Studying Parallel Program Behavior with `Upshot`

Virginia Herrarte
Ewing Lusk

Abstract

This is a description of and a user's manual for `upshot`, an X-based graphics tool for viewing log files produced by parallel programs.

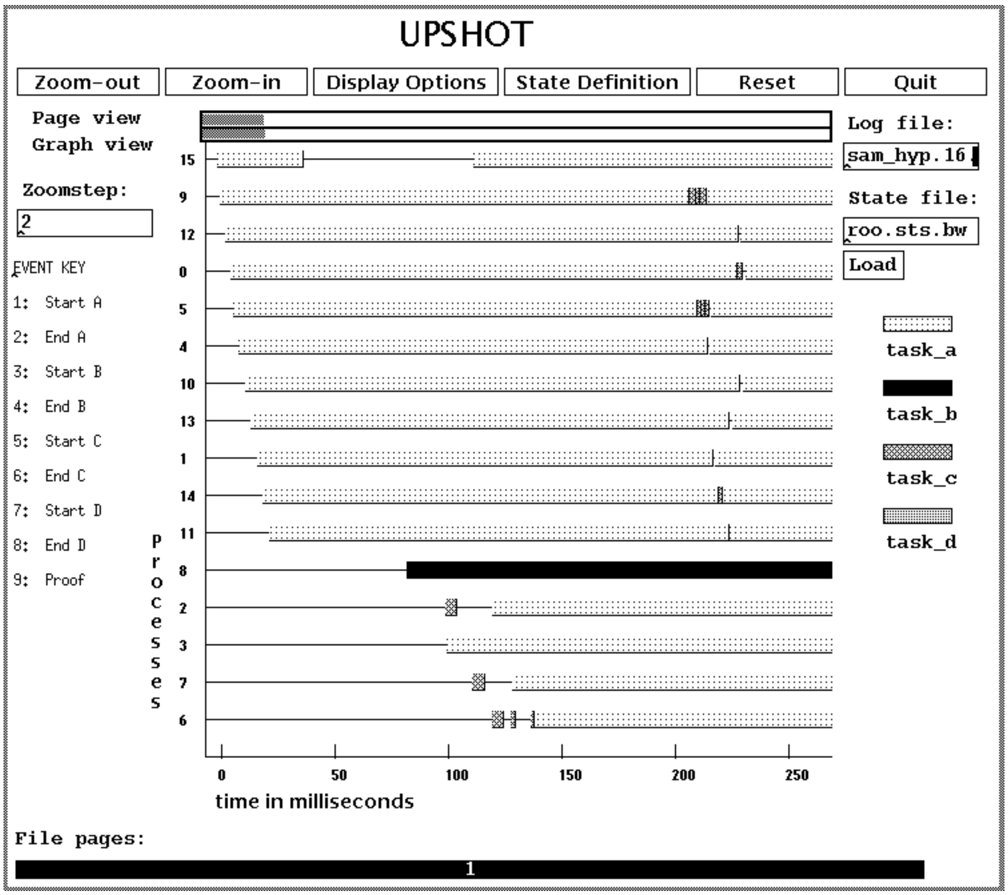

Figure 1: Upshot view of log file data

1 Visualizing Parallel Program Behavior

Whereas one can often predict the behavior of sequential programs by understanding the algorithm employed, the behavior of parallel programs is notoriously difficult to predict. Even more than sequential programs, parallel programs are subject to "performance bugs," in which the program computes the correct answer, but more slowly than anticipated.

One tool that we at Argonne National Laboratory have found useful over the years is the *logfile*, which is a list of certain types of significant events in the order in which they have occurred during the execution of a parallel program[1, 3]. Given such a logfile, regardless of how it is created during execution, we may wish to examine it in a variety of ways. `Upshot` provides one type of view of a logfile, in which events are aligned on the parallel time lines of individual processes. States of processes can be defined and displayed in terms of these events. Other useful views of logfiles, not shown by `upshot`, are animations and statistical analyses; we treat these views elsewhere.

`Upshot` was inspired by (and takes its name from) `gist`, a proprietary tool that was developed to study logfiles produced on BBN parallel computers and runs only on BBN systems. Compared to `gist`, `upshot` provides fewer features; on the other hand, `upshot` uses color more effectively, provides a smoother scrolling mechanism, is portable, and is freely available. `Upshot`, combined with a suitable logging package, provides some of the same functionality as the graphics display part of PICL[5], which displays events associated with message passing in a great variety of ways. Compared with the display facilities of PICL, `upshot` performs fewer functions but in greater depth and generality.

