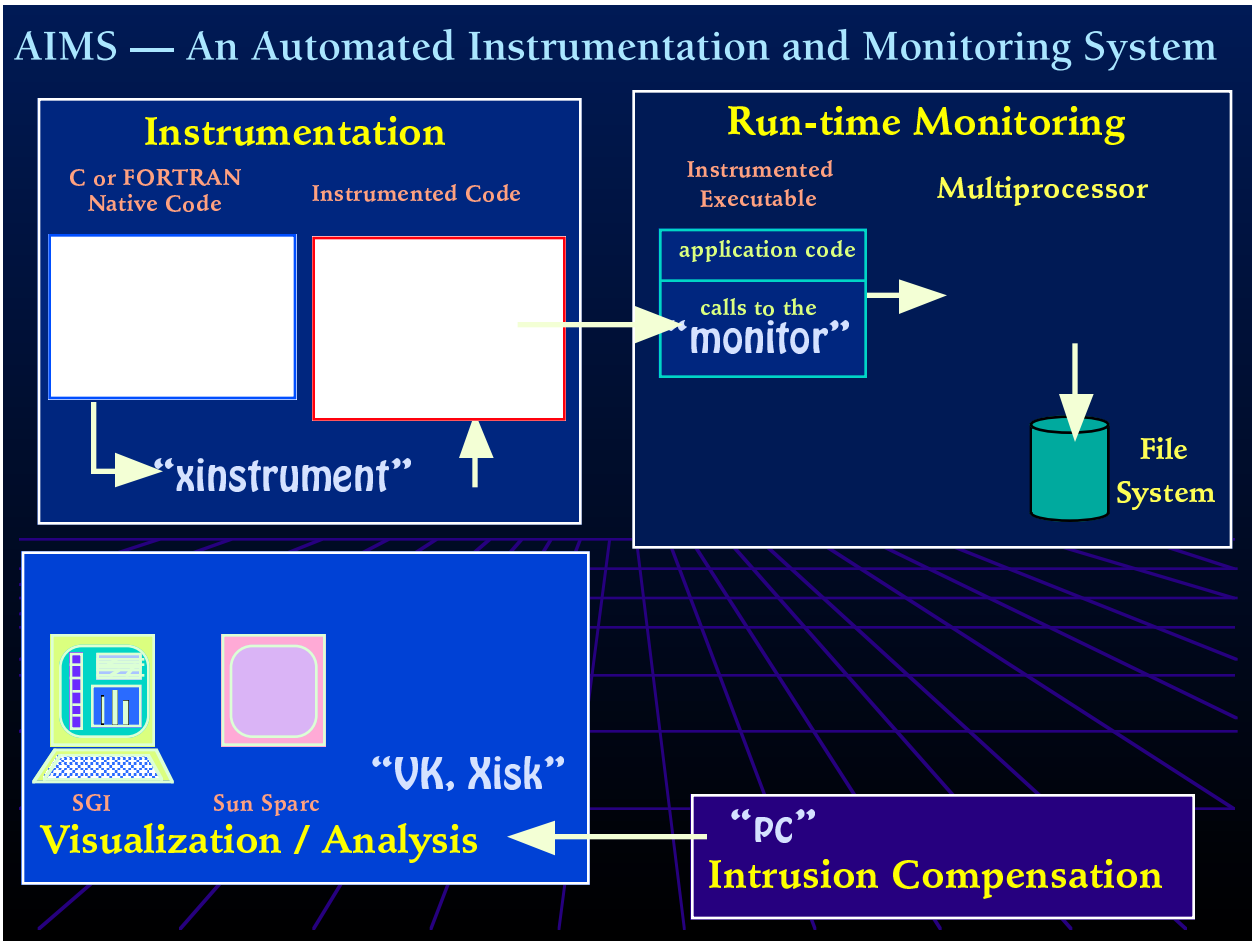
1. Overview of the Automated Instrumentation and Monitoring System

Writing large-scale parallel and distributed scientific applications that make optimum use of computational resources is a very challenging problem. Very often, resources can be under-utilized due to performance failure in the application being executed. Performance tuning tools are essential to expose these performance failures and suggest how program performance should be improved.

AIMS, an Automated Instrumentation and Monitoring System, consists of a suite of software tools for measurement and analysis of performance of FORTRAN and C message-passing programs written using the NX, PVM or MPI communication libraries. As shown in Figure 1-1, the AIMS tool kit includes:

- A source code instrumentor, `xinstrument`, inserts performance monitoring routines into the application. The user may selectively control the instrumentation of the source code.
- A runtime performance monitoring library, or `monitor`, provides a set of monitoring routines that measure and record various aspects of program performance, such as message-passing overhead, synchronization overhead, and time spent in subroutines.



Running AIMS consists of the three basic steps shown in Figure 1-2: instrumenting the source code, compiling and running (monitoring) the application, and analyzing the trace file using various analysis tools.

1. Instrument the source code. (Chapter 2)
2. Monitor the application (Chapter 3)
   - Copy other files not instrumented (including makefiles, include files, input files) into the `inst/` directory.
   - Build the instrumented source code.
   - Run the application
   - Sort the trace file (Intel iPSC/860 and Paragon only)
3. Analyze the trace file using the analysis tools. (Chapters 4-6)

Figure 1-2: Using AIMS in Three Basic Steps.

Instrumentation is the process of adding monitoring routines into the application for performance data collection. `xinstrument` regards the source code as a nested collection of constructs, which are logical entities of the program, such as conditionals, loops, subroutines, or communication calls. `xinstrument` enables users to selectively instrument only source code that is of interest to them. In addition to inserting instrumentation at appropriate locations in the application, `xinstrument` generates two important files: an application database and an instrument-enabling profile. These files, and the instrumented source will be used during run-time monitoring. `xinstrument` will be discussed in Chapter 2.

After instrumentation, is the monitoring phase, shown in Figure 1-3. The user must compile the application in order to build the instrumented source code containing event recorders that were inserted during instrumentation. The user links the instrumented source code with the `monitor` to create the instrumented executable. When the application is executed on a multiprocessor, the event recorders write records into memory buffers at each processing node. The contents of these buffers are intermittently written (or flushed) to disk when the buffer becomes full. This creates a trace file containing time-stamped events for each occurrence of an instrumented construct. Because writing the buffers to disk is time consuming, the frequency of file flushing can be controlled by the user via selection of different buffer sizes. (See Section 2.4 on the `xinstrument` Settings Pane.) Alternatively, changing the Monitor Mode from Trace to Statistics in the Settings Pane can be used to generate only summary statistics. With this option turned on, during execution of the instrumented code, an aggregate time will be calculated for each instrumented construct, and a count will be made of the total number of times it was executed. This will result in a much smaller file than a trace-file.

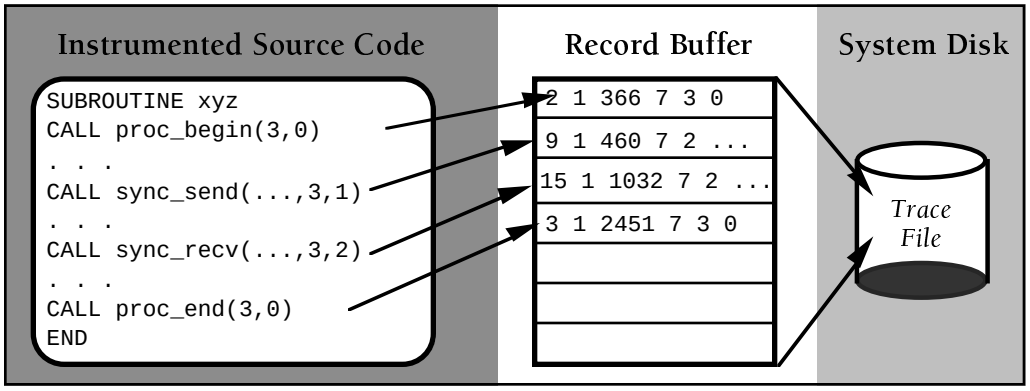


Figure 1-3: AIMS' monitoring phase. Instrumented code is executed and writes buffers to system disk.

