Abstract: - Since scientific applications usually manipulate a very huge volume of data, parallel computing is the best choice for solving this kind of applications. However, making parallel programs is not an easy task for programmers. They need to have a good tool or environment wherein they can easily specify what they want to solve. Because of the complexity in making parallel programs, visual programming is a very promising approach in this area. This paper describes a system that allows the creation of programs from algorithmic “film” specifications, where the use of text is minimal. In this system, a language of micro-icons is used to specify any collective operation and this paper shows how this visual programming environment supports the manipulation with these micro-icons. Because of space constraints, we limit the example to a broadcast operation. Using the same technique, it is possible for other types of collective communication operations to be specified in this visual programming environment.

Key-Words: - visual tool, collective operation, broadcast operation, master/slave, parallel computing, algorithmic film

1 Introduction
Since scientific applications usually manipulate a very huge volume of data, parallel computing is the best choice for solving this kind of applications. However, it can not be denied that making parallel programs is not an easy task for programmers. They need to have a good tool or environment wherein they can easily specify what they want to solve. Visual programming is a very promising approach in this area because of the complexity in making parallel programs. In fact, there were researches made on visualizing parallel and/or concurrent programs [21, 18], animating or visualizing parallel algorithms [11, 21], visualizing parallel software execution [21], performance evaluation and optimization of parallel programs [21], memory access behavior of parallel programs [22], etc. Of course, visual approach is also used in sequential programming. In fact, visual approaches are also being considered in different types of applications such as in object-oriented databases and programming [8], SQL databases [3], XML queries [6], multimedia applications [7], etc.

This paper describes a small portion of a system that allows the creation of programs from algorithmic “film” specifications [10, 24, 5], where the use of text is minimal. Algorithmic films of a multiple view format are used in which the different parts can be specified or viewed in a non-linear order according to user's demands. In this new programming technique using such films, a programmer can browse, watch, edit, and specify or attach operations to film frames inside some panels. When creating a program, he will just be combining some films from a database suitable to his needs and updating the operations within the chosen films. The source code (C+MPI) will be generated automatically by the system using the templates from a database [5].

To illustrate the features of this system, the specification of a broadcast operation based on the master/slave scheme of computation is presented. This operation is attached to a master/slave film using a language of micro-icons. Because of space constraints, we limit the example to a broadcast operation. Other MPI collective operations using the same technique are also possible. The system provides an open set of micro-icons that are intuitive enough for understanding even for a novice programmer. As the number of micro-icons grows, a programmer may need some aid in understanding the functions of the different micro-icons. So the system should be ready to provide an effective help in the form of animation, sound, and text.

This paper presents an approach that aims at providing a visual support for the existing MPI library designed for collective communication operations. In Section 2, some works which have some relations to this research are described briefly.
In Section 3, an overview of the entire system called “Active Knowledge Studio” (AKS) [10, 24, 5] is given. The specification of a broadcast operation is illustrated in Section 4. In Section 5, the discussion about source code generation is given. Finally, this paper is concluded in Section 6 with statements about the future direction of this work.

2 Related Works
A lot of researches have been done to provide an environment where making (sequential or parallel) programs will be made easy. Even at present, many researchers are still studying on how to make programs from visual symbols in order to facilitate software constructions. The following paragraphs describe briefly some of the systems that use visualization as a technique to make parallel programs easy to create and their computational activities easy to understand.

Visper [20] is a graph-based visualization tool designed for making parallel message-passing programs. It uses a formalism called Process Control Graph (PCG) [19]. It has a user-friendly PCG visual editor for creating programs. In creating a program, a programmer draws some graphs that will be annotated by filling in a set of forms, e.g., ExecuteForm for execute blocks. Execute blocks are used to add some codes to a PCG and to describe how sequential components are composed into a parallel construct. The created graphs can be compiled to automatically compose the program. This is quite close to our approach in the sense that it tries to minimize the typing of text as in the case of filling in a NodeForm. However, in some cases like defining some sequential computation to be executed by the (all or exclusive) execute blocks, they are entered as text just like making modules in a conventional programming language. Thus they claim that programming in PCG is not visual in all aspects [19].