2 Obtaining and Installing `Upshot`

`Upshot` can be obtained by anonymous `ftp` from anagram.mcs.anl.gov (look in the `pub/p4` directory for the file `upshot.tar.Z`). `Upshot` is installed with a simple `make` in the `src` subdirectory. The `logfiles` subdirectory contains some examples of logfiles and statefiles. To obtain the display shown in Figure 1, go to the `logfiles` subdirectory and execute the command

```
../bin/upshot -l sam_hyp.16.log -s roo.sts
```

3 File Formats

In this section we describe the format of logfiles and statefiles. A logfile contains one event per line, made up of the following fields, all in character format and separated by whitespace.

Field	*Meaning*
event type	a nonnegative integer representing a user-defined event type
process id	an integer representing the process in which the event occurred
task id	an integer representing a different notion of task. This field is ignored unless the `-t` command-line option is used.
integer data	an integer representing user data for the event
clock cycle	an integer representing a timer cycle, used to distinguish between times returned by a timer that "rolls over" during the run
timestamp	an integer representing (when considered in conjunction with the cycle number) a time for the event. `Upshot` treats the units as microseconds.
string data	a character string representing user data (12 characters maximum)

In addition, `upshot` expects to find at the beginning of a logfile a number of pseudo-events, which are indicated by having negative event type. The pseudo-events and the interpretations of their fields are given in the following table:

Type	Proc.	Task	Integer Data	Cycle	Timestamp	String Data
-1						creator and date
-2			# events			
-3			# procs			
-4			# tasks			
-5			# event types			
-6					start time	
-7					end time	
-8			# timer cycles			
-9			event type			description
-10			event type			printf string
-11					rollover point	

The blank spaces in the above table are of no significance. Thus, the first few lines of the example logfile `sam_hyp.16.log` are as follows (the line numbers are not in the file):

```
 1.   -2 0 0 593 0 0
 2.   -3 0 0  16 0 0
 3.   -4 0 0 1 0 0
 4.   -5 0 0 7 0 0
 5.   -6 0 0 0 0 2025436865
 6.   -7 0 0 0 0 2028176869
 7.   -8 0 0 1 0 0
 8.   -1 0 0 0 0 2023363901  P4 May-9-90
 9.  -11 0 0 0 0 4294967295
10.   -9 0 0 1 0 2023364078  Start A
11.  -10 0 0 1 0 2023364137
```

```
12.   -9 0 0 2 0 2023364187  End A
            .
            .
            .
28.    1 15 0 1 0 2025436865
29.    1  9 0 2 0 2025438268
30.    1 12 0 3 0 2025440600
```

The meanings of these lines are as follows:

Line Number	*Meaning*
1.	This file contains 593 events.
2.	Sixteen different processes logged events.
3.	There was only one task (actually, the concept of task is not used by the logging package that produced this file).
4.	There were 7 event types.
5.	The first event occurred at time 2025436865.
6.	The last event occurred at time 2028176869.
7.	There is only one timer cycle. (The clock did not roll over during the run.)
8.	The log was created by a p4 program on May 9, 1990.
9.	The clock rollover point on the machine this was run on is 4294967295.
10.	The meaning of event type 1 is "Start A". This is used in the event type key appearing along the left side of the `upshot` display.
11.	There is no printf format string associated with event type 1.
12.	The meaning of event type 2 is "End A".
.	
.	
28.	At time 2025436865 process 15 logged an event of type 1 with integer data 1.
29.	At time 2025438268 process 9 logged an event of type 1 with integer data 2.
.	
.	

Logfiles in this format can be generated with the `alog` package, distributed separately. Versions of both Strand[4] and PCN[2] also generate such logfiles, which can thus be examined with `upshot`.

A user may define a *statefile* to accompany a set of logfiles. Such a file defines a collection of process states by identifying an entry and exit event type for each state. As an example, the file `roo.sts`, used to produce Figure 1, contains

```
1 1 2 blue    task_a
2 3 4 red     task_b
3 5 6 cyan    task_c
```

```
4 7 8 magenta task_d
```

There is one line for each state, containing a state number, entry and exit event types, a color (from the rgb.txt file of colors known to X), and a state name to be used as a key. The existence of statefiles is optional, although they are very useful. Currently, states may not overlap, nor can an event type start or end more than one state.