The `monitor` performs other important tasks The AIMS clock skew mechanism corrects for differences in workstation clocks, and the cost model generation calculates, by experiment, the latency and cost per byte for sending messages in the parallel virtual machine. Both of these are extremely useful when trying to present the most accurate order of events and are done transparently by the `monitor`. In addition, the `monitor` is responsible for remembering the name of the machine that ran a task, converting PVM task identifiers to node numbers, and sorting the trace file.

After the application has terminated and the trace file is saved, the user can analyze the performance of an application by using various tools which AIMS provides. VK's displays present information such as:

active subroutines, disk accesses, messages sent between nodes; and the route messages travel in a network of workstations. Some displays scroll along as time passes, showing a segment of the program's history, while others animate each state in sequence. Several displays can be viewed at once. The trace file can be stepped through or visualized at high speed, stopping only when certain subroutines are invoked. A source code click-back capability allows easy examination of the source code corresponding to selected events on the display.

Another tool, `tally`, provides statistical summaries and indices that describe program behavior. `tally` generates a list of resource-utilization statistics on node-by-node and routine-by-routine bases. The routine statistics give information typically provided by profilers with respect to amount of time spent in various functions. In addition, it provides easy access to the percentage of execution time spent communicating and the significance of the communication time in comparison with the total program execution time. The statistics can help to quickly determine the sections of code which needs to be tuned. The output of tally can be used as input to statistical drawing packages such as Excel and WingZ.

A tool for mapping out the physical network configuration of the network is `sysconfig`, described in section 6 of this manual. Additional information about AIMS may be found in the papers listed in Appendix A. An installation guide may be found in Appendix B.

2. Instrumentation

`xinstrument` allows the user to select specific source code constructs to be instrumented. When a construct is instrumented, it is actually replaced or surrounded by calls to AIMS' monitoring routines. Each instrumented construct generates a time-stamped event when executed. Typical constructs to be instrumented include message sends, receives and synchronizations. In general it is useful to instrument all communication calls and the subroutines that do useful work. Subroutines that are called often without making a significant contribution (e.g., random number generators) are generally not useful to monitor and will produce many uninteresting events in the trace file. However, it would probably be useful to monitor routines like matrix multiplication. It is important to remember that if a routine is uninstrumented, trace file analyzers will be unable to provide any insight into the routine's activity. Time spent in uninstrumented routines is lumped together with the entire application's execution time. Users usually find this acceptable for routines that are just called as initialization in their application.

Some users who want to focus inside a routine will want to instrument loops and conditional statements. This will generate a separate event for each iteration of a loop or pass through a conditional. This can result in many events being generated, so it is important to estimate how many times a statement will be executed so that you don't write to disk every second, or fill up your file system with trace data. It is recommended that you start with the defaults and increase the detail as you need it and as you become more familiar with `xinstrument`.

2.1 Invoking `xinstrument`

`xinstrument` should be invoked from the directory containing the source code. It is invoked in one of the following ways:

```
xinstrument
xinstrument   -help                          provides help information
xinstrument   -adb <application database>    allows specification of an
                                             existing application database file
xinstrument   -overwrite                     allows existing instrumented files
                                             to be overwritten
```

Invoking `xinstrument` will bring up the display shown in Figure 2-1. The `xinstrument` display consists of three sections, or panes: (1) the Status Pane on the upper right, (2) the Settings Pane on the lower right, and (3) the Construct-Tree Pane on the left. The Status Pane informs the user as various actions (such as file loading or instrumentation) are taken by `xinstrument`. The Settings Pane displays various AIMS parameters (e.g., where instrumented files are to be written). The Construct-Tree Pane provides a

structured view of the application and allows selective instrumentation. Pull-down menus at the top allow various aspects of the instrumentation process to be controlled.

Once `xinstrument` has been invoked, the Settings Pane shown in the lower right corner of Figure 2-1 can be used to customize the monitor parameters. Users may take the defaults shown in the Settings Pane when it is brought up, or they may make other choices by selecting the appropriate button or typing the appropriate value. For example, for the Monitor Mode, the user may select either the Trace Mode (for a trace file) or Statistics Mode (for summary statistics). In the case of the Trace Mode, the buffer size can be set to minimize flushing. The name and directory location of the output trace file may also be changed. The options for the Settings Pane are shown in Table 2-1.

2.2 Using `xinstrument`

There are three components to the pull-down menu: Files, Options, and Advanced Options. First choose Load Modules from the Files menu. (See Table 2-2 for the possible actions from the Files menu.) A dialog box, like the one shown in Figure 2-2, will be brought up to help select the files to be loaded. It is important to specify the language and platform for the selected files before loading. If the platform shown is incorrect, use the pull-down button to view alternate selections.

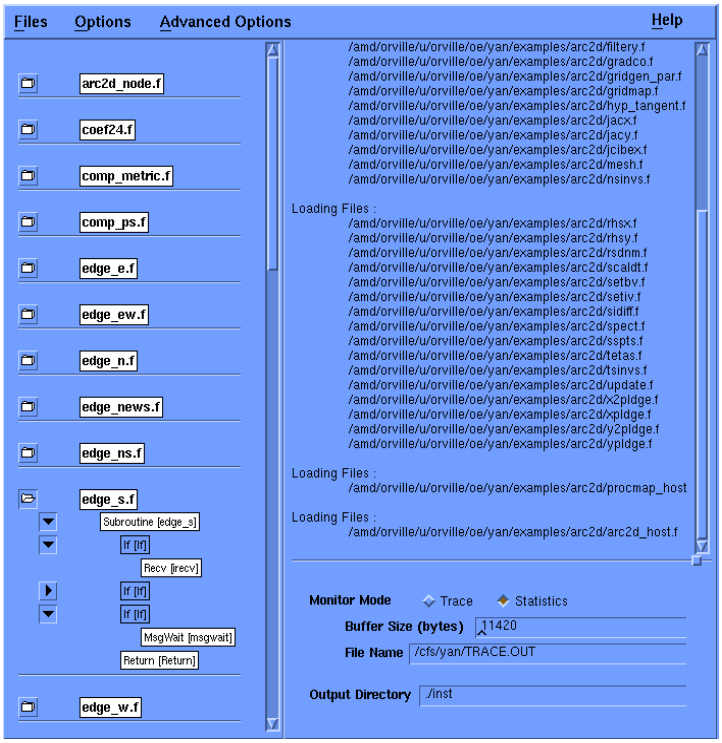


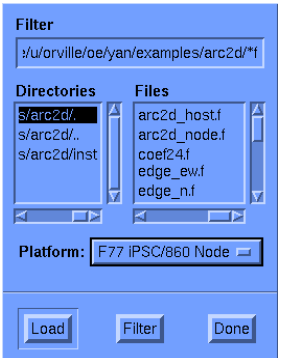
Figure 2-1: Display for controlling `xinstrument`.

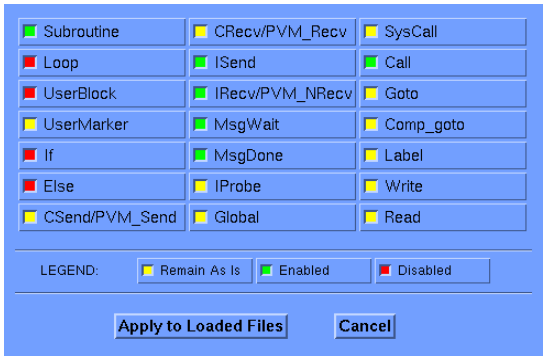
When the platform is correct, select the files to be loaded. Files to be loaded should be FORTRAN or C source code and may contain a main program, a subroutine, multiple subroutines, or a combination. A file containing the main program must be loaded. If necessary, select the appropriate directory by clicking on it or enter a new one in the Filter box and click the Filter button. Then select a file to be loaded by clicking on it. Multiple adjacent files may be selected by holding down the mouse button and dragging it across the files. Files may be deselected by clicking on them a second time. After selecting the files to be loaded, click the Load button in the dialog box. After the files are loaded, click the Done button on the dialog box.

Table 2-1: Summary of options for the Settings Pane.