VPE [14] is another visual parallel programming environment, which is intended for providing a simple human interface for creating message-passing programs. In making programs, a programmer specifies the parallel structure of his programs by drawing pictures. VPE computations are represented as graphs where nodes represent processes and arcs represent the paths for messages to flow from process to process. Each graph contains computation nodes, which will be annotated with program text expressed in C or Fortran that contains simple message-passing calls. It is also not visual in all aspects.

Other related visual systems for making parallel programs include systems like HeNCE [2], CODE [15, 13], Paralex [1], GRAPNEL [9], DIVA [16], Meander [23], and VADE [11] to name a few. Just like the ones mentioned above, these systems do not fully provide an environment where programming will be done by greatly minimizing the use of text in constructing parallel programs.

The visual programming environment presented in this paper uses algorithmic films instead of graphs as the major components of a program. An overview of this system is given in the next section. In general, the use of text is minimized and the use of micro-icons in programming is maximized.

3 An Overview of Our System Called Active Knowledge Studio
“Active Knowledge Studio” (AKS) [10, 24] is our on-going research in the area of “Self-explanatory Component” technology. In AKS a programmer creates a program simply by combining some algorithmic films from a database. He does not have to create a film from scratch. He can browse some films provided by the system and view the desired groups of features. If a film is appropriate for his needs, he can edit it to suit his programming needs. He can combine different films to meet his computational needs. A program may, therefore, consist of one or more films depending on its complexity. A film may consist of one or more scenes for achieving some tasks. A scene is a set of stills or frames performing some related tasks. A still is a single step of a computation and is the smallest item in a film where the programmer specifies or defines an operation.

A typical operation in a master/slave scheme of computation can be used to illustrate when to group the stills into a scene. The slave process usually performs at least three things. It receives data from the master process, performs some computation, and sends the results to the master process. We can consider these three activities as having three scenes. Scene 1 will be used for the operation of receiving data. Scene 2 will be used for computation. Scene 3 will be used for the send operation. Depending on the kind of operation and the algorithm used, a scene may consist of one or more stills. For example, if the slave process must first receive a row from the first matrix and the entire second matrix before doing any computation, then we can consider these two receive operations as one scene.

When using films, different operations are defined in a still. This paper presents an example of attaching a broadcast operation to a still of a film. A language of micro-icons is used to accomplish this. A panel
designed for a certain operation is provided. It contains some buttons with example micro-icons that will give the programmer an idea of the function behind the micro-icons. He will just click on any of the buttons that he thinks necessary to be changed.

One may wonder if it is possible to specify computation without using text at all. So far it is impossible but can be minimized. We use a special kind of text, which is in “enhanced” format [10, 24].

4 Specifying a Broadcast Operation
This paper presents how to specify a broadcast operation to a still of a film. It is assumed that the programmer has already selected a master/slave film. He can now navigate on the scenes by clicking on a particular micro-icon for a particular scene of interest from the display of scenes. The corresponding still(s) of the chosen scene will be displayed inside the area for displaying stills. The first still in a scene is the default still that will be enlarged and displayed on the upper left corner of the Edit window.

Fig. 1 shows a typical Edit window where the programmer can navigate some films using the displayed buttons for scenes and stills. In this figure, some functional buttons are shown at the upper right area of the Edit window. The lower right button is used when a programmer wants to attach an I/O operation or communication among processes in a parallel program. The desired still should be selected first before clicking the lower right button of the functional buttons found at the upper right area of the Edit window. With reference to the current still display, the big square in the middle represents the master process while the small squares below represent the slave processes. The circles enclosed in a round-edged rectangle represent the data.

![Fig. 1. The edit mode window.](image)

Fig. 2 shows an example specification panel for defining a broadcast operation. The panel is supplied with an example micro-icon for each button with replaceable micro-icon. To enhance the intuitiveness of a micro-icon, text is also used. The convention is to use italics font when it is part of a micro-icon but normal font when used as a variable name. For example, in Fig. 2 “BARRIER” is italicized because it is part of the micro-icon while “SBUFF” uses the normal font because it is used as the name of a variable that contains some data. In general, the example micro-icons are more or less sufficient, thus only a simple fine-tuning is necessary.