The above state file defines four states. The first state starts with an event of type 1 and ends with an event of type 2. It will be shown in blue on the display and labeled as "task_a" along the right edge of the `upshot` display.

`Upshot` is written using the Xt toolkit and Athena widget set, making many of its attributes user-configurable via the resource database. The `app-defaults` file supplied with `upshot` contains some choices for colors. These can easily be changed by the user.

4 Using `Upshot`

Figure 1 shows a typical `upshot` display. It was the result of issuing the command

```
../bin/upshot -l sam_hyp.16.log -s roo.sts
```

as described in Section 2. More generally, the command line operations and their meanings are as follows:

Flag	*Meaning*	*Valid Range*
-l	log file name	string up to 250 characters
-logfile	log file name	string up to 250 characters
-s	state file name	string up to 250 characters
-statefile	state file name	string up to 250 characters
-ch	canvas height	480 to 1000
-cheight	canvas height	480 to 1000
-t	graph tasks	no value necessary
-tasks	graph tasks	no value necessary

In addition, `upshot` accepts the usual Xt resource options. For example, to bring it up on a color display with black background, one might say

```
../bin/upshot -l sam_hyp.16.log -s roo.sts -bg black -fg white -bd yellow
```

The main object in the `upshot` window is the central viewport onto the canvas showing the time lines of the processes. Each line represents events logged by and states of the process whose identifier appears along the left edge. Time elapsed since the time of the first event is shown along the bottom edge, with units described based on the assumption that the units given in the timestamp field in the logfile are microseconds. The current logfile and statefile are shown at the right.