Field	Comments
Output Directory	Indicates where instrumented files are to be written. By default, `./inst` is used.
File Name	Indicates where the trace file is to be written. By default, TRACE.OUT is used. All nodes of the multiprocessor must recognize the pathname specified here.
Monitor Mode	Setting Monitor Mode to Statistics (as opposed to Trace) generates only summary statistics (as opposed to a lengthy trace file of events).
Buffer Size	If Monitor Mode is in Trace, buffer size is user selectable. If Statistics mode is used, Buffer Size indicates the amount of memory needed by the `monitor` to store the summary statistics and is not selectable.

The next step is to decide on the instrumentation to be done.[1] To take the default instrumentation (all communication constructs), select Instrument All Modules. After the modules are instrumented, select Exit (from the Files menu). Alternatively, use the Options menu (Table 2-3) to select either All Subroutines, All Communications Constructs (the default), All I/O or Enable by Type. If Enable by Type is selected another dialog box (like the one shown in Figure 2-3) will be brought up to enable instrumentation by construct type. Click on the appropriate buttons to enable or disable the instrumentation of the constructs. The constructs selected will be instrumented in all loaded files. After selecting the instrumentation, choose Instrument All Modules from the Files menu. Finally, select Exit from the Files menu.





Figures 2-2 and 2-3: Dialog Boxes for File Selection and Enable by Type, respectively.

Table 2-2: Files Menu Selections.

Files	Actions
Load Modules	As shown in Figure 2-2, a dialog box for file selection is brought up. One or more files may be selected simultaneously. It is important to specify the language/platform (C or FORTRAN) for the selected files before loading. Click the Load button (or press return) to load selected files. Although not all of the application's files have to be selected, the file containing a "main program" must be loaded. Click the Done button after files are loaded. Loaded files appears in the Construct-Tree Pane.
Instrument All Modules	This instruments all files loaded. Individual modules that are selected may be instrumented under the Advanced Options menu.
Exit	User quits `xinstrument`.

Table 2-3: Options Menu Selections.

Options	Actions
All Subroutines	Instruments entry to and exit from all subroutines.
All Communication Constructs	Instruments all sends and receives. This is the default.
All I/O	Instruments all I/O activities (`read()`, `write()`, `io_wait()`, `l_seeks()` etc.)
Enable by Type	As shown in Figure 2-3, a dialog box is brought up allowing customized control for instrumentation to all loaded files. Each type may be enabled, disabled, or left as is. Click Apply to Loaded Files to execute the selection. Click Cancel to exit without affecting current instrumentation settings.

[1] Some versions of AIMS may not implement all communication constructs.

More selective instrumentation may be obtained using the Advanced Options menu (Table 2-4) along with the Construct-Tree Pane (left side of Figure 2-1). Using the Instrument Selected Modules options from the Advanced Options menu allows users to instrument only selected constructs in the specified files. Otherwise, the instrumentation applies to *all* constructs of a specified type in *all* files. Point-and-click actions on the Construct-Tree Pane are used to select or deselect constructs to be instrumented. Constructs that are available for instrumentation are indicated by a surrounding box. When a construct is selected, it is highlighted. Files and constructs within a file may be selected. Constructs may also be selected using the dialog box brought up by the Enable Selected Modules by Type option. The various icons and the actions and effects they represent are summarized in Table 2-5. Holding down the shift key and clicking with the left mouse button on any displayed construct brings up the corresponding source code in a pop-up window.

Table 2-4: Advanced Options Menu Selections.

Advanced Options	Actions
Instrument Selected Modules	Instruments files selected in the Construct-Tree Pane.
Enable Selected Modules by Type	An Enable by Type dialog box is brought up and applied to files selected in the Construct-Tree Pane.
Remove Selected Modules	Unload files selected in the construct tree pane.
Profile	Load or Save profiles.
Monitor File	Load or Save `AIMS.monrc` files.
Set Preprocessor Options	A dialog box is brought up, and preprocessor options for the parser, such as `-D` flags can be simply typed in.

Table 2.5: Summary of Options for the Construct-Tree Pane.

Icon Displayed	Icon's Meaning	Effect of LEFT-Click	Icon after Clicking
	A source file	expands file to reveal one level of detail	
	An opened source file	hides contents of the corresponding file	
	A construct with sub clauses	reveal one more level of sub-clauses	
	An opened construct	hides sub-clauses	
	An instrumentable construct (e.g. `MsgWait`)	instrument this construct	

2.3 Files created by `xinstrument`

The following files are created by `xinstrument` and placed in the `inst/` or designated output directory:

- Instrumented source code.
- AIMS.monrc file containing parameters shown in the Settings Pane and additional, more advanced options that can only be changed by editing the file directly.
- The application database (default name is APPL_DB) that stores information about the static structure of the application. The analysis tools use this information to relate traced events to instrumented constructs. This database file is subsequently incorporated at the beginning of the trace file produced by executing the instrumented application.
- The instrument-enabling profile file that is a table of flags, one for each construct in the application database. This profile is useful when a user wants to change instrumentation on a source file. The profile can be used as a "saved" instrumentation default for an application.

3. Compiling and Running Instrumented Code

`xinstrument` writes instrumented source files into the Output Directory (default `inst/`) specified in the Settings Pane. This directory should have the same structure as the original, therefore it may be necessary to copy over files such as include files, makefiles, and files that were not instrumented. Afterwards, the `inst/makefile` must be augmented to link the AIMS monitor library.

Figure 3-1 shows an example source directory and the steps needed to make the instrumented source directory identical to the original source directory. The directory containing the application project is shown on the left of Figure 3-1. The files `main.c`, `sub1.c`, `sub2.c` and `sub.dir/sub3.c` were selected to be instrumented. After instrumentation, a new directory `inst/` was created in which `xinstrument` placed four instrumented files, one subdirectory, and the files `AIMS.monrc` and `APPL_DB`. Finally, four remaining files were copied by hand so that `inst/` reflects the same structure as `my_app/`.

Next the `inst/makefile` must be modified to link in the AIMS monitor library. Check with the systems administrator to find the location of this library. In the case of a PVM application, the monitor library is called `pvmlib.a`. So if `pvmlib.a` is in directory `/usr/local/lib`, the command to compile the application should include `/usr/local/lib/pvmlib.a` at the end of the command. The MPI monitor library is called `mpilib.a` and is handled in a similar manner. For example, the `makefile` for PVM might contain the following lines:

```
#Specify location of AIMS' monitor
MON_LIB = $(AIMS_DIR)/lib
                ...
#Link application with monitor libraries

app:  $(F77) -o app_program $(APP_OBJS) $(MON_LIB)/pvmlib.a
```

Once the application is compiled and linked, it is ready to run. Execution should produce a tracefile with the default name of TRACE.OUT or the name indicated in the Settings Pane. With iPSC/860 or Paragon versions of AIMS, this file will need to be sorted as described in section 3.1. With PVM or MPI versions, a sorted tracefile (default name TRACE.SORT) will be created.

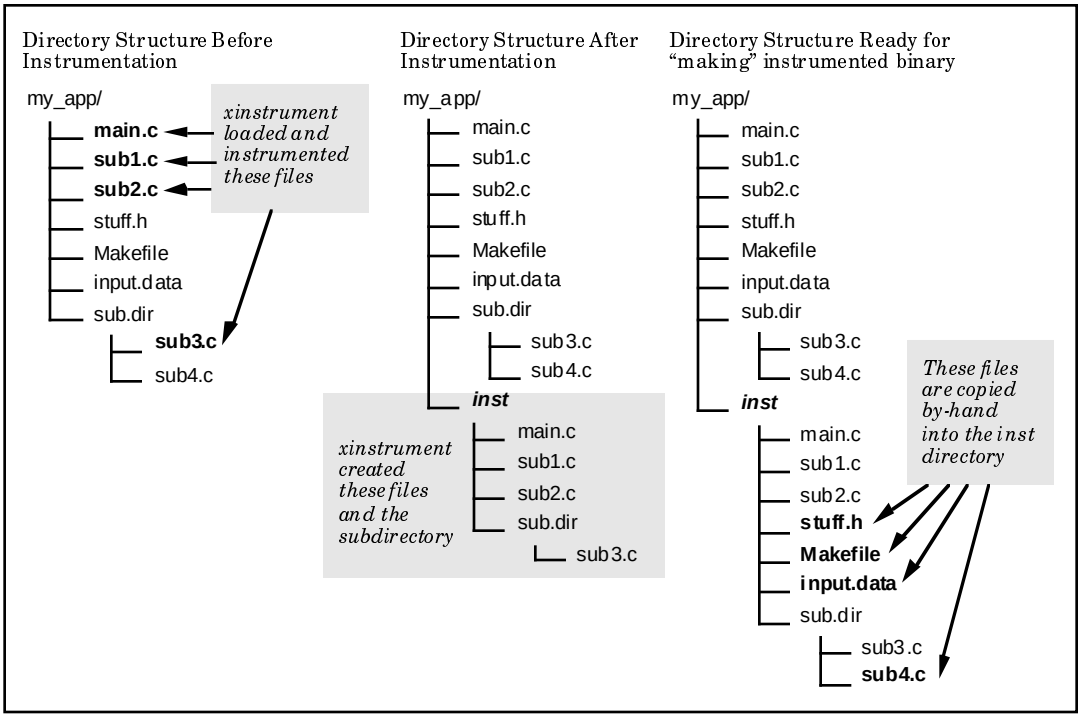


Figure 3-1: Illustration of the steps in organizing directories and preparing to use the makefile.