The appearance of the panel varies according to the number of the needed buttons to specify an operation. Fig. 2 contains three (3) areas, the center of which is for specifying the main operations. The first area (left) contains a node indicator signifying the type of a node [24, 25] to which the operation is defined. As shown in the figure, the node indicator is a micro-icon that is made up of a single square. It means that the definition contained in the panel is a specification for an activity defined for a single node. The color of the node indicator depends on the color of the node(s) to which the given definition is attached. During the execution of the program, the chosen node(s) will be flashed in some way depending on the activity of a given still [5, 25]. For a definition attached to a set of nodes, a different micro-icon is used. In that case, the micro-icon shows a set of squares rather than a single square. The system uses two shapes of nodes: one is for ordinary nodes represented as circles and the other one, which calls some functions or films, represented as squares. The system automatically shows this node indicator based on the available information like the type and color of the nodes.

The second area (center) of the specification panel is used to define the main operation. This part is called the Main Definition Area (MDA). This area contains two non-replaceable micro-icons (with white background) and two sets of buttons with replaceable micro-icons (with gray background). In Fig. 2, the upper non-replaceable micro-icon specifies a broadcast operation. It portrays a configuration of a set of processes depicted as small boxes
aligned vertically. When processes are involved in a given operation, their color is black while those that do not participate, their color is white. As indicated by the arrow, the left boxes mean the initial configuration and the right boxes signify the final configuration of processes after the execution of an operation. In the example, the initial configuration shows only one small box with a black color. This represents the process that performs the broadcast operation. From this sending process, there are lines connected to all boxes at the right side of the micro-icon. All lines have an associated black square signifying data. This is so because in a broadcast operation, the same data is sent to all processes.

The other non-replaceable micro-icon shows the type of structure being used. The example used in Fig. 2 shows a master/slave scheme. The presence of this micro-icon is not very important in the specification panel under the edit mode because the programmer can see the current structure being enlarged and shown at the upper left corner of Fig. 1. However, edit mode view is just one of the views supported by the system. Animation and tile views are supported as well [17]. These other views help the programmer to easily view his set of specifications several frames at a time. When using these other views, he may not see the Edit window shown in Fig. 1. Thus the presence of this micro-icon helps him to identify easily the structure being used, especially if there are many types of structures used in a complex system. To balance with what he can see in the edit mode as well as in other modes, the structure is also included in the specification panel while under the edit mode.

The next discussion will be for the first two buttons with replaceable micro-icons. These micro-icons belong to the Source Definition (SD) group. The upper button contains a micro-icon showing the name “SBUFF” (send buffer), whose structure is a 2-D type containing double-precision floating point numbers. If a programmer wants to change the given example micro-icon, he can edit this, e.g., to another dimension, structure, or data type. In MPI [12, 4] the buffers are usually 1-D arrays but in real application, other dimensions are also used and manipulated. So when using multi-dimensional arrays, they will first be converted into a 1-D array before passing it to an MPI function. Another solution would be to declare a new data type because MPI supports it.

The lower button of SD contains a text label “INFO.” This button specifies the name of the structure that hold the pieces of information that include at least the type of data and its size or counts, which can be set by the system automatically. This button also includes the information whether a single process called root or members of a group are the source of the given operation. In case of a root process, it will also store its rank. In case of a group, it will also store the identity of its communicator. These pieces of information are combined to simplify the design of the specification panel in AKS.

The next group of buttons belong to the Target Definition (TD) group. The upper button contains a text label “COMM,” that indicates the name of the group of process to which the data will be sent. The lower button specifies a barrier synchronization. The micro-icon shows an image of some synchronizing processes before (to the left) of the clock image symbolizing time. Since the barrier is to the left of the clock image, it means that a barrier must take place before the operation defined in the panel will be executed. It can be changed to another kind of barrier that synchronizes all processes after the operation defined in the panel is executed or to no barrier synchronization at all.

