Programming Environment Specified for Interprocessor Communications Based on Graphical User Interface

Yasutaka SAKAYORI, Motoki MIURA and Jiro TANAKA
Institute of Information Sciences and Electronics,
University of Tsukuba
Tennodai 1-1-1, Tsukuba, Ibaraki, 305-8573 Japan

Abstract *GRIX system, which is a visual programming system specified for interprocessor communications in parallel computing, is proposed in this paper. In general, the scheduling of interprocessor communications during constructing parallel programs is the most difficult and important job in order to realize its high performance. GRIX brings users easy scheduling and intuitive realization of interprocessor communications because of its GUI based operations. Moreover, by automatic code generation by GRIX system, users do not need to know about the specifications of parallel programming like PVM and MPI. All of users have to know is the rules of figured images and operations on the GRIX window. The system outline and ,especially, its GUI based operations are described in this paper.*

Keywords: Visual Programming System, Interprocessor Communications and Parallel Programs

1 Introduction

GRIX system [1, 2], which is a visual programming system specified for interprocessor communications appearing in programs for Message-Passing parallel computing, is proposed in this paper. The aim of GRIX is to realize effective interprocessor communications and to reduce users' jobs by using visual input instead of complex input with textual specifications.

1.1 Background

The parallel computing architectures can be classified by its characteristics, the examples are multiple workstations connected by LAN(Local Area Network) and a single computer including multiple processors. The peak performance of the parallel computers especially called massively parallel computers, which include hundreds or thousands processors, reaches hundreds G-flops or several T-flops. But the peak performance is appeared in the case of the computing which have no computing loss in all processors and, moreover, no communications among processors. The real applications can not reach to the peak performance because of interprocessor communications, cache miss and so on. The serious deterioration of the performance is invoked by kinds of the program structure. This phenomenon commonly appears on all parallel computing architectures.

Especially, programmers must pay large attention for interprocessor communications when construct parallel programs. The cost of interprocessor communications is the bottleneck of computing. Moreover, they bring barrier synchronizations in some cases. Inefficient coding brings frequent barrier synchronizations and the computing loss from the peak performance will be large.

1.2 Motivation

There exists lots of means for coding the sentences of interprocessor communications, like

PVM [3], *MPI* [4] and native functions implemented only for each computers. Those specifications show a tendency to be complex because we require high performance of execution. One of our emphasizing point is those complexities which bring confusion should be eliminated. For instance, in order to realize the most simple communication, which is a pair of send and receive, by using PVM functions, programmers have to complete four procedures at least, packing data (ex. `pvm_pkint`), sending data (ex. `pvm_send`), receiving data (ex. `pvm_recv`) and unpacking data (ex. `pvm_upkint`). Moreover, some variables included in each function must be complex because of bringing high performance. This tendency must be strengthened at more native environments.

We insists that forcing programmers to learn the meaning of difficult functions, especially unessential ones for interprocessor communications like memory mapping, is not desirable. In order to release programmers from complex specifications of interprocessor communications, we have selected Visual Programming method [5]. Visual Programming method is the programming form using the visual images like icons, figures, animations and so on. This interface brings users more intuitive understanding of the interprocessor communications. Users only need to consider the intuitive information like "who (which processing elements)", "where (which address space)", "what (which data)" and "how (like blocking and non-blocking)." If the system automatically generates the actual code of PVM and MPI, users do not have to understand their detailed specifications of each library procedures. Moreover, by adding the scheduling algorithm, users can input any communication patterns.

2 Visual Programming System GRIX

GRIX is used in the case if the interprocessor communications appears in SPMD (Single Program / Multiple Data) programming. GRIX has the following four features: input based on GUI (Graphical User Interface), highly portable optimization technique, GUI output of the optimization result, and code generation corresponded to any parallel computing environments. It has become more intuitive for programmers to input communications because of GUI. The optimization which has high portability has made possible to generate the actual codes corresponded to any environments. The graphical output of the result of optimization enables users to realize the actual code intuitively and support in the program debugging.