3.1 Sorting the Trace File

Sorting the trace file is required for the Intel iPSC/860 and Paragon versions of AIMS, but this is not needed for other versions of AIMS such as the PVM and MPI versions. The trace file must be sorted before invoking the trace analysis tools. While the records for each node are already sorted by time within the trace file, the records for different nodes are interleaved as all nodes use a shared tracefile, and trace records may therefore be out of order. AIMS provides a `tracesort` tool for sorting trace files. This tool may also be used with PVM and MPI versions of AIMS to sort a large tracefile on a system that has more room than the system on which the tracefile was created. Be sure to type in the redirection indicator (>) when invoking tracesort as follows:

```
tracesort  <tracefile> > <sorted_tracefile>
```

3.2 Using the "Profile"

`xinstrument` can also be used to selectively turn inserted instrumentation on or off in. To do this, load in the application database that already exists in the /inst directory using the '`-adb`' flag on the command line for `xinstrument`. After selectively turning on (or off) instrumented constructs, a "profile file" can be saved (with a user supplied file name) using the Advanced Options menu. Do not re-instrument the application. Edit the `AIMS.monrc` file and supply the full pathname of the saved file on the "`PROFILE:`" line. (A new "`PROFILE:`" line may need to be created.) The new `AIMS.monrc` file will be used the next time the instrumented application executes.

4. Visualizing Trace Files with VK

The View Kernel (VK) animates the trace file obtained by executing an instrumented parallel program. VK's displays present information such as: active subroutines, disk accesses, messages sent between nodes, and the route messages travel in a network of workstations. Some displays scroll as time passes, showing a segment of the program's history, while others animate each state in sequence. Several displays can be viewed at once. The trace file can be stepped through or visualized at high speed, stopping only when certain subroutines are invoked. A source code click-back capability allows easy examination of the source code corresponding to selected events on the display. There are also many ways to customize the displays to better reflect the design of the monitored program.

4.1 Invoking VK

VK should be invoked on a workstation that holds the uninstrumented code directory so that the source files are available for click-back. VK may be invoked in one of the following ways:

```
VK

VK  <sorted trace file>
```

The trace file specified on the command line should be sorted if necessary (See section 3.1). VK can view only one trace file at a time. VK may be invoked with the following options.:

`-help`	for help information
`-start <start time>`	to have VK start the display a specified execution time
`-stop <stop time>`	to have VK stop the display at a specified execution time

If the `-help` flag is present, VK prints a usage message and exits.

The values `<start time>` and `<stop time>` should be non-negative real numbers, which represent the time in milliseconds. If no `-start` flag is present, VK will begin viewing at the beginning of the trace file. If no `-stop` flag is present, VK will view to the end of the trace file (or 1,000,000 msec, whichever comes first).

VK has a color editor that allows the change of colors associated with various constructs. Sometimes VK runs out of space for allocating colors. This situation can occur if running several VKs are running, or if viewing a trace file with a very large number of subroutines and blocks. If a message appears indicating that VK cannot allocate a sufficient number of colors, VK should be restarted with the `-fixcolors` flag on the command line (to disable VK's color editor). If this message appears when VK is started, and other VKs are running; the execution of one of the other VKs should be halted and then restarted with the `-fixcolors` flag as well.

4.2 Using VK



Figure 4-1: VK's Menu.

After VK is invoked, a menu like the one in Figure 4-1 appears along with an OverVIEW window like that in Figure 4-2. If no file was used when invoking VK, select Load Trace File from the Files menu (Table 4-1) to bring up a dialog box to enable file selection. If necessary, select the subdirectory, and then select the file to be loaded. Click on Load to load the file, and then click Exit. The file may now be displayed or other menu options (Controls, Views, or Legends) shown in Tables 4-2, 4-3, and 4-4 may be selected. For example the Spokes view may be selected and opened simultaneously with the OverVIEW.

When the tracefile is ready for display, the VCR-like controls shown in Figure 4-1 can be used to control tracefile playback. Once a file has been loaded, and view has been opened, the ▶ (play) button causes VK to start reading and displaying the trace file. The ⏮ (rewind) button will reset VK to the beginning of the trace file. Clicking on ⏸ (pause) pauses VK. If VK was paused in the middle of the file, pressing ▶ will cause it to continue beyond that point, but clicking on ▶ at the end of the trace file has no effect. To step through the trace file one record at a time, click on the ⏭ (step forward) button.

Table 4-1: Files Menu Selections.

Files Menu	Description
Load Trace File	This brings up a file dialog box for loading a new trace file.
Load SysConfig File	This brings up a file dialog for loading a SysConfig file (see Section 4.3.2) to be used in conjunction with the loaded tracefile.
Exit	This exits VK.

Table 4-2: Controls Menu Selections.

Controls	Description
By Time	This allows the user to set pause and resume times for the animation'.
Enable Breakpoints	Turning this on will cause VK to stop animation whenever any node enters one of the selected constructs.

Table 4-3: Views Menu Selections.

Views	Description
OverVIEW	This is an animation with time along the X axis, and nodes along the Y axis.
Spokes	This view displays the processes in a circle, with message lines drawn between them as time progresses. Nodes are color coded.
SysConfig	This is a topological view of the network interconnections between workstations. Paths of messages can be lit, in conjunction with selections made in the OverVIEW.

Table 4-4: Legends Menu Selections.

Legends	Description
Construct Legend	This lists modules instrumented. Any module may be opened to show a tree of the constructs in that application.
Node States	Shows color key corresponding to the spokes view.

4.3 The VK Views

4.3.1 OverVIEW

Figure 4-2 contains an OverVIEW of a PVM program executing on a network of workstations. On the left hand side are the machine names and unique numbers marking each PVM task. The time axis runs horizontally, with the range given at the bottom of the view. By default, the range is 100 milliseconds. Moving the pointer into the OverVIEW window and pressing 'x' creates a dialog box in which the horizontal range can be set.

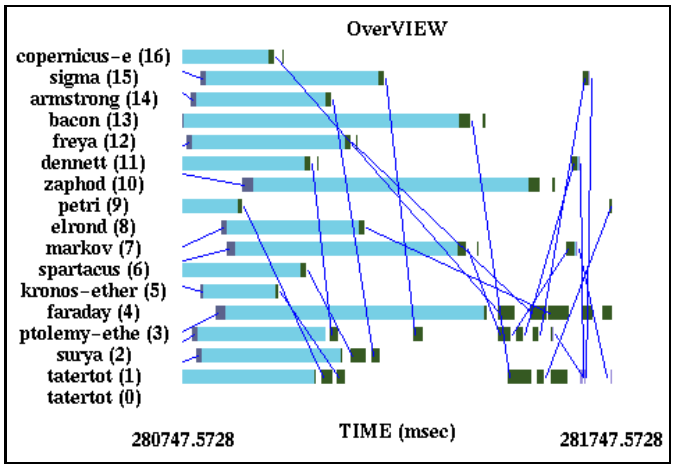


Figure 4-2: OverVIEW.