The third area (right) of the specification panel contains two functional buttons, which are not grouped together. The top button contains the text label “Edit.” When this button is clicked, the label changes to “Help.” Thus this button is called “Edit/Help” button. When the “Edit” label is shown, clicking the buttons with replaceable micro-icons both in SD and TD groups results to showing some options where a programmer can change the example specification. When the “Help” label is shown, clicking any of the buttons in MDA results to showing the system help in the context of the clicked micro-icon. This includes the non-replaceable micro-icons. The system help comes in three types: animation, sound, and text just like the one shown in Fig. 3. The micro-icon itself is very intuitive but if a programmer wants to make sure of its meaning or
function, he can invoke the system help easily. This approach is geared towards the goal of providing self-explanatory components. In fact this is just a very small item designed in order to help in achieving self-explanatory components.

The button below the “Edit/Help” button is used to record the specification found in the panel. This is called the “Set/Reset” button because it toggles between “Set” and “Reset” buttons when clicked. When the system is in the “Set” mode, clicking this button means the programmer wants to save his specification and the corresponding source code will be created automatically without resorting to text-based programming style. When the button is in the “Reset” mode, all buttons in the SD and TD groups with replaceable micro-icons are not active and cannot be replaced. This means that the “Edit” mode has been disabled. Of course, the “Help” mode is enabled and it is still possible to view some animation, hear sound, or read text.

...MPI_Comm COMM;
int X_dim, Y_dim;
int gsize, ROOT, myrank;
double *SBUFF;
...
MPI_Comm_size(COMM,&gsize);
MPI_Comm_rank(COMM,&myrank);
SBUFF = (double *) malloc(X_dim*Y_dim*sizeof(double));
...
convert2Dto1D(&sbuff,X_dim,Y_dim,&SBUFF);
MPI_Barrier(COMM);
MPI_Bcast(&SBUFF,INFO->count,
INFO->type,INFO->ROOT,COMM);
...

Fig. 4. The broadcast operation source code.

5 Source Code Generation
Source codes will be generated automatically by the system using some templates from a database. Our system actually contains six subsystems for implementing the six different groups of frames that are being considered. Each subsystem has its own set of templates and the code generation subsystem will collect the different source codes generated by these subsystems to make a whole program. During the collection of different source codes from different subsystem, the code generation subsystem will try to perform some consistency checking. If no conflicts are found, it then proceeds with the generation of an executable code.

This section describes the source code generation part of the subsystem for specifying I/O operations and communication among processes in a parallel program. After specifying a particular kind of operation by replacing some micro-icons shown as examples in the panel, a programmer can record his set of specifications using the “Set” button provided in the panel. During this operation, this subsystem will also perform some simple consistency checking to make sure that there will be no conflicting definitions. From a template database, this subsystem proceeds to generate a source code, like the one shown in Fig. 4. For the implementation of the message-passing type of communication, the existing C version of the MPI [12, 4] library is used. For more information about the code generation subsystem of “Active Knowledge Studio,” please see [5].

6 Conclusion and Future Work
A visual tool for specifying MPI collective communication operations using the message-passing paradigm has been presented. It used the master/slave scheme of computation as the model to illustrate the specification of the example broadcast operation. Because of space constraints, only broadcast operation was used as an example. This paper presented an approach that aimed at providing a visual support for the existing MPI library designed for collective communication operations. At the same time, the example illustrated a different technique in making parallel programs using algorithmic “films.” In this technique, a programmer navigates the films either by scenes or by stills and manipulate the stills of a film inside some specification panel by attaching operations using a language of micro-icons. Each panel is supplied with example intuitive micro-icons that will serve as guides to the programmer. Moreover, if the intuitive meaning of the micro-icons is not enough, the system provides a simple to understand help that comes in three forms: animation, sound, and text. A programmer will just be performing a simple fine-tuning of the example micro-icons to specify any collective operation.

What we have presented is just a part of our on-going project called “Active Knowledge Studio.” There are still many things to be done in the very near future. Once the entire system is fully operational, some experiments will be conducted such as evaluating its effectiveness as a tool, determining whether or not this can be used in making real complex scientific applications, and validating whether or not programmers really find this environment easy to use and understand.
References:


