Program Monitoring Tools For Parallel Processing With Large-Grain Data Flow Techniques

Robert G. Babb II
David C. DiNucci
Lise Storc

Oregon Graduate Institute
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 87-017

PROGRAM MONITORING TOOLS FOR PARALLEL PROCESSING WITH LARGE-GRAIN DATA FLOW TECHNIQUES

Final Report for Los Alamos National Laboratory

Computer Research and Applications Division

under

Contract 9-Z34-P3915-1 Modification No. 2

Robert. G. Babb II
David C. DiNucci
Lise Storc

Department of Computer Science and Engineering
Oregon Graduate Center
Beaverton, Oregon

8 October 1986

CONTENTS

1. Initial Requirements Definition

1.1. Software

The purpose of this research project was to define and implement a graphics monitor to aid in the debugging and analysis of Large-Grain Data Flow (LGDF) programs. Previously, a Los Alamos gamma ray transport benchmark code (GAMTEB) was parallelized for the Denelcor HEP using the prototype LGDF Toolset[1]. Our goal was to design and prototype a parallel program animation system that would give LGDF programmers "intuition" about how an LGDF computation, such as the LGDF version of GAMTEB, was progressing. As a secondary goal, we wanted the tool to aid in controlled very high level debugging of large parallel application codes.

Our approach was to enhance the LGDF macros so that they would optionally generate sufficient trace data to model the action of the program running on a multi-processing system. This file of trace data would be used by the monitor program (running on an IBM PC) to display parallel process initiations, and data flow events in a graphic form.

Although the monitor was to be capable of monitoring GAMTEB program, the monitor would be constructed to allow other LGDF programs to be modeled by merely re-macro expanding with the program animation option turned on.

The monitoring system was to be as interactive as possible. This suggested that the monitor could run in real-time—i.e., while the program being monitored was running on the host machine.

The LGDF program would be shown on the monitor as a data flow graph consisting of nodes (processes) and edges (datapaths). Processes would change color with execution state (green=running, red=sleeping, purple=terminated) and datapaths would change color with state of empty/full flag (red=empty, green=full).

A "speedometer" would show the speed of the display relative to the speed the program would have been running without display. Provisions would be made for the user to set the speed at some fixed value so that relative speed of display would reflect that of an actual execution.

Provisions for hierarchical expansion of the network display were to be considered; i.e. perhaps the user could, using a mouse, dynamically expand a bubble representing a subnetwork into its components and later collapse the subnetwork components back into a single bubble representing the subnetwork.

---

[1]R. G. Babb II and L. Storc, "Parallel Processing on the Denelcor HEP with Large-Grain Data Flow Techniques", Final Report for Los Alamos National Laboratory, Computer Research and Applications Division, 30 April 1985. Also available as Oregon Graduate Center Technical Report CS/E 85-010, May 1985.

Graphics would be performed with an established standard, preferably GKS to facilitate future development efforts and portability.

1.2. Hardware

An IBM PC or AT compatible system was to be used to run the monitor. The parallel host system was not to be restricted to any one parallel processor system, but our initial implementation could be performed using the LGDF Toolset in simulated parallel mode on Oregon Graduate Center's VAX 11/780.

2. Acquisition of Tools

2.1. Software

The compiler used was Microsoft Fortran 3.3. Code was linked with the IBM Linker 2.3, since the `/X` parameter to increase segment size was needed. A library package called No-limit Fortran was acquired to facilitate communication between the host system and monitor (especially in the event that the monitor would run in real-time), Graphics were to be developed following the Graphical Kernel System (GKS) standard for graphics. For this we purchased the GSS-Toolkit Kernel System #1125 from Graphic Software Systems, which provides the capabilities of Level 2b of the ANSI GKS Specification. GSS-Device Drivers, providing a standard VDI interface, were also used. Unfortunately, initially, there was a bug in the GSS software that precluded any use of a mouse. After several months, a workaround CGI driver was delivered by GSS which restored the necessary aspects of mouse function.

2.2. Hardware

We used an IBM PC-XT with 512K memory and 10Mbyte hard disk running DOS 3.1. To support the GKS kernel, we added an Enhanced Graphics Adaptor (EGA) card, enhanced color monitor and Microsoft Mouse.

3. Initial Specification Ideas

To avoid problems related to retrofitting an existing project with newly conceived features, we started the project with a brainstorming phase. We could then insure a modular and well thought out design, as well as the careful selection of those features which were central to the performance of the project to implement and test first.

3.1. Display Form

3.1.1. LGDF network

An LGDF network consists of a directed graph where each node is a circle and each arc is a line. Each node has a unique name of the form pnn and each data arc has a unique name of the form dnn, where nn is a two-digit integer. Arrows on arcs may be of three types: *clearable* (a solid triangle), *non-clearable*

(an open triangle) and *side-effect* (an open V shape). Arcs can either end at a node's circumference or proceed through the node with a dashed or solid line, at which point it may or may not continue on to another node. If the arc does continue to another node, the continuation arc has the same name as the original arc, but with a one letter suffix. Arcs can contain branches, in which case each branch has the same name as the original arc. An arc cannot pass through a node unless the node reads or writes the arc. (Therefore, arc paths must not cross over or under nodes just to get from here to there.) An arc may cross over another arc.

3.1.2. LGDF subnetwork

A node may represent a single program or a subnetwork. In the latter case, all arcs (and only those arcs) entering the node must be represented in the subnetwork. A subnetwork has the same form as a network, as described above. This nesting can occur to any level. When a subnet is expanded (i.e. its component nodes and arcs are displayed) within the context of its supernet, it is enclosed within a rectangular border (perhaps with rounded corners).

3.2. Display Actions

The display of the network on the graphics screen can be affected by Trace Actions read from the trace stream or by User Actions entered from the keyboard or mouse.

3.2.1. Trace Actions

Trace actions are the affect of reading a trace record from a trace file or from a communication port. There is not necessarily a one-to-one correspondence between trace records and trace actions.

Trace records are of the form:

```
#XpNNQsNNdNNQeNNNN.NNNNNNwNNNN.NNNNNN
```

where

```
X = s - set
        c - clear
        t - terminate
        w - wake up
        n - nap

N = digit (after p = process number
                     s = state number
                     d = data arc number
                     e = elapsed cpu time, in seconc
                     w = wall clock time, in seconds
Q = qualifier (space or lcase letter)
```

Trace actions are as follows:

```
If X = 'w' or 'n' or 't'
    Case X of
    'w':  COLOR = green
          SUBNET = (pNN is within one or more (nested)
                    subnets)

    'n':  COLOR = red
          SUBNET = (pNN is the only green bubble within one
                    or more (nested) subnets)
    't':  COLOR = purple
          SUBNET = (pNN is the only non-purple bubble within
                    one or more (nested) subnets)
    endcase
    If SUBNET
       Color of subnet circle turns COLOR
       Color of subnet rectangle turns light COLOR
    end if
    If Q is blank
       Color of circle associated with pNN turns COLOR
    else
       Color of slice Q of circle associated with pNN turns COLC
    end if
    Display state sNN for node pNNQ
else if X = 's'
    Color of arc dNNQ turns green
else (if X = 'c')
    Color of arc dNNQ turns red
end if
```

A trace action will not necessarily update the display when a trace record is read, because of either the display speed or the display format. Below is a list of cases to be checked after reading a trace record describing whether a screen update is to be performed and when. In all cases, whether or not a screen update is performed, the color and state of all entities must be updated internally with each trace record.

In the following, "Time reqd for screen update" is constant and approximate, and more information on "Pause" and "Speedometer" can be found below in "Performance measures and metrics" under "Relative display speed". (Note: to best facilitate this, it might be best to read a trace record, then check for any user actions, then perform the trace action and then read the next trace record.)

```
If no affected entity is currently displayed
    (No screen update)
    Increment Records_Skipped
else if (speed == 0)
    Perform screen update.
    Speedometer =   (w_time - st_w_time) * 100
          / (r_time - st_r_time)
else
    Pause =   (w_time - st_w_time) * 100 / speed
                    - (r_time - st_r_time)
    If Pause < time reqd for screen update
      (No screen update)
      Increment Records_Skipped
    else
      If Pause - 1 second > time reqd for screen update
         Wait for Pause - 1 seconds
      end if
      Perform screen update
    end if
end if
```

3.2.2. User Actions

User actions are from the keyboard or mouse, and affect the format of the display. Whenever the display is changed in this way, its colors are updated to reflect their state had the display been in this format since the beginning of the trace.

3.2.2.1. Expand a subnet

Cause:

(1) Pick of "expand" menu item with the mouse
(2) Pick of circle which represents a subnet

Effect:

Trace halts. Display is redrawn with the picked circle enlarged to a rectangle containing the subnetwork (one level). All nodes previously on screen will remain on screen. All circles will have equal radii. (If necessary, circle radii will be smaller after redraw to accommodate new subnetwork).