Rows of colored bars scroll left, with each row representing a PVM task. Each color represents a different instrumented subroutine. White space indicates when a task is blocked waiting to complete a send or receive. In PVM trace files, white space is also used to indicate message pack and unpack time. Bars made up of the XX pattern indicate time spent writing AIMS trace files to disk. Bars made of the rr pattern indicate a read, and bars of the ww pattern indicate a write. Bars composed of smiley faces represent `lseek` time, and bars composed of bugs represent Intel `io_wait` time. Table 4-5 shows these bitmaps. During asynchronous I/O in Intel traces, a line below the task bar indicates the duration of the access, illustrating when overlapped I/O and computation occur. Figure 4-3 is an example OverVIEW with instrumented I/O.

Table 4-5: I/O bitmaps in OverVIEW.

Bitmap	Meaning
	lseek time
	io_wait time
	read time
	write time

Thin lines drawn at angles between the horizontal bars represent messages being transmitted between tasks. The left end of the line is the sending side, and the right end the receiving side.[2] Message lines, like subroutine bars, can be clicked on for additional information. The user can interact with the bars and message lines to learn what code is executing at a given time with the keystroke and mouse combinations given in Table 4-6. Using different key combinations and mouse buttons will display the source code or construct tree for a routine.

[2] Lines may be colored if individual data structures were instrumented and monitored. When data structures are monitored, different colored messages represent communication of different data structures.

Table 4-6: Click-back keystroke combinations.

	Action		
Information desired	OverVIEW Object	Mouse button	Key
Routine name / cause for idle	subroutine bar	middle	
Construct tree of routine	subroutine bar	middle	control
Statistics about message	message line	left	
Construct tree of send task	message line	left	control
Construct tree of receive task	message line	right	control
Source of send task	message line	left	shift
Source of receive task	message line	right	shift

A Construct-Tree view shows an abstraction of the code's structure. An example construct tree appears in Figure 4-4. Construct views are usually brought up in relation to some particular construct in which the user is interested (e. g., a subroutine). This construct is then highlighted in the construct tree view. For example, `Multiply` is the highlighted construct in Figure 4-4.

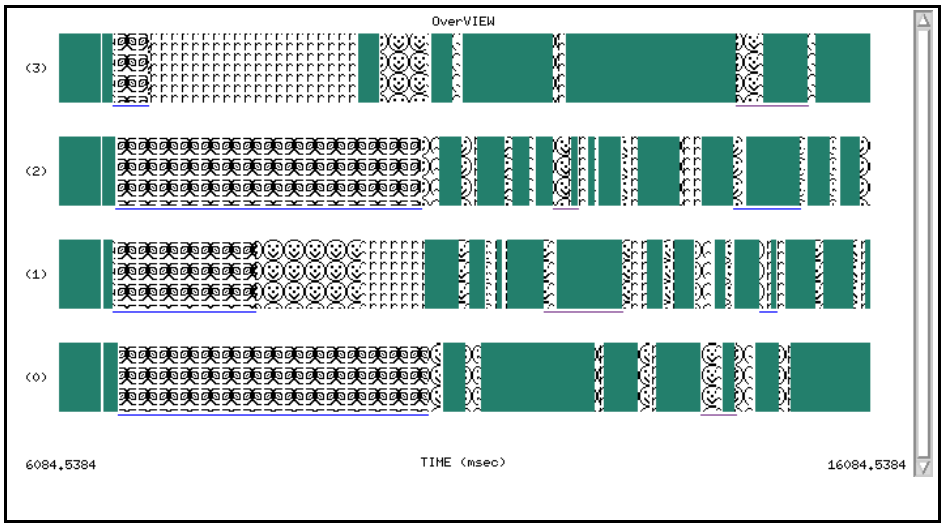


Figure 4-3: OverVIEW illustrating reads, seeks, and io_waits.

OverVIEW may be instructed to stop animation upon reaching a certain routine or message. To do this select a construct view that shows the construct you are interested in. Click the box of that construct so that it is highlighted. In Figure 4-5 the Subroutine [Pipe], Subroutine [Multiply] and a pvm_recv are all selectable constructs. Once the construct is highlighted, the menu Controls/Enable Break Points should be selected. When selected, the menu item should have a small box lit indicating this option is turned on. After resuming animation, OverVIEW will stop animating and display the construct tree when this event occurs. Note that OverVIEW will stop on every instance of this event. To turn off the breakpoint select the Controls/Enable Break Points menu again, and the on box will go away.

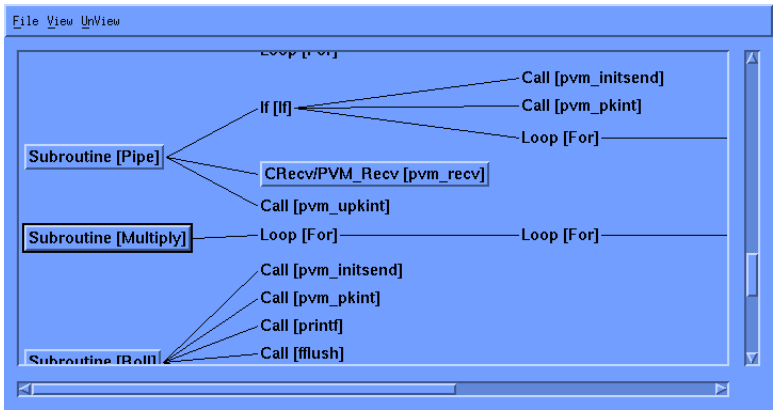


Figure 4-4: Construct Tree.

OverVIEW can also be instructed to stop animation at a particular time. Selecting the Controls/By Time menu option will bring up a control panel. Changing the resume time in this control panel will move OverVIEW forward or backward when animation continues. Setting a pause time will have OverVIEW stop animation at a particular time. The current time field is a display and is not editable. To make the control panel disappear, select the Controls/By Time menu option again.

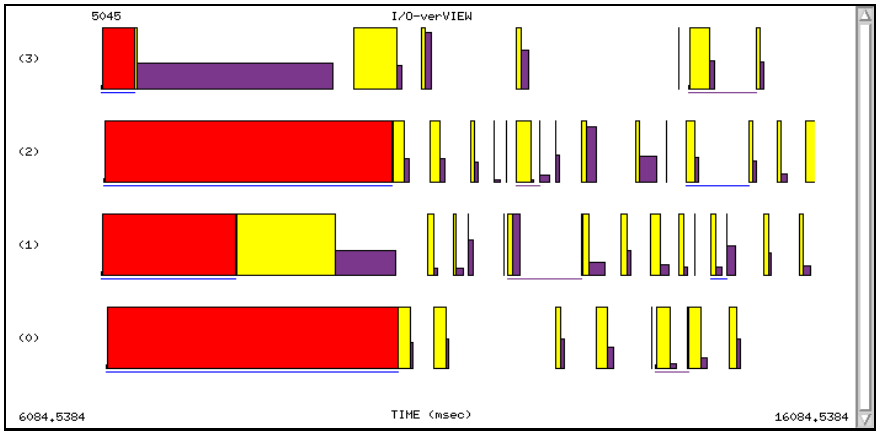


Figure 4-5: I/OverVIEW.

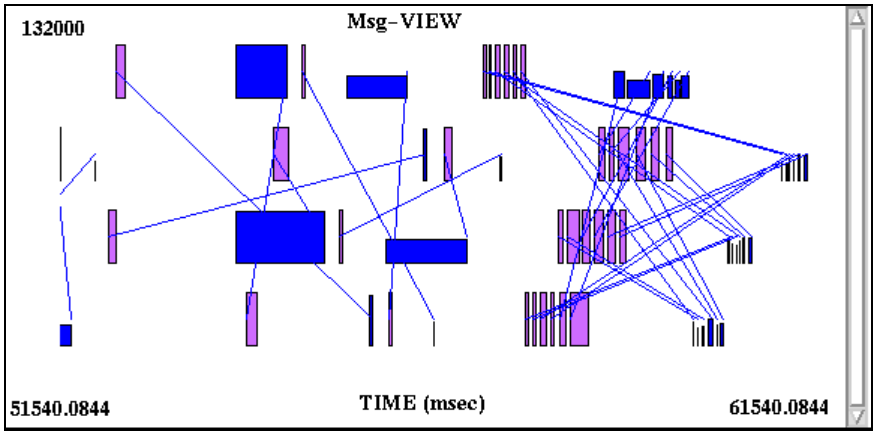


Figure 4-6: MsgVIEW.

The OverVIEW display can be toggled to show two additional views: the I/OverVIEW display and the MsgVIEW display. This is done by pressing 'e' (exchange views) when the pointer is in the OverVIEW window. A title at the top of the display indicates which view is being displayed. I/OverVIEW (shown in Figure 4-5) displays colored bars indicating read, write, seek, and io_wait time, should these events be instrumented. The height of a bar represents the number of bytes being accessed. The number in the upper left corner represents the largest disk access currently on the screen. Disk accesses of this size are shown as full height bars, while smaller disk accesses have proportionately shorter bars.

Pressing 'e' while in the I/OverVIEW will exchange it for the MsgVIEW (shown in Figure 4-6). In a sense, MsgVIEW is the inverse of the OverVIEW. Idle time is shown with two toned colored bars, one for send idling the other for receive idling and time spent computing is shown in white space. Message lines are also displayed. The height of a bar represents the size of the message. Like with the I/OverVIEW, the number in the upper left corner represents the largest message currently on the screen, and all smaller bars are normalized to that size. Messages of this size are shown as full height bars, while smaller messages have proportionately shorter bars

4.3.2 SysConfig View

The SysConfig view is designed especially for understanding the network configuration of workstations. From a `sysconfig` file a topological layout can be made of a Parallel Virtual Machine. An annotated SysConfig view appears in Figure 4-7.

When used in conjunction with the OverVIEW, the SysConfig view can display the path a message takes through the network and highlight resources used. When a message is clicked on in OverVIEW, the corresponding path on the SysConfig view is lit between the machines. This is shown in the Figure 4-7 between the machines Tatertot and Ptolemy.

Horizontal lines represent subnets (their IP addresses given on the left). Vertical lines represent connections from machines to networks. The row of smaller boxes on the far left are routers in the network.

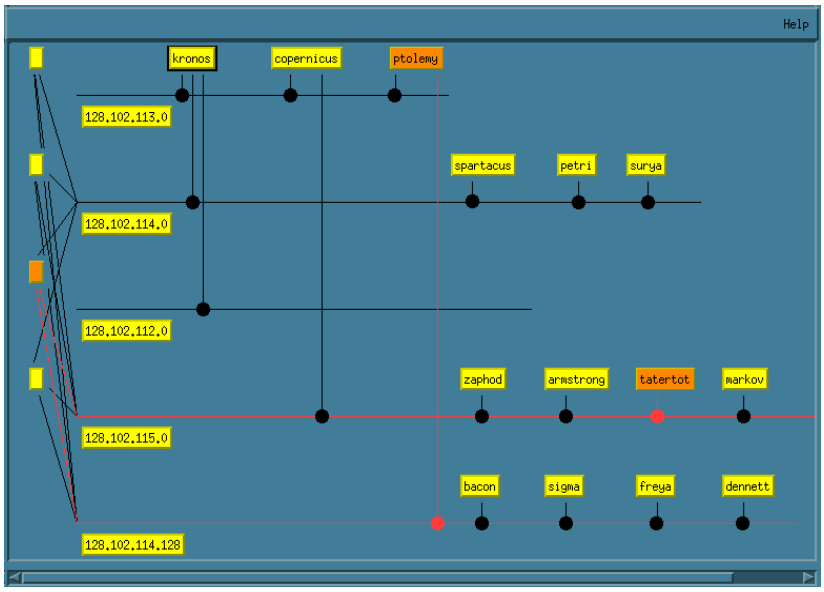


Figure 4-7: SysConfig View.

4.3.3 Spokes View

To open the Spokes window select the Spokes option, under the View menu. The Spokes view animates messages passed between tasks while showing the state of each task. Since the tasks are arranged in a circle some message passing patterns are easier to identify than when shown in OverVIEW. The spokes view is shown in Figure 4-8. After selecting the Spokes option, select the Nodes States option from the Legends menu. This will show the Spokes View color key. this is also shown in Table 4-7.

Table 4-7: Spokes View Color Key.

Color	State
green	Busy
blue	Idle receive
yellow	Idle send
orange	Idle global
brown	Idle other
hatched (xx)	Flushing
black	Not tracing

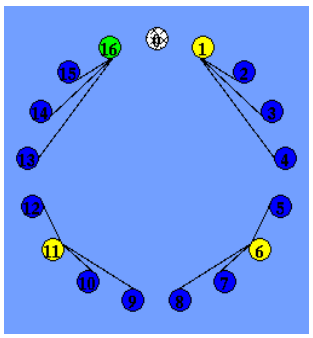


Figure 4-8: Spokes View.

4.4 Speeding up and slowing down VK

If VK is running too slowly, a non-zero jump factor can be specified for the OverVIEW. To enter the jump factor, press "v" while pointing inside the OverVIEW window. The value of this factor, a number between 0 and 1, inclusive, specifies a minimum fraction of the window that VK must scroll. A value of 0 causes VK to scroll the minimum amount necessary, but a value of 0.5 will ensure that VK always scrolls by half a screen. With a larger jump factor VK will scroll fewer times, and will display things faster. In general, small jump factors of about 0.1 speed up VK greatly without disturbing the display too much. Using smaller windows and only having one animation active at a time will also speed up animation.

If VK is displaying the trace records too quickly, setting pause times via the By Time menu, or setting break points on certain constructs will slow it down. Decreasing the jump factor will also slow down the animation when viewing scrolling displays. If pictures are moving out of the window too quickly, the scale of the scrolling views can be changed to view a larger segment of the program .

5. tally

`tally` generates a list of resource-utilization statistics on node-by-node and routine-by-routine bases. The routine statistics give information typically provided by profilers with respect to the amount of time spent in various functions. In addition, it provides easy access to the percentage of execution time spent communicating and the significance of the communication time in comparison with the total program execution time. The statistics can help to quickly determine the sections of code that need to be tuned. The output of `tally` can be used as input to statistical drawing packages such as Excel and WingZ.

5.1 Invoking tally

`tally` is invoked with a sorted trace file or a `-help` flag as follows:

```
tally   <sorted trace file>
        -help
```

If the `-help` flag is present, `tally` prints a usage message and exits. If no trace file is specified, `tally` uses `inst/TRACE.SORT`.

5.2 Output from tally

`tally` produces several tables of statistics. The first table presents data for each function executing in the program. It is sorted in descending order with respect to function execution times and contains the following information:

1. Routine: The routine index and the name of the subroutine.
2. Busy time: Time the function was performing useful work. This is the amount of time not spent in communication.
3. Global Blocking: Time a routine spent in a global blocking operation.
4. Send Blocking: Time a routine spent in a send operation.
5. Receive Blocking: Time a routine spent in a receive operation.
6. Life time: Time taken to execute instructions in each function (excluding the functions called from this function).
7. Percentage Communication: Percentage of total execution time the routine spent in communication.
8. Communication Index: Time spent in the function with respect to the total time spent in the program, as well as the percentage of time spent in communication in this function. (The lower this value, the lower the impact on the total program execution time of reducing this function's communication characteristics).

The second table consists of columns that show the aggregate communication characteristics of nodes executing the program and contains the following information:

1. Node number.
2. Busy time: Time spent not performing communication.
3. Global Blocking: Time spent in a global blocking operation.
4. Send Blocking: Time spent in a send operation.
5. Recv Blocking: Time spent in a receive operation.
6. Life time: Time spent executing the program.
7. Percentage communication: Percentage of execution time spent in communication.
8. Link Contention: Percentage of total communication time a node spent in contention.

After this second table is produced with data for each node, it is produced for each instrumented routine in the application. All the tables described above are directed to standard output and stored in the file `tally.summary`. In addition, two statistical parameters are computed and directed to the standard output and stored in a file called `ncpu.summary`. They are: NCPU and routine concurrency.

The NCPU for a given subroutine and a given k is the amount of CPU time used by that subroutine when k processors are busy, divided by k. For example, the NCPU data for a particular application is plotted in the lower left-hand-corner of Figure 5-1. It is a highly parallel program with all (16) of the processors concurrently busy for 325 msecs. During most of that time, subroutine `eigv` is executing. If a subroutine spent much time executing when only a few nodes were busy, this may indicate that the routine inhibits parallelization and may be a bottleneck.