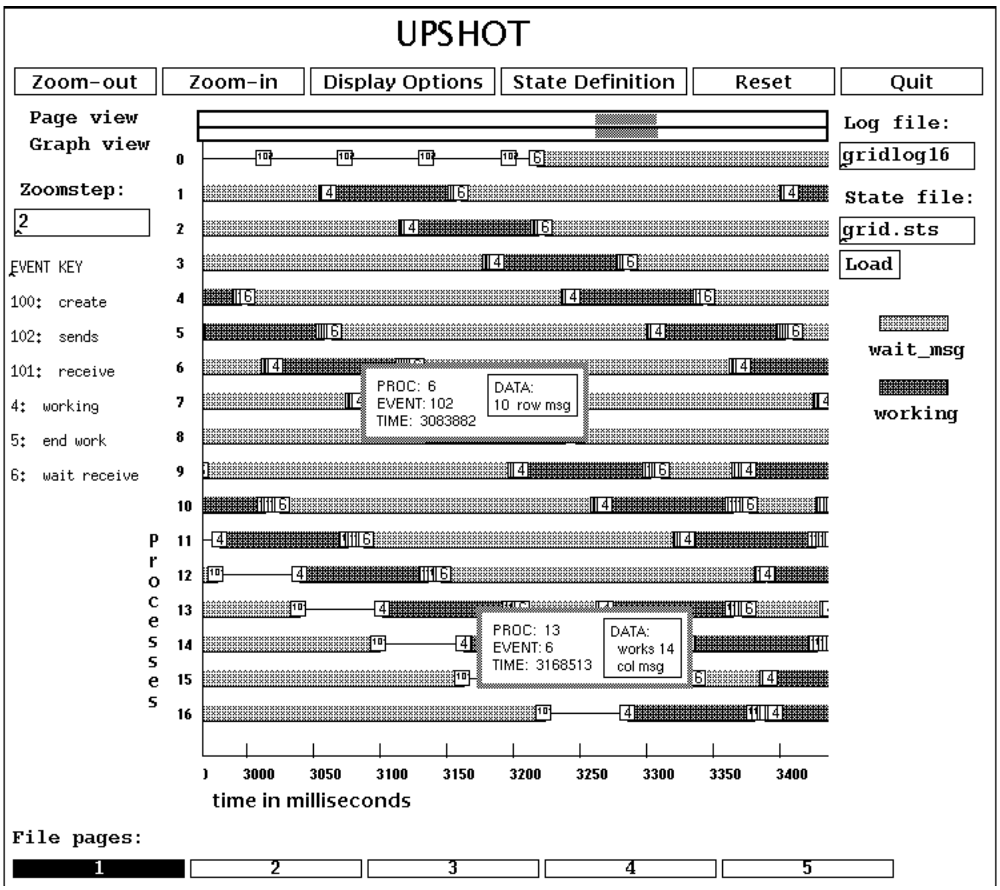

Figure 2: Upshot view with events and popup data boxes

One can scroll forward and backward in time using the two scrollbars at the top of the viewport and the buttons along the bottom edge of the window. The entire logfile is partitioned into *pages*, each consisting of a fixed number of events. Short files (like the one displayed in Figure 1) have only one page. A page is selected by clicking the left mouse button on its page number. Once a page is selected, it can be scrolled using the middle mouse button on either of the two scrollbars above the display. The page view scrollbar indicates the portion (both size and position) of the visible part of the page. The graph view scrollbar indicates the portion of the part of the canvas visible in the viewport, which is usually about ten times the width of the visible part. These two scrollbars behave the same unless the view has been zoomed in or out. Zooming changes the percentage of the page that is shown, thus changing the size of the page view scrollbar's thumb, but does not change the lower scrollbar, which remains a convenient size for scrolling back and forth in the canvas.

The buttons across the top provide further control of `upshot`'s view. The Zoom-out and Zoom-in buttons shrink or stretch the canvas along the horizontal axis, allowing detailed separation of events down to the microsecond level. When the horizontal scale changes, the left edge of the viewport remains fixed. It is possible to zoom in or out too far, in which case the initial scales can be restored by the Reset button. The zoom factor can be modified by editing the Zoomstep box. The zoom factor should always be a positive integer.

The Display Options button brings up a menu of display options. The default is to display only states. Alternatives are to display only events and to display both events and states. Events are displayed as in Figure 2. The State Definition button brings up a window for adding a new state. The Quit button exits from upshot.

Selecting a specific event with the left mouse button pops up a small window containing all of the information in the logfile for the event, including the integer and string data items. The integer data item is additionally formatted with the format string logged as event type -10. If one uses the middle or right buttons instead, the data box will remain on the screen and can be moved around. It will disappear when selected with any mouse button.

Thus the first few lines of the logfile displayed by `upshot` in Figure 2 are as follows:

```
 -2 0 0 8769 0 0
 -3 0 0 17 0 0
 -4 0 0 1 0 0
 -5 0 0 6 0 0
 -6 0 0 0 0 2781263066
 -7 0 0 0 0 2791364419
 -8 0 0 1 0 0
 -1 0 0 0 0 2779321478  May21-90
-11 0 0 0 0 4294967295
 -9 0 0 100 0 2779321670  create
-10 0 0 100 0 2779321734
 -9 0 0 102 0 2779321791  sends
-10 0 0 102 0 2779321855
 -9 0 0 101 0 2779321911  receive
-10 0 0 101 0 2779321970
```

```
 -9 0 0 4 0 2779322027  working
-10 0 0 4 0 2779322090
 -9 0 0 5 0 2779322148  end work
-10 0 0 5 0 2779322207
 -9 0 0 6 0 2779322266  wait receive
-10 0 0 6 0 2779322328  works %d
          .
          .
          .
          .
 6 15 0 0 0 2783527807  w init msg
 6 14 0 0 0 2783544696  w init msg
102 0 0 2 0 2783546175  s init msg
 6 12 0 0 0 2783561310  w init msg
 4  1 0 0 0 2783570823  working
 6  7 0 0 0 2783577774  w init msg
 6  2 0 0 0 2783594067  w init msg
101 2 0 0 0 2783608861  r init msg
 6  5 0 0 0 2783610683  w init msg
 6 13 0 0 0 2783627479  w init msg
102 0 0 3 0 2783642790  s init msg
 5  1 0 0 0 2783664655  done work
102 1 0 5 0 2783665466  row msg
```

Note that integer data and character data for an event are shown in popup data boxes, with the integer data formatted with the appropriate `printf` string.

5 Implementation

`Upshot` is an X Window application implemented in C using the Athena Widget set. The widget hierarchy is shown in Figure 3.