3.2.2.2. Collapse a subnet

Cause:

(1) Pick of "collapse" menu item
(2) Pick of empty spot within an expanded subnet rectangle

Effect:

Trace halts. Display is redrawn with the picked subnetwork represented as

circle. All nodes (outside the rectangle boundaries) previously on screen will remain on screen. All circles will have equal radii.

3.2.2.3. Restrict the display (ZOOMIN)

Cause:

(1) Pick of "zoomin" menu item

(2) Definition of rectangle to restrict display to

Effect:

Trace halts. Display is redrawn (and possibly enlarged) with only those nodes within the defined rectangle represented.

Notes:

Rectangle borders must be along grid lines. It may not be possible to split nodes or subnetworks.

3.2.2.4. Unrestrict the display (ZOOMOUT)

Cause:

(1) Pick of "zoomout" menu item

Effect:

Trace halts. Display is redrawn with all nodes present at time of corresponding zoomin.

Notes:

Should this be implemented at only one level, or as a stack?

3.2.2.5. Halt trace temporarily (STOP)

Cause:

(1) Pick of "stop/go" menu item while trace is active

Effect:

Trace halts (i.e. reading of trace records halts and therefore all trace actions halt).

3.2.2.6. Start or Continue trace (GO)

Cause:

(1) Pick of "stop/go" menu item while trace is stopped

Effect:

Trace starts where it was when last halt occurred. Start time and

3.2.2.7. Perform single visible trace step (NEXT)

Cause:

(1) Pick of "Single step" menu item

Effect:

Trace halts. Trace records are then read and performed until one of them affects an arc or node that is (at least partially) displayed on the screen.

After that trace action is performed, the trace halts again and the "number of trace records skipped" is updated on the screen.

3.2.2.8. Control trace speed

Effect:

Trace speed is set to desired figure.

3.2.2.9. Abort trace (KILL)

Effect:

Trace program terminates.

3.2.2.10. Set/Clear Breakpoint on process or datapath

Cause:

(1) Pick of "Set/clear Breakpoint" menu item
(2) Pick of a process or datapath

Effect:

Trace halts. If breakpoint already present on datapath/process, it is cleared, else one is set. A breakpoint has the affect of halting the monitor whenever the state of the datapath or process containing the breakpoint changes.

3.2.2.11. Clear all Breakpoints

Cause:

(1) Pick of "Clear All Breakpoints" menu item

Effect:

Trace halts. All breakpoints already present on all datapaths/processes are cleared.

3.3. Performance metrics and meters

Following is an incomplete list of performance metrics that could be included in the form of meters (digitally, or graphically in the form of bar charts or pie charts). These will be purely experimental in nature, since performance monitoring is not a primary goal of this project.

3.3.1. Relative display speed(delta)

Measure of:

Speed of display relative to speed of original program execution measured over the last "delta" trace records, giving an 'average relative speed' over a short or long period depending on the value of "delta".

Proposed Equation

$$(w\_time - st\_w\_time) / (r\_time - st\_r\_time)$$

where

"w_time" is the wall clock time from last trace record

"r_time" is the real wall clock time when last trace record was read,
"delta" is some small integer constant (3? 5? 10?)
"st_w_time" is the wall clock time from the trace record *-delta, where * is the ordinal of the last trace record read
"st_r_time" is the real wall clock time when trace record *-delta was read

Note:

This measure seems to only make sense back to the last "GO", since real wall clock time continues through a HALT while trace record wall clock time stops. Therefore, if *-delta addresses a record before the last GO, it could either be taken to be the record read after the last GO, or else the real wall clock time which has elapsed between intervening HALTs and GOs could be subtracted from the denominator.

3.3.2. Parallelism(Process_set)

Measure of:

Amount of parallelism achieved within the "Process_set".

Proposed Equation:

(Total wall clock execution time for all processes) / (Wall clock from last trace record - wall clock from first trace record - wall clock time while all processes idle)

Note:

If all processes in the Process set are on separate processors, this is a measure of speedup, since it is ratio of the time that the process would have taken on one processor to the time it took on the many processors. If the processes in the Process set are all on a single processor, this is a measure of contention among processes for CPU time. To get a speedup measure when processes/processors > 1, perhaps a similar metric could be devised using CPU time measures.

3.3.3. Processor under-utilization(Processor)

Measure of:

Percent of time a Processor is idle.

4. Practical Considerations

In our initial thinking, the monitor was to run as the host program was running, requiring bidirectional data transfer between the monitor and host systems. Monitor to host data would include the trace stream and any normal program output, while host to monitor would consist of flow control and interactive program input.

This design suffered from problems arising from the flow control. In order for the speedometer to remain accurate, the actual speed execution of the host program could not be affected by the speed of the display, meaning that a large

output buffer had to be maintained for the trace between the host program and the display system - larger than the Unix default of 512 characters. Although there are probably ways to handle this on Unix and other systems (e.g. by having the trace go to a file on the VAX and then having a separate process such as "`tail`" spool the file to the monitor), this is very system dependent and offers little advantage over simply creating a trace file which is moved over to the monitor after the trace is complete.

5. Design

5.1. Subnetwork Expansion Algorithm

A subnetwork expansion/collapse algorithm is not as trivial as it may first seem if it is to have certain desirable properties. Among these is that all bubbles (process nodes) on the screen should be of a uniform size, and the network should in some nice sense fill the area available on the screen. Furthermore, datapaths should not be made to cross or uncross because of an expansion/collapse, and datapaths should never cross over or under bubbles.

It initially seemed as though GKS could help us out by allowing us to redefine coordinate systems or by performing certain transformations for us. Unfortunately, these capabilities turned out to be of little use.

Our solution was to enclose each bubble in a square zone exactly 10 units wide by 10 units high to enclose it. (The display area is rescaled as necessary to accommodate all zones.) All datapaths arriving or departing from the bubble would meet the zone and then proceed radially to the bubble. Each bubble would keep a record of where datapaths impinged on its zone; i.e. the side (top, bottom, left, right) and the relative point along the side. In addition, zones were not allowed to overlap, and datapaths were not allowed to pass through zones.

The detail level of each subnetwork was defined the same way, with the entire subnetwork defined within a rectangle (again, scaled to accommodate all of the bubble zones). Any datapath entering or leaving the subnetwork would meet this external rectangle.

With this basis, expansion and collapse of subnetworks can be performed dealing only with the rectangular zones, with the bubbles re-added (with their centers coinciding with the rectangles' centers) after the mapping has been performed. This means that after the mapping, the restriction is that the zones must still be at least 10 by 10 (to hold the bubbles again).

The algorithm works by checking to see if the detail level network will fit into the zone of the subnetwork bubble it is replacing, one dimension at a time. If the detail level is smaller than the zone, then the subnetwork is stretched along that dimension to fit in the zone. If the detail level is larger than the zone (which is usually the case), the zone is stretched to accommodate the detail level. As the zone is stretched, all points adjacent to the zone along that

dimension (i.e. either all of the points directly above and below the zone or all the points directly to the left and right of the zone) are moved accordingly to preserve the correspondence between them and the points on the zone edge.

After the stretching is performed, the detail level is moved into the zone and the screen is rescaled if necessary to accommodate the new network. The network is re-drawn with each bubble placed at the center of it's corresponding zone.

The "stretching" mentioned above may be non-linear depending on the past history of stretches in the network, but it can always be partitioned into linear stretches.

5.2. BUBLIB - Higher level network display routines

To avoid being dependent on any given graphics package or terminal, and to avoid carrying (sometimes voluminous) calls to GKS within our higher level routines, we designed this intermediate level of graphics routines to do most of the dirty work of network display and manipulation. These routines were designed in light of both the capabilities of generally available graphics packages (e.g. segments and primitives) and the needs of our LGDF monitor.

```
SUBROUTINE INITBB(GRIDX, GRIDY, STATUS)
INTEGER GRIDX, GRIDY, STATUS
```

Undefines all segments, sets scaling factor for all further calls, such that there are at least GRIDX units horizontally and GRIDY units vertically, with the lower left hand corner coordinate (0, 0) and aspect ratio of 1. Clears screen to empty, except for control menu. STATUS is returned zero if all ok.

```
SUBROUTINE DEFBUB(CENTRX, CENTRY, SLICES, BUBNAM, BUBIDN, STATUS)
REAL CENTRX, CENTRY
INTEGER SLICES, BUBIDN, STATUS
CHARACTER BUBNAM*3
```

Defines a bubble with given center at (CENTRX, CENTRY). Radius of bubble will be 7 units. Bubble will be divided into SLICES equal pie-slices. Bubble will be labeled with BUBNAM (which may be able to be defined as a separate segment - see DRWARC with SLICE=0). Returns BUBIDN to be used in actions on bubble in further calls. STATUS is returned zero if all ok. No drawing will occur.

```
SUBROUTINE DEFARC(XYPATH, ARCIDN, STATUS)
REAL XYPATH(2, *)
INTEGER ARCIDN, STATUS
```

Defines an arc with path described in XYPATH. In general, XYPATH(1, I) is an x coordinate, XYPATH(2, I) a y coordinate. X and y coordinates are always positive. If XYPATH(1, I) is -1, it repre sents the end of one branch of the path, with other branches to follow; If -2, it represents the end of the path definition. In either case, XYPATH(2, I) represents the type of arrowhead from the following list:

XYPATH(2, I)	Type of arrowhead
0	No arrowhead
1	Open arrow
2	Closed arrow
3	V arrow

Arrowhead should be 2 units in length, and should point in approximately the right direction with the point at the last coordinate in the path.