The Routine Concurrency data for the same trace is plotted in the lower right-hand-corner of Figure 5-1. It indicates the amount of time spent by each subroutine when k copies were executing simultaneously. This view indicates the degree to which each routine was parallelized. If a routine never has more than a few copies running simultaneously, it may indicate that the routine is inherently sequential. Note that this property differs from that of inhibiting parallelism for all subroutines, as described above with the NCPU chart. As expected, `eigv` was the most parallelized routine: it executes concurrently on all the processors for 150 msec.

A great deal of information is output by `tally`, so using a statistical drawing package to look at the data can be beneficial. An example of this is shown in Figure 5-1. To facilitate that process, each row of tally's tables is a list of numbers or strings separated by tabs. Tables are preceded by a title and separated by a blank line.



```
sysconfig <trace-file>
```

To use a PVM hostfile type:

```
sysconfig -hostfile <host-file>
```

`sysconfig` will then examine each machine used in the tracefile, or listed in the hostfile. `sysconfig` will find what network connections each machine has, and use `traceroute` to see what route a packet takes between every machine pair. These routes are assumed to be the path that PVM messages take between tasks running on the machines. Output will be saved in a file named after the input file with `.SC` appended. `sysconfig` data can be viewed in VK as explained in Section 4.3.2.

7. Customizing AIMS

AIMS tools have many parameters that allow changes to things like fonts, initial window sizes, default locations of the trace file and application database, and specific features of VK's views. The parameters have names like `xinstrument.height`, and `vk.overview.font`. The sections below explain how to change the default values of the parameters and how to change the values of certain parameters at run-time. Appendix C contains a list of AIMS's parameters.

7.1 Setting Defaults for Parameters

Defaults for each of AIMS' parameters are built into the system. However, there are a number of ways to override these defaults. This may be needed if, for example, the default fonts are not available on their system. The defaults can be set on the command-line or in one of several default files.

7.1.1 Specifying Defaults on the Command Line

Several switches are provided to set values for parameters on the command-line. These are `-bg` and `-fg` for background and foreground, `-bd` and `-bw` for border color and border width, and `-fn` for font. For example, "`VK -bg black -fg chartreuse`" would give a glow-in-the-dark look. In addition, the `-xrm` switch can be used to specify the value of any parameter, by following the switch with the full name of the parameter, a colon, and the parameter's value. For example, "`VK -xrm vk.overview.messageColor:magenta`" would make it very easy to spot the messages that OverVIEW draws.

7.1.2 Specifying Defaults in Files

AIMS looks in several files, including `.Xdefaults`, for defaults. To specify default values in one of these files, add to the file lines of the form "`<default name>:<default value>`". The * notation may be used to specify several defaults with one line. For example, the line "`vk.*.borderWidth:5`" will set the border width of all of VK's views to 5. (The * notation is discussed more fully in many X manuals. See, for example, Section 11.4 in Volume One, the Xlib Programming Manual, by Adrian Nye.) Lines in a default file that begin with an exclamation point are treated as comments. A small default file is shown in Figure 7-1.

```
! Set default trace file for all X-based tools
*.traceFile: inst/tsort

! Set fonts for VK
vk.*.font: *lucida-medium-r-normal-sans-12-*
vk.help.font: *fixed*medium*-r-*-10-*

! Set jump factor to scroll faster
vk.*.jumpFactor: 0.15

! Position the OverVIEW, and make it long
vk.overview.x: 10
vk.overview.y: 200
vk.overview.width: 800
```

Figure 7-1: Example of X-defaults.

7.1.3 How AIMS Finds Defaults

Like many X-based applications, AIMS looks for defaults in the following four sources, in order, until it finds a match:

- Command line
- File named in the `XENVIRONMENT` variable (or `.Xdefaults` file, if `XENVIRONMENT` is not set)
- Database created by the `xrdb` program (or `.Xdefaults` file, if `xrdb` has not been run)
- The file `Aims` in the subdirectory `/app-defaults`

A command-line value takes precedence over a value in a `.Xdefaults` file, which in turn takes precedence over one in the system defaults file. If no default value is present in any of the four sources, AIMS uses its built-in defaults.

7.2 Changing VK's Parameters Dynamically

VK allows the user to change the value of many parameters while the program is running. These dynamic parameters are changed by pressing certain keys in the appropriate VK window. As a result of the key press, a window appears. The current value is displayed and can be erased by backspacing. Enter the new value and hit `<Return>` or click on the Okay button. If a value is entered that is not legal, the terminal beeps. Clicking on ⏮ sets the value back to the previous value, and clicking Cancel resets the value and removes the editing window. The window includes a Help button to explain the selections.

For example, the user may change VK's stop and resume times. To do this, bring up the editing window by pressing a 't' in the OverVIEW window and then put the cursor over the value to change. Remember to hit `<Return>` in the window when done, or the change will not take affect.

The following are the keys corresponding to the different parameters:

Parameter	Key
`vk.<view>.minTime`	t
`vk.<view>.maxTime`	T
`vk.<view>.jumpFactor`	j or J
`vk.<view>.minValue`	v
`vk.<view>.maxValue`	V
`vk.<view>.minScalingOp`	s
`vk.<view>.maxScalingOp`	S
`vk.overview.showMarks`	r or R
`vk.overview.showMessages`	m or M
`vk.overview.barWidthFactor`	b or B
`vk.overview.drawDividers`	d or D
`vk.commLoad.volumeOrCount`	c or C
`vk.commLoad.scaleToFactor`	o or O
`vk.commLoad.scaleWhenFactor`	w or W
`vk.commLoad.scaleAfterValue`	a or A
`vk.inboxSizes.numSizes`	n or N
`vk.inboxSizes.maxSize`	m or M
(node ordering in overview)	o or O

8. Appendix A. Bibliography

The following is a list of papers that may be useful for the AIMS users.

[1] J. C. Yan, S. R. Sarukkai, and P. Mehra. "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit". Software Practice & Experience. April 1995. Vol 25, No. 4, pages 429-461

[2] S. R. Sarukkai and J. C. Yan. "Event-Based Study of the Effect of Execution Environments on Parallel Program Performance". Proceedings of the Forurth International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOT '96), San Jose CA, January 1996.

[3] S. R. Sarukkai, J. C. Yan and M. Schmidt. "Automated Instrumentation and Monitoring of Data Movement in Parallel Programs". Proceedings of the 9th International Parallel Processing Symposium, Santa Barbara, CA. April 25-28, 1995. pages 621-630

[4] P. Mehra, B. VanVoorst, and J. C. Yan. "Automated Instrumentation, Monitoring and Visualization of PVM Programs". Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing. San Francisco, CA. February 15-17, 1995. pages 832-837

[5] S. R. Sarukkai, J. C. Yan and J. Gotwals. "Normalized Performance Indices for Message Passing Parallel Programs," Proceedings of the International Conference on Supercomputing ICS-94, Manchester, England, July 11-15, 1994. pages 323-332.

[6] J. C. Yan, M. Schmidt and S. R. Sarukkai. "Monitoring the Performance of Multidisciplinary Applications on the iPSC/860". Proceedings of the 1994 Scalable High Performance Computing Conference, Knoxville, Tennessee, May 23 - 25, 1994. pages 277-284.

[7] J. C. Yan. "Performance Tuning with AIMS — An Automated Instrumentation and Monitoring System for Multicomputers". Proceedings of the 27th Hawaii International Conference on System Sciences, Wailea, Hawaii, January, 4 - 7, 1994. Vol. II. pages 625-633.

[8] J. C. Yan and S. Listgarten. "Intrusion Compensation for Performance Evaluation of Parallel Programs on a Multicomputer". Proceedings of the ISCA 6th International Conference on Parallel and Distributed Computing Systems, Louisville, KY, October 14-16, 1993, pages 427-431.

[9] J. C. Yan, P. J. Hontalas, and C. E. Fineman. "Instrumentation, Performance Visualization and Debugging Tools for Multiprocessors". Proceedings of Technology 2001, San Jose CA, December 4-6, 1991. Vol. II., pages 377-385.