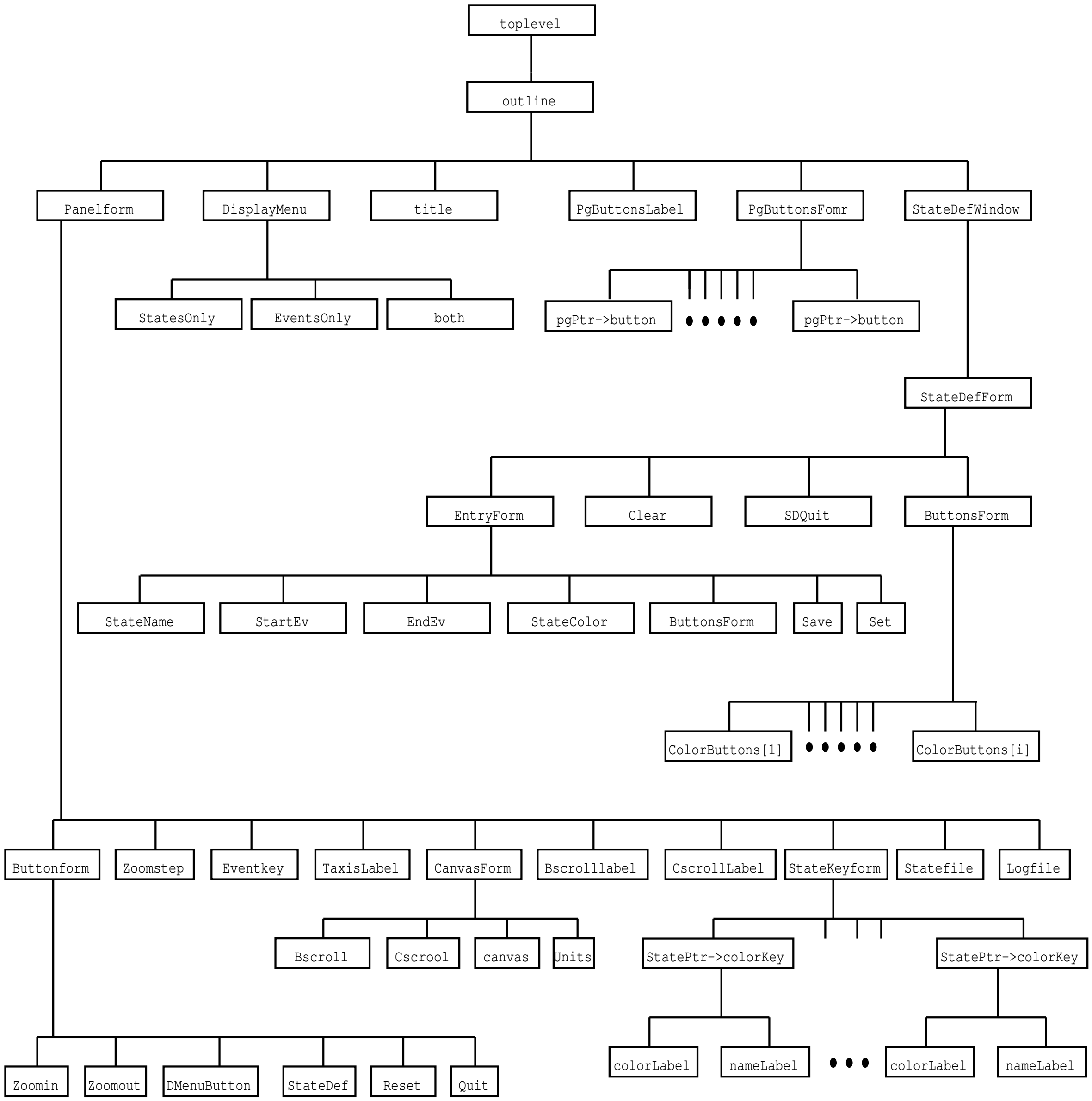

Figure 3: Widget hierarchy for `upshot`

6 Future Work

`Upshot` is one of a family of X-based graphics programs for displaying the information captured in a logfile during the execution of a parallel program. It emphasizes the static display of detailed information. Two other approaches currently under development are animation- and statistics-based.

`Upshot` has proven useful in tuning parallel programs running on a moderate number of processors. Further research is needed in order to understand how to gather and display similar information from programs running on hundreds of processors.

References

[1] James Boyle, Ralph Butler, Terrence Disz, Barnett Glickfeld, Ewing Lusk, Ross Overbeek, James Patterson, and Rick Stevens. *Portable Programs for Parallel Processors.* Holt, Rinehart, and Winston, 1987.

[2] M. Chandy and S. Taylor. *An Introduction to Parallel Programming.* Jones and Bartlett, 1991.

[3] Terrence Disz and Ewing Lusk. A graphical tool for observing the behavior of parallel logic programs. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 46–53, 1987.

[4] Ian Foster and Stephen Taylor. *Strand: New Concepts in Parallel Programming.* Prentice-Hall, 1990.

[5] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. PICL: A portable instrumented communications library. Technical Report ORNL TM-11130, Oak Ridge National Laboratory, 1990.