No drawing will occur. ARCIDN returned to identify arc in subsequent calls. STATUS is returned zero if all ok.

```
SUBROUTINE DRWBUB(BUBIDN, SLICE, XSTATE, STATUS)
INTEGER BUBIDN, XSTATE, STATUS
```

Draw slice# SLICE of bubble with id BUBIDN on screen, with execution state = XSTATE as shown:

XSTATE	Execution state
1	Waiting
2	Executing
3	Done (Stopped)

If bubble is unsliced, SLICE will be equal to 1. If SLICE = 0, just the outline (whatever that means - maybe nothing) and bubble name will be drawn. (If bubbles are implemented such that outlines and names are re-drawn whenever the state changes, a call with SLICE=0 may result in no action.) The execution state will be reflected as a color. If slice is already on the screen, it will be redrawn.

```
SUBROUTINE DRWARC(ARCIDN, MTFULL, STATUS)
INTEGER ARCIDN, MTFULL, STATUS
```

Draw arc with id ARCIDN on screen with empty-full state = MTFULL as shown:

MTFULL	Empty-Full state
1	Empty
2	Full

The empty-full state will be reflected as a color. If arc is already on the screen, it will be redrawn.

```
SUBROUTINE DSPSTT(CENTRX, CENTRY, LABEL, STATUS)
INTEGER CENTRX, CENTRY, STATUS
CHARACTER LABEL*3
```

Label bubble, whose center is at CENTRX, CENTRY, with LABEL.

This routine will be used to display a bubble's internal state changes.

```
SUBROUTINE ARWOFF(DIFFX, DIFFY, OFF1X, OFF1Y, OFF2X, OFF2Y)
REAL DIFFX, DIFFY, OFF1X, OFF1Y, OFF2X, OFF2Y
```

Computes two points for back of arrow head relative to point, (OFF1X, OFF1Y) and (OFF2X, OFF2Y), given the differences in X coordinates and in Y coordinates for the last two points on the arc; i.e. if the point before the end of the arc was (125, 7) and the point at the end of the arc (the point of the arrow) is (170, 5), then DIFFX should be (170 - 125) = 45 and DIFFY = (5 - 7) = -2.

5.3. External data structures

Since this monitor was to be general enough to any LGDF program, it was logical to store the general structure of the data flow graph being monitored outside of the program in a file which, perhaps someday, would be built by another program. This program would act as a graphics editor for a user to create the graphs and subgraphs to be shown in the monitor. It is easy to conceive that this program would be capable of collecting all of the data currently held in the LGDF "wirelist" file, since most of this information is already present in the data flow graph.

The GKS system offered us one choice for external representation of the data flow graph in the form of a *metafile.* A metafile is a file which has a format known to GKS, and which can hold GKS segments. Although we intended to use segments for our display, our datapath segments had to be redefined every time a transformation (such as expand or collapse) took place on the display, and we needed the component coordinates of the segments available within the program to perform these transformations. This was not possible if using a metafile as our only form of data transfer between the graphics editor and the monitor program.

We therefore devised our own file format for such a file. This would also facilitate the extension which would have the graphics monitor accept all data currently present in the wirelist.

The file format is:

```
BUBBLE pp  nn xxxxxx yyyyyy ss wwwwww hhhhhh
ARC    ddq nn                              o
ARCPNT ddq nn xxxxxx yyyyyy
```

where

BUBBLE records describe bubbles and/or networks
ARC records describe arcs
ARCPNT records describe points on arc and arrows at terminals

pp	process number for the bubble
nn	network the bubble resides within
xxxxxx	x coordinate of the bubble or arcpoint within nn
yyyyyy	y coordinate of the bubble or arcpoint within nn
ss	number of slices in bubble
wwwwww	width of network when expanded
hhhhhh	height of network when expanded
dd	arc number for arc
q	arc qualifier (one letter or space)
o	Flag (t/f) for whether arc has an origin in nn

Restrictions

(1) All ARC records having identical nn fields must appear together (except for intervening ARCPNT records)

(2) All ARCPNT records for a particular arc must immediately follow the ARC record for the same arc

(3) The BUBBLE record for a network must appear before the the BUBBLE records for bubbles within that network and before the ARC records for arcs within that network

6. Implementation Experience

6.1. BUBLIB - Dealing within GKS

The monitor program must have available some form of specification of the program graphics. Within GKS, the simplest means of specifying a graphic image such as a bubble or arc is as a *segment.* A segment is opened, GKS routines are called to insert the appropriate graphical information, and the segment is closed. High level routines are then available to manipulate the segment. However, segments restrict further program animation graphic activity essential to the program monitor. For example, once a segment is created, its color cannot be changed.

The current implementation of the monitor bypasses this problem by creating multiple segments for each LGDF object. This is done in an initialization step. The (currently hardwired) interface file is processed, and a segments for each possible state (color) of each bubble and arc are created. The high level GKS segment display routines are then used during the actual monitoring phase. If, for example, a bubble is to be turned from green to red, the green segment of that bubble is erased and the red segment drawn. The reason for the erasure is to allow selective screen update. The GKS system allows setting of update modes and segment priorities, and these can be combined to control the time of screen updates. However, when an update is forced within GKS, the screen is cleared and *all visible segments* are redrawn. This is totally infeasible due to the slowness of redraw. A sample set of trace actions is also hardwired into the monitor code.

An alternative to using segments is to write routines to mimic GKS segment storage and display. Arcs and bubbles would be preprocessed in the setup phase of the monitor. A data structure would be associated with each arc and bubble to hold all precomputable graphical data, in order to minimize the amount of computation to be done and the number of GKS routines to be called at display time. It is conceivable that this approach might result in faster execution than the current implementation, but this has not been tested since it is unlikely and development would be lengthy.

6.2. Program Development Experiences

Program development on an IBM PC/XT was unpleasant at best. Though compilation was tolerably fast, linking was intolerably slow. The GKS kernel is quite large. A 500-statement program with 130 subroutine calls (most to GKS routines) took half an hour to link, and linking time increases with program size. 500 lines is small compared to the code size of a full-blown monitor. In addition, the experimental nature of the code (we were not adapting any pre-existing code), the poor documentation of some of the graphics library and slow graphics display hardware make the "let's try it and see" (frequent write/debug/test cycles) method of programming a necessity. The long linking time mades this process agonizing.

7. Future Possibilities/Suggestions

7.1. LGDF Monitor - Conclusions and Recommendations

The resulting demonstration software is SLOW, probably too slow to be of any practical use. This seemed to be a result of two aspects of the GKS system;

(1) It uses abundant floating point arithmetic, even though we could have supplied all points as integer values without much problem.

(2) There was no straightforward way of changing colors without redrawing the entire figure. Even then, it was necessary (when using segments) to erase the old figure before redrawing the new one, which took even more time, especially with the bubbles.

It is possible that upgrading to a faster machine (PC/AT with a math coprocessor) would help some. However, the mismatch between our application and the functions offered by GKS seemed to be the primary problem.

Resources for the project ran out before we were able to try out some portions of the design. The poor performance of the PC as a development system was a major cause of this.

A more useful configuration for implementation of a system such as this would be to have a mainframe doing most of the dirty work with a smart graphics terminal performing the monitoring. Perhaps, under this condition, GKS could run on the mainframe at a reasonable rate and development would also go smoother. If a personal computer configuration is desired, lower-level graphics should be considered, and perhaps a computer with more of a graphics orientation such as the Amiga, Apple IIgs, or Macintosh.

7.2. A Multi-level Debugger/Testbed

This project certainly opened our eyes to some possible extensions in the realm of real time monitoring of parallel programs. Perhaps the most intriguing of these became apparent while considering the kinds of interaction the user could have with the program while it was running.

In most parallel programming environments, the relative timing of the separate processes can affect the results of the program, yielding the nondeterministic behavior which these programs have become infamous for. Adding tracing to such programs can affect these relative timings, thereby frequently affecting the results of the program. To minimize this problem, it is important that tracing be as unobtrusive as possible.

An interesting technique that might be used with such a program would be to trace only enough of the execution so that it could be reconstructed with the same relative timings at a later time. This reconstructed execution could then be slowed down or monitored in any more detailed way desired, as long as the important relative timings occur the same way as they did during the actual execution. Although the trace itself would contain very little information, the trace together with the original program and original input data could yield a wealth of information.

It may not even be necessary for the reconstruction to take place on the same hardware that originally ran the program. A program that was originally run on a parallel processor could be recreated under a multiprocessing operating system, like Unix, as long as the compiler and machine arithmetic were similar. The tool performing the reconstruction could, in fact, allow the user to call a

standard interactive debugger (such as dbx on Unix) to allow the user to set breakpoints and/or monitor values during the execution.

The LGDF programming model provides an excellent base for such a system. Since LGDF programs have very few opportunities for timing to modify results, there are very few places where tracing needs to occur. During execution of the reconstruction, the user could watch the execution from a high level monitor, such as the one we have developed here, until the program enters a particularly important phase, at which time the user could query the data available on data paths directly from this high level or could opt to invoke the standard interactive debugger.

In fact, an LGDF program is so stable with respect to relative timing that it may be desirable in some cases to skip the tracing phase and simply run the monitored program in a testbed environment, where the user could dictate when processes should be held from executing. In addition, the user may want to manually deposit data on a datapath or modify a variable within a program. The tool controlling these interactions could possibly warn the user when the program acts in a non-deterministic way (i.e. when there is a choice of two processes that can start, both of which access a common data path).

8. Listings

```
C
C *** COLORS
C
      integer BACKGR,FOREGR,RED,GREEN
      integer BLUE,YELLOW,ORANGE,VIOLET

C
C *** DEFINES
C
      integer BUNDLED,INDIVIDUAL
      integer WC,NDC
      integer FHOLLOW,FSOLID,FPATTERN,FHATCH
      integer SOLID,DASHED,DOTTED
      integer POINT,PLUS,ASTERISK,OMARK
      integer NONE,OK,NOPICK
      integer POSTPONE,PERFORM
      integer DETECTABLE
      integer HIGHLIGHT
      integer INVISIBLE,VISIBLE
      integer HNORMAL,HLEFT,HCENTER,HRIGHT
      integer VNORMAL,VTOP,VCAP,VHALF,VBASE,VBOTTOM
      integer RIGHT,LEFT,UP,DOWN
      integer STRING,CHAR,STROKE
      integer HIGHER,LOWER
      integer BAR,ARC,PIE,CIRCLE
C
C *** DEVICES
C
      integer wssdev,crtdev,joydev
      integer gmodev,gmidev

C
C *** FONTS
C
      integer FONT0,FONT1,FONT2,FONT3
      integer FONT4,FONT5,FONT6

```

```
 1  C
 2  C *** COLORS
 3  C
 4     data BACKGR,FOREGR,RED,GREEN /0,1,2,3/
 5     data BLUE,YELLOW,ORANGE,VIOLET /4,5,6,7/
 6
 7
 8  C
 9  C *** DEFINES
10  C
11     data BUNDLED,INDIVIDUAL /0,1/
12     data WC,NDC /0,1/
13     data FHOLLOW,FSOLID,FPATTERN,FHATCH /0,1,2,3/
14     data SOLID,DASHED,DOTTED /1,2,3/
15     data POINT,PLUS,ASTERISK,OMARK /1,2,3,4/
16     data NONE,OK,NOPICK /0,1,2/
17     data POSTPONE,PERFORM /0,1/
18     data DETECTABLE /1/
19     data HIGHLIGHT /1/
20     data INVISIBLE,VISIBLE /0,1/
21     data HNORMAL,HLEFT,HCENTER,HRIGHT /0,1,2,3/
22     data VNORMAL,VTOP,VCAP,VHALF,VBASE,VBOTTOM /0,1,2,3,4,5/
23     data RIGHT,LEFT,UP,DOWN /0,1,2,3/
24     data STRING,CHAR,STROKE /0,1,2/
25     data HIGHER,LOWER /0,1/
26     data BAR,ARC,PIE,CIRCLE /-1,-2,-3,-4/
27  C
28  C *** DEVICES
29  C
30     data wssdev,crtdev,joydev /0,1,2/
31     data gmodev,gmidev /4,5/
32
33  C
34  C *** FONTS
35  C
36     data FONT0,FONT1,FONT2,FONT3 /1,-101,-102,-103/
37     data FONT4,FONT5,FONT6 /-104,-105,-106/
38
```

```
$STORAGE:2

        program lgdf

c*************************************************************************
c                                                                        *
c                          DECLARATIONS                                  *
c                                                                        *
c*************************************************************************
c
c GKS constants
$INCLUDE:'lgdf.def'

c DO NOT REMOVE THE LINE BELOW!!! (Used by GKS for scratch memory)
        integer*4 size,intary(5000)

c polyline aspect source flags
        integer plasf(13)

c coordinates used in transformation computations:
c maximum device / normalized device
        real  xdcmax, ydcmax, xndc, yndc

c values used in computing segment names:
c numsgs is the current number of segments in existence, sgstrt is
c the last segment created for the control portion of the display
c control section: 0 through sgstrt / lgdf graph: sgstrt+1 to numsgs
        integer numsgs, sgstrt

c VARIABLES FOR TEST STUFF:
c   centers, number of slices, id numbers, and names of the 7 bubbles
        real center(7,2)
        integer slices(7),bubidns(7)
        character*3 bubnams(7)
c   specification points and id numbers of the 14 arcs
        real arcs(2,59)
        integer arcidns(14)
c   standard array for passing an arc to the defarc subroutine
        real xypath(2,50)
c   catches arc and bubble ids returned by defining subroutines
        integer idn
c   catches status returned by defining subroutines
        integer status
c   plenty of loop indices for doing test runs
        integer i,j,k

c*************************************************************************
c                                                                        *
c                          COMMON BLOCKS                                 *
c                                                                        *
c*************************************************************************
c
c DO NOT REMOVE THE LINE BELOW!!! (Used by GKS for scratch memory)
        common /gracom/ size,intary

        common /dcmax/  xdcmax, ydcmax, xndc, yndc
```

```
      common /segs/   numsgs, sgstrt

c*************************************************************************
c                                                                        *
c                          DATA STATEMENTS                               *
c                                                                        *
c*************************************************************************
c
$INCLUDE:'lgdf.dat'

      data plasf /0,0,1,1,1,1,1,1,1,1,1,1,1/

c DATA FOR TEST STUFF:
c   centers of circles (all x's then all y's)
c                        p12  p13   p14  p16  p17  p18  p10
      data center /14.0,35.0,105.0,42.0,63.0,84.0,14.0,
     *             59.0,59.0, 59.0,35.0,35.0,35.0,13.0/
c   number of slices in each circle
c                         p12  p13   p14  p16  p17  p18  p10
      data slices /  1,    1,    1,   1,   6,   1,   1 /
c   names of the bubbles
      data bubnams /'p12','p13','p14','p16','p17','p18','p10'/
c   specification points for arcs ((x,y) pairs)
      data arcs /
c   arc#1  d01
     *                                1.0,59.0,   7.0,59.0, -2.0,3.0,
c   arc#2  d02
     *                               21.0,59.0,  28.0,59.0, -2.0,2.0,
c   arc#3  d10
     *                               42.0,59.0,  98.0,59.0, -2.0,2.0,
c   arc#4  d07
     *                              112.0,59.0, 119.0,59.0, -2.0,3.0,
c   arc#5  d06
     *  19.0,54.0,  22.0,47.0, 52.5,47.0,  58.0,40.0, -1.0,1.0,
     *              52.5,47.0, 73.5,47.0,  79.0,40.0, -2.0,1.0,
c   arc#6  d11
     *  40.0,54.0,  42.0,53.0, 75.0,53.0,  84.0,42.0, -2.0,2.0,
c   arc#7  d12
     *                               35.0,52.0,  42.0,42.0, -2.0,2.0,
c   arc#8  d06a
     *                               89.0,40.0, 100.0,54.0, -2.0,2.0,
c   arc#9  d13
     *                               49.0,35.0,  56.0,35.0, -2.0,2.0,
c   arc#10 d14
     *                               70.0,35.0,  77.0,35.0, -2.0,2.0,
c   arc#11 d05
     *                          20.0,16.5, 49.0,16.5,  58.0,30.0, -2.0,1.0,
c   arc#12 d04
     *                          21.0,13.0, 52.5,13.0,  63.0,28.0, -2.0,1.0,
c   arc#13 d03
     *  20.0, 9.5,  56.0, 9.5, 68.0,26.0,  68.0,30.0, -1.0,1.0,
     *              56.0, 9.5, 87.0, 9.5, 105.0,52.0, -2.0,1.0,
c   arc#14 d15
     *                          19.0,64.0, 24.5,69.5, 119.0,69.5, -2.0,1.0/

```

```
113  c*************************************************************************
114  c                                                                        *
115  c                          EXECUTABLE CODE                               *
116  c                                                                        *
117  c*************************************************************************
118  c
119  c DO NOT REMOVE THE LINE BELOW!!! (Used by GKS for scratch memory)
120  c AND IT MUST BE FIRST LINE OF EXECUTABLE CODE!!!
121          size = 10000
122
123  c GKS initialization, open files
124          call init
125
126  c set normalization transformations
127          call setnrm (crtdev)
128
129  c set polyline aspect source flags and polyline index
130          call gsasf (plasf)
131
132  c draw graph area box (seg #1)
133          call box (1,1,1,0.0, 70.0, 0.0, 70.0)
134
135  c draw control area box (seg #2)
136          call box (3,2,2,0.0, 100.0, 0.0, 20.0)
137
138  c draw control segments (segments numbered 3 through sgstrt)
139  c
140  c       NOT IMPLEMENTED
141  c
142
143  c set sgstrt and numsgs to proper values (done below for the case
144  c that the control section ended on segment 10)
145          sgstrt = 10
146          numsgs = sgstrt
147
148  c TEST STUFF
149  c   the lgdf graph is bounded by a 120x75 box in world coordinates
150  c   initbb initializes the necessary transformation, after cleaning
151  c   up after any previous graph displayed
152          call initbb(120.0,75.0,status)
153
154  c   define the arc segments:
155  c     read the arc specification points into the xypath array
156  c     at the end of each arc, call the defarc subroutine
157  c     save the arc id numbers for later use
158          j = 1
159          k = 1
160          do 10 i=1,59
161             xypath(1,j) = arcs(1,i)
162             xypath(2,j) = arcs(2,i)
163             j = j+1
164             if (arcs(1,i) .eq. -2) then
165                  call defarc(xypath,idn,status)
166                  arcidns(k) = idn
167                  k = k+1
168                  j = 1
```

```
169               endif
170          10   continue
171
172     c    define the bubble segments:
173     c      send the center and number of slices for each circle to defbub
174     c      save the bubble id numbers for later use
175               do 20 i = 1,7
176                  call defbub(center(i,1),center(i,2),
177           *                        slices(i),bubnams(i),idn,status)
178                  bubidns(i) = idn
179          20   continue
180
181     c    draw each slice for each bubble in suspended state (red) using drwbub
182               do 30 i = 1,7
183                  do 40 k = 1,slices(i)
184                       call drwbub(bubidns(i),k,1,status)
185          40           continue
186          30   continue
187
188     c    draw each arc in empty state (red) using drwarc
189               do 50 i = 1,14
190                  call drwarc(arcidns(i),1,status)
191          50   continue
192
193     c    change a few for fun and to see how it's working
194
195     c    set d01
196               call drwarc(arcidns(1),2,status)
197     c    wake up p12
198               call drwbub(bubidns(1),1,2,status)
199     c    set d06
200               call drwarc(arcidns(5),2,status)
201     c    set d02
202               call drwarc(arcidns(2),2,status)
203     c    wake up p13
204               call drwbub(bubidns(2),1,2,status)
205     c    set d12
206               call drwarc(arcidns(7),2,status)
207     c    wake up p16
208               call drwbub(bubidns(4),1,2,status)
209     c    clear d12
210               call drwarc(arcidns(7),1,status)
211     c    set d13
212               call drwarc(arcidns(9),2,status)
213     c    wake up p17d
214               call drwbub(bubidns(5),4,2,status)
215     c    terminate p16
216               call drwbub(bubidns(4),1,3,status)
217
218     c END TEST STUFF
219
220     c clear screen (suppresses final redraw)
221     c         call gclrwk (crtdev,1)
222
223     c shut down GKS and close files
224               call shutd
```

```
225
226  c insert code here to call "mode co80" to enable proper text display
227  c for subsequent programs such as editors
228  c
229  c       NOT IMPLEMENTED
230  c
231
232          end
233
234  c*************************************************************************
235  c                                                                        *
236  c                          SUBROUTINE INIT                               *
237  c                                                                        *
238  c       1. Open files                                                    *
239  c       2. Initialize GKS                                                *
240  c       3. Call "iniarw"                                                 *
241  c                                                                        *
242  c*************************************************************************
243          subroutine init
244
245  $INCLUDE:'lgdf.def'
246          integer unit1
247
248  $INCLUDE:'lgdf.dat'
249          data unit1 /14/
250
251  c file used by GKS to log error messages
252          open (unit1,file='errors',status='new')
253
254  c file for debug write statements
255          open (15,file='debugs',status='new')
256
257  c open kernel system for business
258          call gopks (unit1, 1024)
259
260  c open workstations
261          call gopwk (wssdev, 0, wssdev)
262          call gopwk (crtdev, 0, crtdev)
263          call gopwk (gmodev, 0, gmodev)
264          call gopwk (joydev, 0, joydev)
265
266  c activate workstations
267          call gacwk (wssdev)
268          call gacwk (crtdev)
269          call gacwk (gmodev)
270          call gacwk (joydev)
271
272  c suppress display updates unless asked for
273          call gsds (crtdev,0,0)
274
275  c compute tables for use in computing the angle
276  c of arrowheads at the end of arcs
277          call iniarw
278
279          return
280          end
```

```
281
282  c*************************************************************************
283  c                                                                        *
284  c                          SUBROUTINE SETNRM                             *
285  c                                                                        *
286  c       1. Define transformation (#1) for graph area                     *
287  c       2. Define transformation (#2) for control area                   *
288  c                                                                        *
289  c*************************************************************************
290          subroutine setnrm (dev)
291
292  $INCLUDE:'lgdf.def'
293          integer dev
294          integer err, dcunit, xras, yras
295          real xdcmax, ydcmax, scale, xndc, yndc
296
297          common /dcmax/  xdcmax, ydcmax, xndc, yndc
298
299  $INCLUDE:'lgdf.dat'
300
301  c inquire maximum display surface size
302          call gqdsp (dev, err, dcunit, xdcmax, ydcmax, xras, yras)
303
304  c calculate the aspect ratio of display surface
305          if (xdcmax .GT. ydcmax) then
306              scale = xdcmax
307          else
308              scale = ydcmax
309          end if
310          xndc  = xdcmax / scale
311          yndc  = ydcmax / scale
312
313  c set world window and viewport for transformation 1 (graph area)
314          call gswn (1, 0.0, 70.0, 0.0, 70.0)
315          call gsvp (1, 0.0, xndc, 0.21*yndc, yndc)
316
317  c set world window and viewport for transformation 2 (control area)
318          call gswn (2, 0.0, 100.0, 0.0, 20.0)
319          call gsvp (2, 0.0, xndc, 0.0, 0.20*yndc)
320
321  c set display window and viewport
322          call gswkwn (dev, 0.0, xndc, 0.0, yndc)
323          call gswkvp (dev, 0.0, xdcmax, 0.0, ydcmax)
324
325          return
326          end
327
328  c*************************************************************************
329  c                                                                        *
330  c                          SUBROUTINE BOX                                *
331  c                                                                        *
332  c       1. Define a box segment at the indicated coordinates,            *
333  c          using the indicated line attributes and transformation        *
334  c                                                                        *
335  c*************************************************************************
336          subroutine box (pli,trnum,sgnm,xmin,xmax,ymin,ymax)
```

```
337
338   $INCLUDE:'lgdf.def'
339           integer pli,trnum,sgnm
340           real xmin, ymin, xmax, ymax
341   c
342           real x(5), y(5)
343   $INCLUDE:'lgdf.dat'
344
345   c set polyline index
346           call gspli(pli)
347
348   c set normalization transformation
349           call gselnt (trnum)
350
351           call gcrsg (sgnm)
352           x(1) = xmin
353           y(1) = ymin
354           x(2) = xmax
355           y(2) = ymin
356           x(3) = xmax
357           y(3) = ymax
358           x(4) = xmin
359           y(4) = ymax
360           x(5) = xmin
361           y(5) = ymin
362
363   c draw the box
364           call gpl (5, x, y)
365
366           call gclsg
367           return
368           end
369
370   c*************************************************************************
371   c                                                                        *
372   c                          SUBROUTINE INITBB                             *
373   c                                                                        *
374   c       1. Undefine all graph segments                                   *
375   c       2. Define a transformation which maps GRIDXxGRIDY                *
376   c          onto the graph display with aspect ratio 1                    *
377   c          and set this transformation as the current one                *
378   c          for all subsequent output                                     *
379   c       3. Clear the graph area of the display                           *
380   c       4. Return STATUS                                                 *
381   c                                                                        *
382   c*************************************************************************
383           subroutine initbb (gridx,gridy,status)
384
385           real gridx,gridy
386           integer status
387           integer numsgs, sgstrt
388           real xdcmax,ydcmax,xndc,yndc
389
390           common /segs/ numsgs, sgstrt
391           common /dcmax/ xdcmax,ydcmax,xndc,yndc
392
```

```
c delete all graph segments currently existing
      do 10 i = sgstrt+1, numsgs
         call gdsg(i)
10    continue
      numsgs = sgstrt

c calculate aspect ratio
c
c       NOT IMPLEMENTED
c

c set world window and viewport
      call gswn(3,0.0,gridx,0.0,gridy)
      call gsvp(3,0.01*xndc,0.99*xndc,0.21*yndc,0.99*yndc)
      call gselnt(3)

c clear graph display
c
c       NOT IMPLEMENTED
c

c return status (0 if ok)
c
c       NOT IMPLEMENTED
c

      return
      end

c*******************************************************************************
c                                                                              *
c                          SUBROUTINE DEFBUB                                   *
c                                                                              *
c       1. Define a bubble with center (CENTRX,CENTRY) and radius 7.           *
c          Create three segments for each slice, via "doarc" and               *
c          "docirc". Associate BUBNAM with each segment.                       *
c       2. Return BUBIDN and STATUS                                            *
c                                                                              *
c*******************************************************************************
      subroutine defbub(centrx,centry,slices,bubnam,bubidn,status)
      real centrx,centry
      integer slices, bubidn, status
      character bubnam*3
$INCLUDE:'lgdf.def'
      character datrec(1)
      real rad
      real xarc(3),yarc(3)
      real xcir(2),ycir(2)
      integer i
      real angle
      real pi
      real tempx,tempy

      integer numsgs, sgstrt
      common /segs/ numsgs, sgstrt

```

```
$INCLUDE:'lgdf.dat'
        data rad /7.0/
        data pi  /3.14159/

c compute bubidn
        numsgs = numsgs + 1
        bubidn = numsgs
        if (slices .gt. 1) then
           tempx = rad*1.0 + centrx
           tempy = 0.0 + centry
           do 10 i=1,slices
                xarc(1) = centrx
                yarc(1) = centry
                xarc(2) = tempx
                yarc(2) = tempy
                angle = (2.0*pi*i)/slices
                tempx = rad*cos(angle) + centrx
                tempy = rad*sin(angle) + centry
                xarc(3) = tempx
                yarc(3) = tempy
                call doarc(xarc,yarc,RED,bubnam)
                call doarc(xarc,yarc,GREEN,bubnam)
                call doarc(xarc,yarc,VIOLET,bubnam)
   10       continue
        else
                xcir(1) = centrx
                ycir(1) = centry
                xcir(2) = centrx + rad
                ycir(2) = centry
                call docirc(xcir,ycir,RED,bubnam)
                call docirc(xcir,ycir,GREEN,bubnam)
                call docirc(xcir,ycir,VIOLET,bubnam)
        endif

c return 0 status if ok (add error checking)
        status = 0
        return
        end

c*******************************************************************************
c                                                                              *
c                          SUBROUTINE DOCIRC                                   *
c                                                                              *
c       Create a bubble/label segment                                          *
c                                                                              *
c*******************************************************************************
        subroutine docirc(xcir,ycir,ccolor,bubnam)
        real xcir(2),ycir(2)
        integer ccolor
        character*3 bubnam
$INCLUDE:'lgdf.def'
        character datrec(1)
        real rad

        integer numsgs, sgstrt
        common /segs/ numsgs, sgstrt
```

```
505
506   $INCLUDE:'lgdf.dat'
507           data rad /7.0/
508
509           call gsfais(FSOLID)
510           call gsfaci(ccolor)
511           numsgs = numsgs + 1
512           call gcrsg(numsgs)
513           call gsdtec(numsgs,DETECTABLE)
514           call gsvis(numsgs,INVISIBLE)
515           call ggdp(2,xcir,ycir,CIRCLE,0,datrec)
516
517   c draw label of circle
518           call gstxci(YELLOW)
519           call gstxal(HCENTER,VHALF)
520           call gtxs(xcir(1),ycir(1),3,bubnam)
521
522           call gclsg
523
524           return
525           end
526
527   c*************************************************************************
528   c                                                                        *
529   c                          SUBROUTINE DOARC                              *
530   c                                                                        *
531   c       Create an arc/label segment.                                     *
532   c                                                                        *
533   c*************************************************************************
534           subroutine doarc(xarc,yarc,ccolor,bubnam)
535           real xarc(3),yarc(3)
536           integer ccolor
537           character*3 bubnam
538   $INCLUDE:'lgdf.def'
539           character datrec(1)
540           real rad
541
542           integer numsgs, sgstrt
543           common /segs/ numsgs, sgstrt
544
545   $INCLUDE:'lgdf.dat'
546           data rad /7.0/
547
548           call gsfais(FSOLID)
549           call gsfaci(ccolor)
550           numsgs = numsgs + 1
551           call gcrsg(numsgs)
552           call gsdtec(numsgs,DETECTABLE)
553           call gsvis(numsgs,INVISIBLE)
554           call ggdp(3,xarc,yarc,PIE,0,datrec)
555   c draw label of circle
556           call gstxci(YELLOW)
557           call gstxal(HCENTER,VHALF)
558           call gtxs(xarc(1),yarc(1),3,bubnam)
559
560           call gclsg
```

```
561
562           return
563           end
564
565   c*************************************************************************
566   c                                                                        *
567   c                          SUBROUTINE DRWBUB                             *
568   c                                                                        *
569   c       Draw a bubble (or slice) in the specified state. This is done    *
570   c       by erasing the other states of the bubble before display.        *
571   c       (Only one needs to be erased if the state of the display         *
572   c       is known).                                                       *
573   c                                                                        *
574   c*************************************************************************
575           subroutine drwbub(bubidn,slice,xstate,status)
576           integer bubidn,slice,xstate,status
577   $INCLUDE:'lgdf.def'
578           integer sgnm
579           integer base
580           real tmat(2,3)
581           integer err,vis,high,det
582           real sgpr
583           integer numsgs,sgstrt
584           common /segs/ numsgs,sgstrt
585   $INCLUDE:'lgdf.dat'
586
587           base = bubidn + (slice-1)*3
588
589           if (xstate .eq. 1) then
590                   call gsvis(base+2,INVISIBLE)
591                   call gsvis(base+3,INVISIBLE)
592                   call gsvis(base+1,VISIBLE)
593           endif
594           if (xstate .eq. 2) then
595                   call gsvis(base+3,INVISIBLE)
596                   call gsvis(base+1,INVISIBLE)
597                   call gsvis(base+2,VISIBLE)
598           endif
599           if (xstate .eq. 3) then
600                   call gsvis(base+1,INVISIBLE)
601                   call gsvis(base+2,INVISIBLE)
602                   call gsvis(base+3,VISIBLE)
603           endif
604
605           return
606           end
607
608
609
610
611
612
613
614
615
616
```

```
c*************************************************************************
c                                                                        *
c                          SUBROUTINE DEFARC                             *
c                                                                        *
c       Define an arc with path XYPATH. Create two segments for          *
c       the arc via "defpth", and create a label segment.                *
c       No drawing will occur.  ARCIDN is returned to identify           *
c       the arc in subsequent calls.  STATUS is returned zero            *
c       if all ok.                                                       *
c                                                                        *
c*************************************************************************
        subroutine defarc(xypath,arcidn,status)
        real xypath(2,50)
        integer arcidn, status
$INCLUDE:'lgdf.def'

        integer numsgs, sgstrt
        common /segs/ numsgs, sgstrt

$INCLUDE:'lgdf.dat'

        numsgs = numsgs + 1
        arcidn = numsgs
        call gcrsg(numsgs)
        call gsvis(numsgs,INVISIBLE)
        call gstxci(YELLOW)
        call gstxal(HCENTER,VHALF)
        call gtxs(1.0,1.0,1,'a')
        call gclsg

        call defpth(xypath,RED,status)
        call defpth(xypath,GREEN,status)

        return
        end

c*************************************************************************
c                                                                        *
c                          SUBROUTINE DEFPTH                             *
c                                                                        *
c       Trace the datapath and output polylines as appropriate.          *
c       XYPATH(1, I) is an x coordinate, XYPATH(2, I) a y coordinate.    *
c       X and y coordinates are always positive. If XYPATH(1, I) is      *
c       -1, it represents the end of one branch of the path, with        *
c       other branches to follow; If -2, it represents the end of the    *
c       path definition. In either case, XYPATH(2, I) represents the     *
c       type of arrowhead and "arwhed" is called to output this.         *
c                                                                        *
c*************************************************************************
        subroutine defpth(xypath,ccolor,status)
        real xypath(2,50)
        integer ccolor, status
$INCLUDE:'lgdf.def'
        real xarc(10),yarc(10)
        integer length

