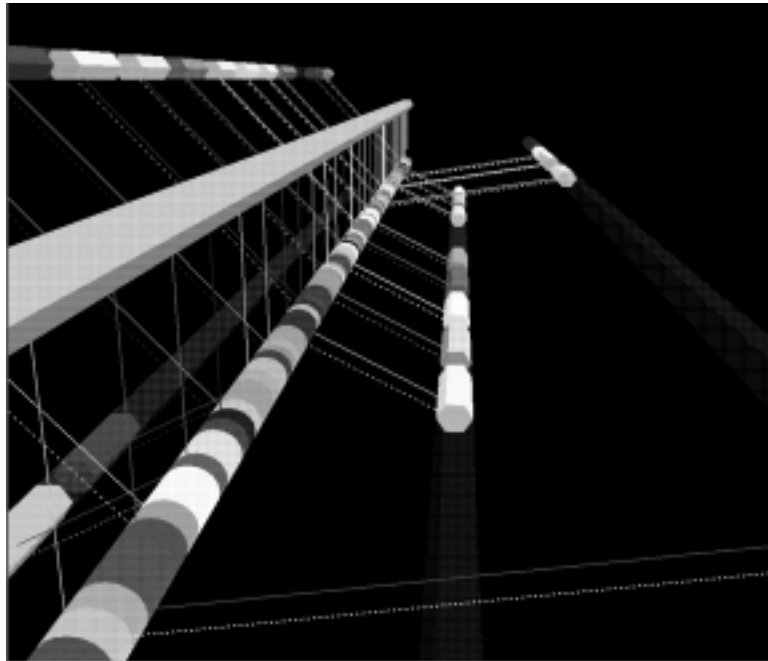


(A)

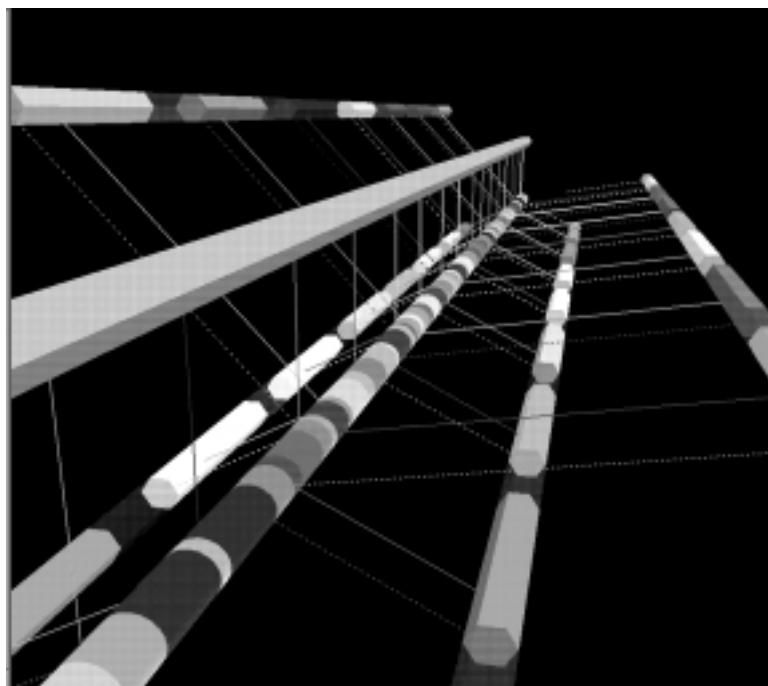


(B)

Figure 7: Using the VisuaLinda for debugging. The (A) is a visualization of the program with bug. Since a tuple required by process (A) is not in the tuple space, the process A is blocked. The (B) is a visualization after fixing this bug.



(A)



(B)

Figure 8: Using the VisuaLinda for performance tuning. The (A) is a visualization of the program coded inefficiently. The programmers recognized its inefficiency because there are many transparency parts in this visualization. On the other hand, the (B) is a visualization of the program coded efficiently. There are less transparency polygons.

its effectiveness.

CONCLUSIONS

This paper describes the VisuaLinda which is a Linda server itself as well as a visualization system for parallel Linda programs.

- Built-in design of the visualization module minimize the probe effect which is a main concern of monitoring parallel systems;
- Client programs do not need to be changed to obtain visualization. This extremely reduces programmers' work;
- VisuaLinda's 3D framework solves some of the limitation in 2D process-time diagram.

ACKNOWLEDGMENTS

The authors would like to thank Toshiyuki Masui of SHARP Corporation, who generously developed and provide us with a pure Linda server.

REFERENCES

- [1] BROWN, M. H. *Algorithm Animation*. MIT Press, Cambridge, MA, 1988.
- [2] CARD, S. K., ROBERTSON, G. G., AND MACKINLAY, J. D. The Information Visualizer, an information workspace. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)* (1991), ACM Press, pp. 181–188.
- [3] CARRIERO, N., AND GELERNTER, D. Linda in context. *Communications of the ACM* 32, 4 (1989), 444–458.
- [4] ET.AL., H. A virtual reality application for software visualization. In *Proceedings of IEEE VRAIS'93* (1993).
- [5] FAIRCHILD, K. M., POLTROCK, S. E., AND FURNAS, G. W. SemNet: Three-dimensional graphic representation of large knowledge bases. In *Cognitive Science And Its Applications For Human-Computer Interaction*, R. Guindon, Ed. Lawrence Erlbaum Associates, 1988, pp. 201–233.
- [6] KOIKE, H. The role of another spatial dimension in software visualization. *ACM Trans. on Information Systems* 11, 3 (July 1993), 266–286.
- [7] LEHR, T., SEGALL, Z., VRSALOVIC, D. F., CAPLAN, E., CHUNG, A. L., AND FINEMAN, C. E. Visualizing performance debugging. *IEEE Computer* (1989).
- [8] LINDEN, L. B. Parallel program visualization using ParVis. In *Performance Instrumentation and Visualization*, M. Simmons and R. Koskela, Eds. ACM Press, 1990, pp. 157–188.
- [9] MACKINLAY, J. D., ROBERTSON, G. G., AND CARD, S. K. The perspective wall: detail and context smoothly integrated. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)* (1991), ACM Press, pp. 173–179.
- [10] MALONY, A. D. JED: Just an event display. In *Performance Instrumentation and Visualization*, M. Simmons and R. Koskela, Eds. ACM Press, 1990, pp. 99–116.
- [11] MCDOWELL, C. E., AND HELMBOLD, D. P. Debugging concurrent programs. *ACM Computing Surveys* 21, 4 (1989).
- [12] ROBERTSON, G. G., MACKINLAY, J. D., AND CARD, S. K. Cone Trees: Animated 3D visualizations of hierarchical information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)* (1991), ACM Press, pp. 189–194.
- [13] STASKO, J. T. TANGO: A framework and system for algorithm animation. *Computer* 23, 9 (September 1990), 27–39.