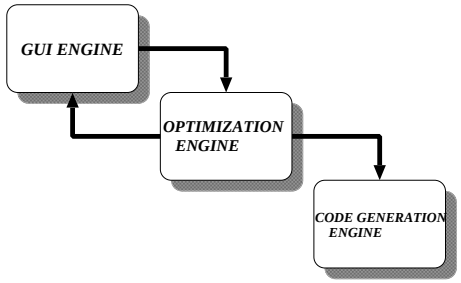

Figure 1: System Structure of GRIX

GRIX system can be classified into three subsystems: GUI Engine, Optimization Engine and Code Generation Engine (Figure 1). GUI Engine changes the input information from the users into the file which can be executed in Optimization Engine. Optimization Engine changes the communications provided from the file into the one of the faster to communicate and easier to generate the actual code. The modified information is written into two files, one is for the graphical output and the other is for code generation. GUI Engine shows the modified information graphically. Graphical output is for the purpose of intuitive realization of optimizations and can also help users during their debugging works. The file for code generation is used by Code Generation Engine in order to generate the actual code for each environment automatically. Code generation for each environment can be realized by preparing corresponding translator.

3 Interface Specified for Interprocessor Communications

We have paid attention to the viewpoint of the users, when they input the send-receive relationships. We explain the visual interface of GRIX with the most basic example assuming the input of 1-dimensional node ID (We call the physical processor represented on the GUI system "node").

3.1 Input from Absolute Viewpoint

As the one of the viewpoint, we have thought about the case that users already know the number of processors and they input the relationships in advance. For instance, it is the case that the programmer uses the single computer including multiple processors, like massively parallel computers (Figure 2), and he wants to use the actual ID of processors which have been assigned by each environment, like *"The processor 2 sends the data toward the processor 5."* We call this viewpoint *"Absolute Viewpoint."*

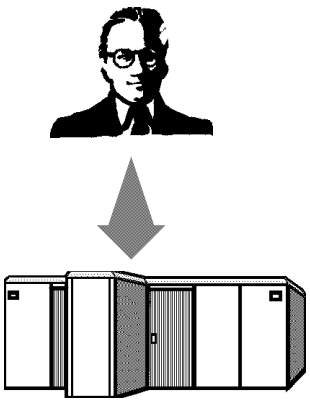

Figure 2: Absolute Viewpoint

The initial window (left of Figure 3) appears by informing the system of the number of showing nodes. In this window, both of the send-node and the receive-node are represented as circles. The nodes lined up vertically mean the set of processors, the two nodes aligned on the horizontal line represent the same processor: the left one is a send-node and the right one is a receive-node. The number in the left of each send node shows the ID of its processor, which is 0-origin. The user describes the relationship by drawing arrows. The relationships between nodes are shown as arrows. The most basic operation is dragging a mouse from the send-node to the receive-node. The user can describe any relationship among nodes naturally (right of Figure 3).

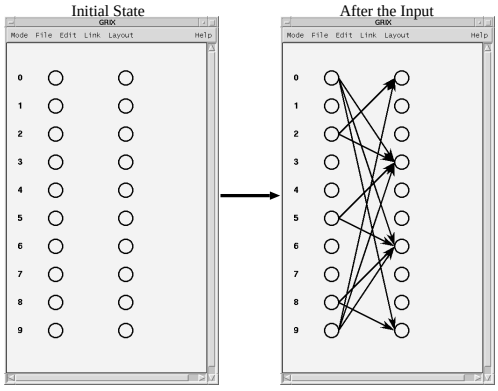

Figure 3: Input from Absolute Viewpoint

3.1.1 Advanced Operations for Frequent Communication Patterns

Though the user can describe every type of relationship by using this operation, he has to draw the arrows one by one. Therefore, we implement some multiple-arrow drawing operations which help the users input three typical communications like Broadcast, All-to-All Broadcast and Shift. In the case of the input of 1-to-multiple communications like Broadcast and Scatter, the user selects 1 send node and multiple receive nodes at first (upper left of Figure 4: selected nodes change into emphasized color). Then, by selecting existence order from the menu bar or typing established shortcut key (right of Figure 4), the system describes all arrows for selected nodes (lower left of Figure 4).