```

```
      integer sgstate
c index into xypath array
      integer i

      integer j

      integer numsgs, sgstrt
      common /segs/ numsgs, sgstrt

$INCLUDE:'lgdf.dat'

      call gspli(1)
      call gsplci(ccolor)
      call gsfaci(ccolor)
      call gspmci(ccolor)
      call gsmk(1)
      call gsmksc(1.1)
      i = 1
      length = 0
      numsgs = numsgs + 1
      call gcrsg(numsgs)
      call gsvis(numsgs,INVISIBLE)

 10   continue
      if (xypath(1,i) .eq. -2) then
         do 98 j = length+1,10
              xarc(j) = xarc(length)
              yarc(j) = yarc(length)
 98      continue
         call gpl(10,xarc,yarc)
         call arwhed(xarc,yarc,length,xypath(2,i))
         call gclsg
         goto 20
      endif
      if (xypath(1,i) .eq. -1) then
         do 99 j = length+1,10
              xarc(j) = xarc(length)
              yarc(j) = yarc(length)
 99      continue
         call gpl(10,xarc,yarc)
         call arwhed(xarc,yarc,length,xypath(2,i))
         i = i+1
         length = 0
         xm = xypath(1,i)
         ym = xypath(2,i)
         call gpm(1,xm,ym)
         goto 10
      endif
      length = length + 1
      xarc(length) = xypath(1,i)
      yarc(length) = xypath(2,i)
      i = i+1
      goto 10

 20   continue
      return
```

```
729           end
730
731   c*************************************************************************
732   c                                                                        *
733   c                          SUBROUTINE ARWHED                             *
734   c                                                                        *
735   c       Output an arrowhead of the TYPE:                                 *
736   c                0       No arrowhead                                    *
737   c                1       Open arrow                                      *
738   c                2       Closed arrow                                    *
739   c                3       V arrow                                         *
740   c       Compute the location of the arrowhead using "arwoff".            *
741   c       An arrowhead is 2 units in length, and points in                 *
742   c       approximately the right direction with the point at              *
743   c       the last coordinate in the path.                                 *
744   c                                                                        *
745   c*************************************************************************
746           subroutine arwhed(xarc,yarc,last,type)
747           real xarc(10),yarc(10)
748           integer last
749           real type
750   $INCLUDE:'lgdf.def'
751
752   c index into x and y arc arrays
753           integer i
754           real xloc(3),yloc(3)
755           real x1,y1,dx,dy
756
757   $INCLUDE:'lgdf.dat'
758
759   c compute points in arrowhead
760           x1 = xarc(last-1)
761           y1 = yarc(last-1)
762           xloc(2) = xarc(last)
763           yloc(2) = yarc(last)
764           dx = x1-xloc(2)
765           dy = y1-yloc(2)
766           call arwoff(dx,dy,xloc(1),yloc(1),xloc(3),yloc(3))
767           xloc(1) = xloc(1) + xloc(2)
768           yloc(1) = yloc(1) + yloc(2)
769           xloc(3) = xloc(3) + xloc(2)
770           yloc(3) = yloc(3) + yloc(2)
771
772           if (type .eq. 1) then
773              call gsfais(FHOLLOW)
774              call gfa(3,xloc,yloc)
775           endif
776           if (type .eq. 2) then
777              call gsfais(FSOLID)
778              call gfa(3,xloc,yloc)
779           endif
780           if (type .eq. 3) then
781              call gpl(3,xloc,yloc)
782           endif
783
784           return
```

```
785           end
786
787   c*****************************************************************************
788   c                                                                              *
789   c                          SUBROUTINE DRWARC                                   *
790   c                                                                              *
791   c       Draw an arc in the specified state.  This is done by                   *
792   c       erasing the other state of the arc before display.                     *
793   c                                                                              *
794   c*****************************************************************************
795           subroutine drwarc(arcidn,mtfull,status)
796           integer arcidn,mtfull,status
797   $INCLUDE:'lgdf.def'
798   $INCLUDE:'lgdf.dat'
799
800           if (mtfull .eq. 1) then
801                   call gsvis(arcidn+2,INVISIBLE)
802                   call gsvis(arcidn+1,VISIBLE)
803           endif
804           if (mtfull .eq. 2) then
805                   call gsvis(arcidn+1,INVISIBLE)
806                   call gsvis(arcidn+2,VISIBLE)
807           endif
808
809   c       call gsvis(arcidn,VISIBLE)
810           return
811           end
812
813   c*****************************************************************************
814   c                                                                              *
815   c                          SUBROUTINE SHUTD                                    *
816   c                                                                              *
817   c       1. Close down the Kernel System.                                       *
818   c       2. Close files.                                                        *
819   c                                                                              *
820   c*****************************************************************************
821           subroutine shutd
822   $INCLUDE:'lgdf.def'
823           integer unit1
824   $INCLUDE:'lgdf.dat'
825           data unit1 /14/
826   c
827           call gdawk (wssdev)
828           call gdawk (crtdev)
829           call gdawk (gmodev)
830           call gdawk (joydev)
831           call gclwk (wssdev)
832           call gclwk (crtdev)
833           call gclwk (gmodev)
834           call gclwk (joydev)
835           call gclks
836           close (unit1)
837           close (15)
838           return
839           end
840
```

```
841  c*************************************************************************
842  c                                                                        *
843  c                          SUBROUTINE INIARW                             *
844  c                                                                        *
845  c       Set up arrow offsets and angles tables.                          *
846  c                                                                        *
847  c*************************************************************************
848        subroutine iniarw
849
850        real             arwofa(5), arwofb(5), slp1v2, slp2v3
851        common /arwcom/  arwofa,    arwofb,    slp1v2, slp2v3
852        integer          arwang
853        real             radian
854
855        parameter (pi = 3.14159265358979323846264433)
856  c
857  c - - these are x, y offsets for arrow tips
858  c
859        do 1000 arwang = 1, 5
860          radian = (arwang-2)*pi/8
861          arwofa(arwang) = cos(radian) * 2
862          arwofb(arwang) = sin(radian) * 2
863   1000 continue
864  c
865  c - - these are the slopes of the lines dividing angles 1 - 2, and
866  c - - 2 - 3, for determining which arrow angle most closely matches
867  c - - slope of line arrow is to be added to
868  c
869        slp1v2 = tan(pi/16)
870        slp2v3 = tan(3*pi/16)
871        return
872        end
873
874  c*************************************************************************
875  c                                                                        *
876  c                          SUBROUTINE ARWOFF                             *
877  c                                                                        *
878  c       Finds appropriate coordinates for an arrowhead, based            *
879  c       on the direction in which it points.                             *