[10] J. C. Yan, P. Hontalas, S. Listgarten, C. Fineman, M. Schmidt and C. Schulbach. "The Automated Instrumentation and Monitoring System (AIMS) Reference Manual". NASA Technical Memorandum. 108795, December 1993.

9. Appendix B. Installation Guide

Installing AIMS requires three steps: creating the source tree, compiling the system, and installing the executables. Various portions of AIMS may be installed on different machines. The source code is distributed in UNIX's `tar` format.

9.1 Extracting the Source Code from the Archive

The first step once the archive (`tar`) file has been obtained, is to create a directory for AIMS. The tar file should be installed into that directory (as shown in Example B-1).

```
cd
mkdir aims_source
mv aims.tar aims_source
cd aims_source
tar xof aims.tar
rm aims.tar
```

Example B-1

After extracting the file, the source directory should have a `Makefile` and eight subdirectories: `common/`, `example/`, `instrumentors/`, `misc/`, `monitor/`, `notes/`, `sage/`, and `tools/`. Once the source tree has been created, the source can be compiled. To do this, the `Makefile` in the top-level directory should first be edited. The following may need to be changed: compilation directives, installation directories, location of X libraries, default location for binary files, and other system-specific definitions. These are discussed in turn below.

9.2 Setting Up AIMS for the Installation Site

The `Makefile` contains various environment settings to help the user customize AIMS for a specific site.

- `ARCH` — Defines the multiprocessor platform on which AIMS will be used. It is important to build AIMS for the correct platform. Alternatives for ARCH are shown in Table B-1

Table B-1: Alternative platforms for AIMS.

ARCH (Architecture)	Platform to be Used
nx860	NX on the Intel iPSC/860
pvm	PVM on Sun and SGI workstations
mpisp2	MPI on IBM SP2
mpi	MPI on Sun and SGI workstations

- "Compilation directives" — indicates which parts of AIMS are to be created. The settings shown in Example B-2 will generate binaries for `VK`, `tally`, and `tracesort` but NOT generate for the `monitor` and the instrumentors. ALL components should be generated the first time AIMS is installed.
- "Installation directories" — indicate where the appropriate files will be located (as shown in Example B-3).
- The location of the X include files and libraries should then be specified.

```
MONITOR=0
INSTRUMENTORS=0
VK=1
TALLY=1
TRACESORT=1
```

Example B-2

```
INSTALL_DIR = $(HOME)/aims_source/bin

MONITOR_INSTALL_DIR = $(HOME)/aims_source/lib

MAN_INSTALL_DIR = $(HOME)/aims_source/man
```

Example B-3

Finally, there are a few other system-specific definitions required for compiling the system. They are grouped into three sections, depending on whether the machine is running IRIX, Sun OS, or Ultrix. "Uncomment" the lines corresponding to the system of choice by removing the "#" signs, ensuring the other lines are commented out.

9.3 Compiling and Installing AIMS

The system takes about 20 minutes to compile (by typing "`make`"). When compilation has completed, typing "`make install`" should be typed to install AIMS in the directories specified in the `Makefile`. If the full system was created, the executables listed in Table B-2 will be moved to the binary installation directory: The monitor installation directory will also contain files. The Intel iPSC/860 version produces `nodelib.a` and `hostlib.a`, the PVM version produces `pvmlib.a`, and the MPI versions produce `mpilib.a`.

Table B-2: AIMS executables.

`atopg`	translates AIMS tracefiles to ParaGraph format
`cfp`	FORTRAN 77 parser for Sigma
`dumpdep`	Sigma utility
`stat2tally`	generates tally output from a `.stat` file
`stattidmap`	`tidmap` for `.stat` files
`tally`	tabulates statistics from the tracefile
`tidmap`	replaces pvm processes-ID's in the trace file (which are large integers) by integers ranging from 0 to n-1
`tracesort`	sorts a tracefile by time
`unparse`	Sigma utility
`vcc`	Sigma C parser
`VK`	graphically displays the trace file
`vpc`	Sigma utility
`xinstrument`	instrumentor front-end

9.4 Managing Windows With AIMS

AIMS provides application specific buttons and menu options for destroying and closing windows. Some users depend upon their Window Manager's title bar menu for quitting or closing windows. However, in the current version, these Window Manager specific options may cause problems within AIMS. Future versions will handle these menu options internally. At this time, users should only use the AIMS specific menu options. Those who use the Window Manager menus, should add to their Window Manager's Resource Description File (e.g., `.mwmrc`, `.twmrc`, etc.) a menu which contains all of the same options of their DefaultWindowMenu except the "Close" or "Quit" options (which may map to the `f.kill` and `f.quit_app` event functions). This menu should be named "AimsWindowMenu". In addition, windowMenu options for each AIMS window should be added to the user's system default file. If the user is using one of the AIMS supplied defaults file, these options are already present.

10. Appendix C. AIMS' Parameters

This is a list of AIMS's parameters.

*circleBox*windowMenu
*commLoad*windowMenu
*constructTreeShell*windowMenu
*controlByTime*windowMenu
*grid*windowMenu
*help*windowMenu
*inboxSizes*windowMenu
*nodeState*windowMenu
*vk*windowMenu
vk.applicationDatabase
vk.blocked.color
vk.boxes.breakpointsEnabled
vk.boxes.maxInboxCount
vk.boxes.spectrumSize
vk.circle.background
vk.circle.borderColor
vk.circle.borderWidth
vk.circle.font
vk.circle.foreground
vk.circle.height
vk.circle.width
vk.circle.x
vk.circle.y
vk.clickback.big.font
vk.clickback.medium.font
vk.clickback.small.font
vk.commLoad.background
vk.commLoad.borderColor
vk.commLoad.borderWidth
vk.commLoad.countColor
vk.commLoad.font
vk.commLoad.foreground
vk.commLoad.height
vk.commLoad.jumpFactor
vk.commLoad.maxCount
vk.commLoad.maxScalingOp
vk.commLoad.maxTime
vk.commLoad.maxVolume
vk.commLoad.minCount
vk.commLoad.minScalingOp
vk.commLoad.minTime
vk.commLoad.minVolume
vk.commLoad.scaleAfterValue
vk.commLoad.scaleToFactor
vk.commLoad.scaleWhenFactor
vk.commLoad.volumeColor
vk.commLoad.volumeOrCount
vk.commLoad.width
vk.commLoad.x
vk.commLoad.y
vk.eventsLoop
vk.fixColors
vk.grid.background
vk.grid.borderColor
vk.grid.borderWidth
vk.grid.font
vk.grid.foreground
vk.grid.height
vk.grid.width
vk.grid.x
vk.grid.y
vk.help.normfont
vk.help.sharpfont
vk.inboxSizes.background
vk.inboxSizes.borderColor
vk.inboxSizes.borderWidth
vk.inboxSizes.font
vk.inboxSizes.foreground
vk.inboxSizes.height
vk.inboxSizes.maxSize
vk.inboxSizes.numSizes
vk.inboxSizes.width
vk.inboxSizes.x
vk.inboxSizes.y
vk.menu.background
vk.menu.borderColor
vk.menu.borderWidth
vk.menu.font
vk.menu.foreground
vk.menu.title.font
vk.menu.title.foreground
vk.nodeState.blockedGlobal.color
vk.nodeState.blockedReceiving.color
vk.nodeState.blockedSendingColor
vk.nodeState.font
vk.nodeState.notTracing.color
vk.nodeState.running.color
vk.overview.background
vk.overview.barWidthFactor
vk.overview.borderColor
vk.overview.borderWidth
vk.overview.font
vk.overview.foreground
vk.overview.height
vk.overview.highlightColor
vk.overview.jumpFactor
vk.overview.maxTime
vk.overview.messageColor
vk.overview.minTime
vk.overview.showDividers
vk.overview.showMarks
vk.overview.showMessages
vk.overview.width
vk.overview.x
vk.overview.y
vk.startTime
vk.stopTime
vk.timeStep
vk.traceFile
vk.traceRecord.background
vk.traceRecord.borderColor
vk.traceRecord.borderWidth
vk.traceRecord.font
vk.traceRecord.foreground
vk.traceRecord.height
vk.traceRecord.width
vk.traceRecord.x
vk.traceRecord.y
vk.utilizationLegend.font