While the input with Absolute Viewpoint, user can also use the advanced operation for

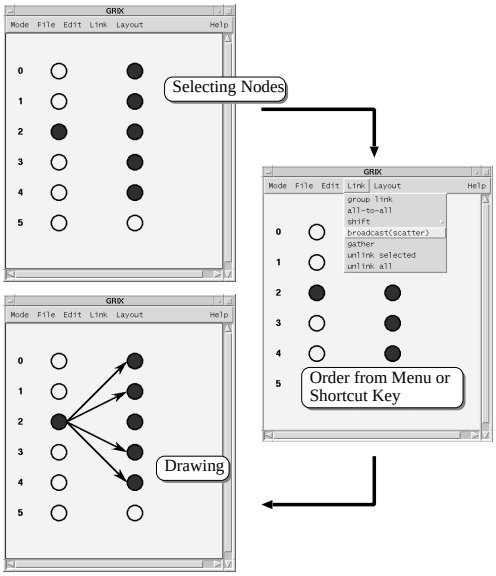

Figure 4: Advanced Operation for Scatter

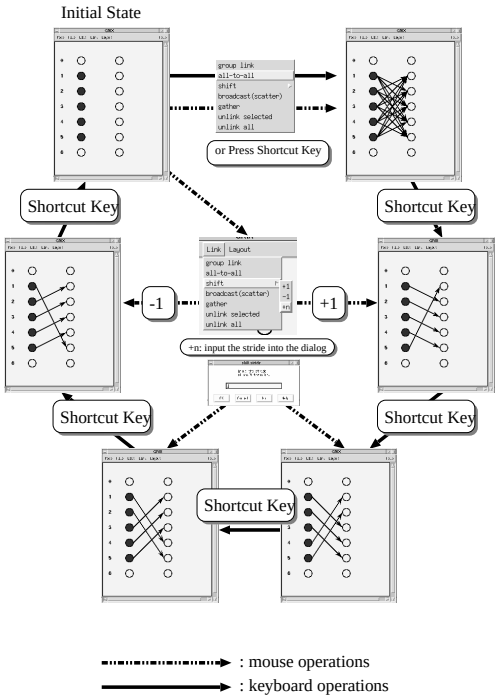

Figure 5: Advanced Operation for All-to-All Broadcast

the input of All-to-All Broadcast and Shift transfers. First, the user selects multiple send nodes (upper left of Figure 5). With the input of the shortcut key, the system describes the arrows of All-to-All Broadcast (upper right of Figure 5). With the once more input of the shortcut key, the system changes the arrows into the one of Shift transfer whose stride is +1. By the repetition of the input of the shortcut key, the stride of Shift transfer is incremented. If there exists no more stride, the window is returned to the initial state. The describing of All-to-All Broadcast is also available by the selecting '`Link` → `all-to-all`' after node definition (upper left to upper right of Figure 5). Similarly, the Shift transfers are available by selecting '`Link` → `shift`' and defining the stride (center of Figure 5).

We have implemented the advanced operation for Unlink which clears the arrows connected to existence node. This operation is completed by selecting the existence one node and ordering with '`Link` → `unlink`' (Figure 6). Those advanced operations bring more rapid input by omitting monotonous repetition.

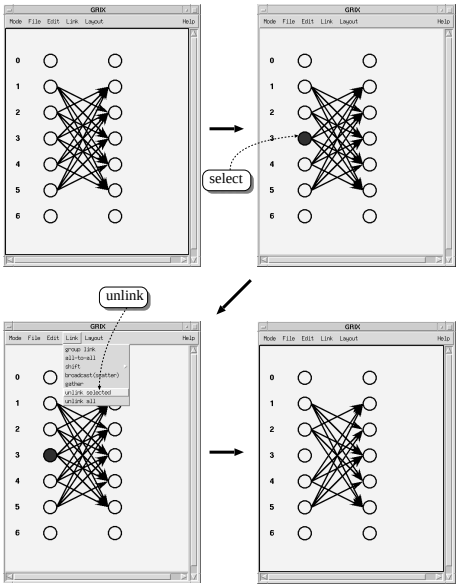

Figure 6: Unlink

3.2 Input from Relative Viewpoint

In contrast to Absolute Viewpoint, the user frequently images with the relative relationships among processors. This is often happened when the programmer uses the multiple workstations connected by LAN. For instance, it is one of the Scatter image like *"the processors whose ID are even number send the data to the processors which have the -3,-1,+2 distance from them."* In this viewpoint, users construct the communications with the notice of the movement in existence processors. We call this viewpoint *"Relative Viewpoint"* (Figure 7).

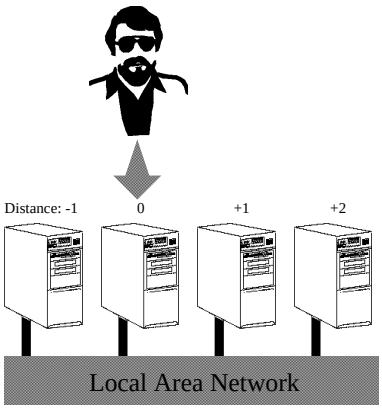

Figure 7: Relative Viewpoint

Explaining how to input this example, at first, user must inform the system of the number of nodes which is needed to represent all the relative processors (in this example, it is 6). The upper left of Figure 8 is the initial window with 6 nodes. In this window user selects one node which is the center of viewpoint, we call the node *Owner*, and the system paints it with emphasized color and shows the distance in the left of nodes (upper right of Figure 8). If the user changes the owner, the value of distance shown in the left of the nodes is rewritten into the new value. Afterward, user describes the arrows with the same operations of Absolute Viewpoint (lower left of Figure 8).

There remains the job of defining the condition of Owner. In this example, the condition of the active nodes is *"ID is even number."* Defining the condition starts from selecting `'Edit` → `define active nodes'` from the menu. The choices prepared as the default are `'all'` (all nodes are active), `'1 node'` (existence 1 node is active), `'even'` (the nodes which have even ID are active), `'odd'` (the nodes which have odd ID are active) and `'customize'` (active nodes are defined by the user). The input of this example is completed by selecting `'even'` (lower left of Figure 3). The user who wants to define another condition can select `'customize'` and input the sentence of the definition in the dialog (this dialog also appears when user selects `'1 node'`). The system allows users to input the constant, the variable, the four basic operations of arithmetic and logical operations (ex. $((id/2)\%2) == 0$: id is the variable of own node ID). Definition of the condition can be executed not only after the input of communication relationships but also before that. Without the definition of the condition, the system judges the condition as "all active." In the case of Gather, it is pos-

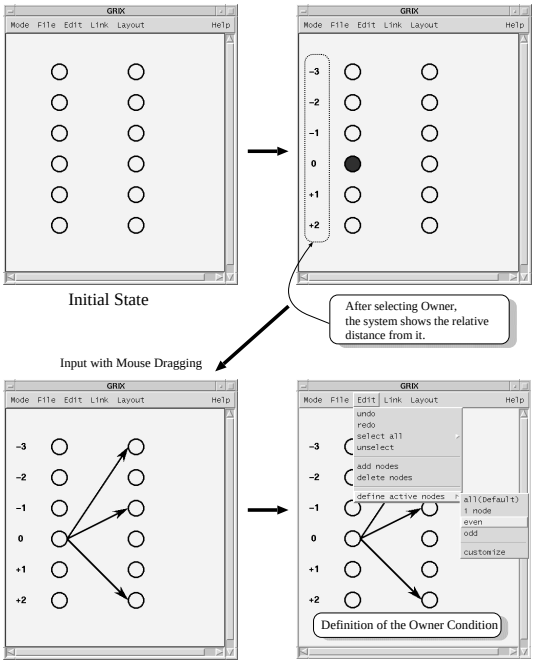

Figure 8: Input with Relative Viewpoint

sible to input it by selecting Owner from the receive nodes. In the case of the input with this viewpoint, the system does not allow users to describe the arrow which is not connected with Owner. The advanced operations like the case of Absolute Viewpoint can be used while keeping that restriction.

4 Optimization

GRIX optimizes the inputted communication relationships in order to improve the efficiency. The concept of optimization is to make the communication pattern which is *"fast to communicate"* and *"easy to generate"* the actual code. The principal jobs of Optimization Engine are as follows.

- Removing communication conflicts
- Reducing the number of communication by memory copying
- Generating conditional sentences of active (or inactive) nodes

The detail of those optimization is shown in bibliography [1, 2].

5 Code Generation

GRIX involves the subsystem which generates the actual code automatically in order to deal with the recent various environments, like PVM and MPI. This code generation removes coding with the knowledge of those specifications and enables the programmer to write the code with the intuitive manner. Moreover it is possible to acquire the code for various environments with their corresponding translation engines. We have prepared the code generation subsystem for PVM at first. For instance, the actual code of Figure 8 which is the input with Relative Viewpoint is following Table 1.

The followings are simple explanations of variable names written in the program.

- `SPMD_nprocs`:the number of Virtual Processors(VP)

Table 1: Actual Code with Relative Viewpoint

```
int stride[3] = "-3, -1, 2";
int skip = 0;

/*---------- SEND ----------*/
if ((SPMD_procnum % 2) == 0){
   for (i = 0; i < 3; i++){
       pvm_pkbyte(&send_area, datasize, 1);
       pvm_send(SPMD_tid[(SPMD_procnum +
                          SPMD_nprocs + stride[i])
                         % SPMD_nprocs], i);
   }
}
/*---------- RECV ----------*/
for (i = 0; i < 3; i++){
    if ((sender = SPMD_procnum - stride[i]) < 0)
        sender += SPMD_nprocs;
    else if (sender >= SPMD_nprocs)
        sender -= SPMD_nprocs;
    if ((sender % 2) == 0){
        recv_area += datasize * skip;
        pvm_recv(SPMD_tid[sender], i);
        pvm_upkbyte(&recv_area, datasize, 1);
        skip++;
    }
}
```

- `SPMD_procnum`:the VP number of myself(zero origin)
- `SPMD_tids`:the array of tid those indexes are VP numbers

6 Related Works

GRIX is a new GUI tool for parallel computing because we have implemented it specified for the input of interprocessor communications. There exists some GUI systems for parallel computing. P. Newton and J. C. Browne propose *CODE* [6], which is a visual parallel programming language. G. A. Geist proposes *HeNCE* [8], it is also the visual parallel programming language. Newton has compared their characteristics in the bibliography [7]. Those visual programming languages force users to write the program on their GUI windows. The visual input for the sentences which is not redundant, like the normal sentence of basic operations `C[i,j] = C[i,j] + A[i,k] * B[k,j];` , has some waists. We insist that the visual input for

the redundant points is most effective and the redundant point in the parallel program is the sentence for interprocessor communications.

XPVM [9] is the GUI system in order to control and visualize the PVM programs. GRIX and XPVM have same characteristics on the point of controlling interprocessor communications. But GRIX is not aiming only on PVM but also the multiple plat home for executing interprocessor communications. Performance monitors which visualize the load performance of each processor, are also the GUI systems for parallel computing. *ParaGraph* [10] is the one of performance monitors. Though performance monitors are only for output, GRIX has the functions of both input and output.

7 Conclusion

There exists lots of implementation methods to code interprocessor communications and they shows a tendency to be complex. Visual programming method has large possibility to make such a difficult programming more smooth. We have proposed the GRIX system, the visual programming system for making the programming of interprocessor communications less difficulty. Because of the GUI system, GRIX is the efficient environment to realize the faster interprocessor communications. Adopting the pictorial implementation, we do not need to use complex texts to the implementation. We can apply intuitive graphical charts to the implementation. GRIX can release programmers from complicated works without being familiar with a lot of complex procedures about interprocessor communications.

GRIX will be more effective and more interactive system by jointing the existing textual editors.

References

[1] Y. Sakayori, M. Miura and J. Tanaka, GRIX: Visual Programming System for Interprocessor Communications, Proceedings of the 10th IASTED International Conference PDCS '98, pp.503-508, Oct. 1998

[2] Y. Sakayori, GRIX: The Proposal of Visual Programming System Specified for Interprocessor Communications, Master Thesis, University of Tsukuba, Feb. 1999 (in Japanese, English extended abstract is available)

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck and V. Sunderam, PVM, The MIT Press, 1994

[4] W. Gropp, E. Lusk and A. Skjellum, USING MPI, The MIT Press, 1994

[5] B. A. Mayers, Taxonomies of Visual Programming and Programming Visualization, Journal of Visual Language and Computing, 1(1):97-123, 1990

[6] P. Newton and J. C. Browne, The CODE 2.0 Graphical Parallel Programming Language, Proceedings of ACM International Conference on Supercomputing, July, 1992

[7] P. Newton, Visual Programming and Parallel Computing, Delivered at Workshop on Environments and Tools for Parallel Scientific Computing, May, 1994

[8] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, Graphical Development Tools for Network-Based Concurrent Supercomputing, Proceedings of Supercomputing 91, pp.435-444, 1991

[9] G. A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam, PVM 3 Users Guide and Reference Manual, Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, 1993

[10] M. T.Heath, ParaGraph: A Tool for Visualizing Performance of Parallel Programs, Univ of Illinois, Jennifer Etheridge Finger, Oak Ridge National Laboratory, June 1994