880  c                                                                        *
881  c*************************************************************************
882        subroutine arwoff(diffx, diffy, off1x, off1y, off2x, off2y)
883        real diffx, diffy
884        real off1x, off1y, off2x, off2y
885        real xsign, ysign, slope
886  c
887  c  computes two points for back of arrow head relative to point,
888  c  (off1x, off1y) and (off2x, off2y), given the differences in x
889  c  coordinates and in y coordinates for the last two points on the
890  c  arc; i.e. if the point before the end of the arc was (125, 7) and
891  c  the point at the end of the arc (the point of the arrow) is (170, 5),
892  c  then diffx should be (170 - 125) = 45 and diffy = (5 - 7) = -2.
893  c
894  c      - - translate slope to first quadrant (to avoid overflow)
895  c
896        xsign = diffx
```

```
      diffx = abs(diffx)
      ysign = diffy
      diffy = abs(diffy)
      if (diffx .gt. diffy) then
        slope = diffy / diffx
      else
        slope = diffx / diffy
      end if
c
c      - - get offsets for base of arrow (switching x and y to put
c      - - it in right half of quadrant, if necessary)
c
      if (diffx .gt. diffy) then
        call offcpy(slope, xsign, ysign, off1x, off1y, off2x, off2y)
      else
        call offcpy(slope, ysign, xsign, off1y, off1x, off2y, off2x)
      end if

      return
      end

      subroutine offcpy(slope, sgna, sgnb, off1a, off1b, off2a, off2b)
      real              slope, sgna, sgnb, off1a, off1b, off2a, off2b
      integer           arwslp
c
c      - - these are x and y offsets for points on a circle with
c      - - radius 2 for every pi/8 radians (22.5 degrees) starting
c      - - at "arrow slope 0" (-22.5 degrees) to "arrow slope 4"
c      - - (67.5 degrees) in a counter-clockwise direction.  the idea
c      - - is (1) figure which arrow slope 1-3 best fits the slope of
c      - - the line, and (2) make the sides of the arrow head line up
c      - - with the adjacent arrow slopes.
c
      real            arwofa(5), arwofb(5), slp1v2, slp2v3
      common /arwcom/ arwofa,    arwofb,    slp1v2, slp2v3
c
c      - - figure which arrow angle will be closest
c
      if (slope .lt. slp1v2) then
        arwslp = 1
      else if (slope .lt. slp2v3) then
        arwslp = 2
      else
        arwslp = 3
      end if

      off1a = sign(arwofa(arwslp),     sgna)
      off1b = sign(arwofb(arwslp),     sgnb)
      off2a = sign(arwofa(arwslp + 2), sgna)
      off2b = sign(arwofb(arwslp + 2), sgnb)
      if (arwslp .eq. 1) off2b = -off2b

      return
      end
```

```
 1  #wp10 s00     e0000.216666w3826.010000
 2  #sp10 s00d03  e0000.250000w3826.050000
 3  #sp10 s00d04  e0000.266666w3826.080000
 4  #sp10 s00d05  e0000.283333w3826.120000
 5  #np10 s00     e0000.300000w3826.180000
 6  #wp12 s00     e0000.366666w3826.260000
 7  #sp12 s01d02  e0000.383333w3826.280000
 8  #sp12 s01d06  e0000.400000w3826.300000
 9  #np12 s01     e0000.433333w3826.330000
10  #wp13 s00     e0000.450000w3826.360000
11  #sp13 s00d12  e0000.466666w3826.380000
12  #sp13 s00d11  e0000.483333w3826.400000
13  #sp13 s00d10  e0000.516666w3826.420000
14  #cp13 s00d02  e0000.516666w3826.430000
15  #np13 s00     e0000.550000w3826.450000
16  #wp16 s00     e0000.583333w3826.520000
17  #sp16 s00d13  e0000.600000w3826.530000
18  #np16 s00     e0000.616666w3826.550000
19  #wp17 s00     e0000.650000w3826.580000
20  #cp17 s00d13  e0000.666666w3826.600000
21  #sp17 s00d14  e0000.683333w3826.610000
22  #np17 s00     e0000.700000w3826.630000
23  #wp18 s00     e0000.716666w3826.650000
24  #cp18 s00d14  e0000.750000w3826.670000
25  #np18 s00     e0000.766666w3826.690000
26  #wp16 s00     e0000.816666w3826.750000
27  #sp16 s00d13  e0000.833333w3826.770000
28  #np16 s00     e0000.866666w3826.880000
29  #wp17 s00     e0000.883333w3826.900000
30  #cp17 s00d13  e0000.933333w3827.010000
31  #sp17 s00d14  e0000.933333w3827.030000
32  #np17 s00     e0000.966666w3827.060000
33  #wp18 s00     e0001.000000w3827.100000
34  #cp18 s00d14  e0001.016666w3827.310000
35  #np18 s00     e0001.033333w3827.350000
36  #wp16 s00     e0001.100000w3827.440000
37  #sp16 s00d13  e0001.100000w3827.460000
38  #np16 s00     e0001.133333w3827.480000
39  #wp17 s00     e0001.149999w3827.510000
40  #cp17 s00d13  e0001.166666w3827.550000
41  #sp17 s00d14  e0001.183333w3827.570000
42  #np17 s00     e0001.216666w3827.650000
43  #wp18 s00     e0001.233333w3827.680000
44  #cp18 s00d14  e0001.250000w3827.700000
45  #np18 s00     e0001.283333w3827.780000
46  #wp16 s00     e0001.316666w3827.830000
47  #sp16 s00d13  e0001.316666w3828.010000
48  #np16 s00     e0001.333333w3828.200000
49  #wp17 s00     e0001.366666w3828.230000
50  #cp17 s00d13  e0001.399999w3829.450000
51  #sp17 s00d14  e0001.433333w3829.470000
52  #np17 s00     e0001.433333w3829.520000
53  #wp18 s00     e0001.466666w3829.560000
54  #cp18 s00d14  e0001.500000w3829.580000
55  #np18 s00     e0001.516666w3829.610000
56  #wp16 s00     e0001.549999w3829.650000
```

```
 57  #sp16 s00d13 e0001.566666w3829.670000
 58  #np16 s00    e0001.566666w3829.690000
 59  #wp17 s00    e0001.616666w3829.830000
 60  #cp17 s00d13 e0001.633333w3829.850000
 61  #sp17 s00d14 e0001.666666w3829.890000
 62  #np17 s00    e0001.683333w3829.910000
 63  #wp18 s00    e0001.733333w3829.960000
 64  #cp18 s00d14 e0001.750000w3830.160000
 65  #np18 s00    e0001.783333w3830.460000
 66  #wp16 s00    e0001.816666w3830.700000
 67  #sp16 s00d13 e0001.833333w3830.710000
 68  #np16 s00    e0001.833333w3830.730000
 69  #wp17 s00    e0001.866666w3830.800000
 70  #cp17 s00d13 e0001.883333w3830.820000
 71  #sp17 s00d14 e0001.900000w3830.840000
 72  #np17 s00    e0001.916666w3830.850000
 73  #wp18 s00    e0001.950000w3831.000000
 74  #cp18 s00d14 e0001.966666w3831.020000
 75  #np18 s00    e0001.983333w3831.030000
 76  #wp16 s00    e0002.033333w3831.200000
 77  #sp16 s00d13 e0002.049999w3831.220000
 78  #np16 s00    e0002.066666w3831.240000
 79  #wp17 s00    e0002.083333w3831.260000
 80  #cp17 s00d13 e0002.116666w3831.280000
 81  #sp17 s00d14 e0002.133333w3831.320000
 82  #np17 s00    e0002.149999w3831.340000
 83  #wp18 s00    e0002.183333w3831.370000
 84  #cp18 s00d14 e0002.200000w3831.460000
 85  #np18 s00    e0002.216666w3831.500000
 86  #wp16 s00    e0002.250000w3831.530000
 87  #sp16 s00d13 e0002.266666w3831.550000
 88  #np16 s00    e0002.283333w3831.570000
 89  #wp17 s00    e0002.333333w3831.630000
 90  #cp17 s00d13 e0002.366666w3831.820000
 91  #sp17 s00d14 e0002.383333w3831.840000
 92  #np17 s00    e0002.383333w3831.850000
 93  #wp18 s00    e0002.416666w3831.880000
 94  #cp18 s00d14 e0002.433333w3832.000000
 95  #np18 s00    e0002.450000w3832.020000
 96  #wp16 s00    e0002.483333w3832.120000
 97  #sp16 s00d13 e0002.516666w3832.160000
 98  #np16 s00    e0002.533333w3832.170000
 99  #wp17 s00    e0002.550000w3832.200000
100  #cp17 s00d13 e0002.583333w3832.220000
101  #sp17 s00d14 e0002.600000w3832.240000
102  #np17 s00    e0002.616666w3832.250000
103  #wp18 s00    e0002.633333w3832.280000
104  #cp18 s00d14 e0002.650000w3832.300000
105  #np18 s00    e0002.666666w3832.330000
106  #wp16 s00    e0002.716666w3832.380000
107  #sp16 s00d13 e0002.733333w3832.390000
108  #np16 s00    e0002.750000w3832.410000
109  #wp17 s00    e0002.783333w3832.450000
110  #cp17 s00d13 e0002.800000w3832.470000
111  #sp17 s00d14 e0002.833333w3832.490000
112  #np17 s00    e0002.833333w3832.520000
```

```
113  #wp18 s00    e0002.866666w3832.550000
114  #cp18 s00d14 e0002.900000w3832.570000
115  #sp18 s00d06ae0002.900000w3832.580000
116  #cp18 s00d11 e0002.950000w3832.640000
117  #np18 s00    e0002.966666w3832.670000
118  #wp16 s00    e0003.000000w3832.700000
119  #cp16 s00d12 e0003.000000w3832.720000
120  #np16 s00    e0003.016666w3832.740000
121  #wp14 s00    e0003.066666w3832.780000
122  #cp14 s00d06ae0005.066666w3835.260000
123  #cp14 s00d10 e0005.099999w3835.360000
124  #np14 s00    e0005.116666w3835.410000
125  #wp12 s01    e0005.233333w3835.560000
126  #sp12 s01d15 e0005.266666w3835.610000
127  #np12 s01    e0005.266666w3835.640000
```