# Performance Instrumentation and Visualization

Edited by

Margaret Simmons
Rebecca Koskela

**ACM Press**
New York, New York

**Addison-Wesley Publishing Company**
*The Advanced Book Program*
Redwood City, California • Menlo Park, California • Reading, Massachusetts
New York • Don Mills, Ontario • Wokingham, United Kingdom • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan

# 7

# JED: Just An Event Display

Allen D. Malony[1]

## 7.1 Introduction

Event tracing has become a popular form of gathering performance data on multiprocessor computer systems. Indeed, a performance measurement facility has been developed for the Cedar multiprocessor that uses tracing as a back-end mechanism for collecting several run-time measurements including count, time, virtual memory, and event data [1,2]. Tools to study an event trace, however, are typically specialized according to the type of data collected. Usually various trace analyses and displays are developed based on some event interpretation model. Whereas this approach will give specific information about particular events and their occurrence in a trace, it is not particularly easy to extend; new events often require new analysis and display techniques.

One approach to developing a more flexible performance analysis and visualization system is that proposed in Hyperview [3]. The Hyperview architecture supports the easy addition of new event filter and display modules into the

[1]Supported in part by NSF Grants NSF MIP-8410110 and NSF DCR 84-06916, DOE Grant DOE DE-FG02-85ER25001, Air Force Office of Scientific Research Grant AFOSR-F496200, and a donation from IBM.

system and provides a patch-cord style of interconnection of filter/display combinations. This approach is the desired one for developing high-end performance environments where there are a variety of analyses and displays that must be integrated in a single system.

In the first phases of performance measurement, a user is often interested in such data as the relative sequence of events on different execution threads, the time a certain event occurs in the computation, or the state of each task as execution proceeds. High-end environments such as Hyperview, although powerful, can be overkill in situations where the user only desires to observe the sequence of events present in the trace together with information about each event's type, its time of occurrence, its place of occurrence, and any other information associated with the events as recorded in the trace. Simple analysis and presentation of individual events might be all that is required for these situations.

The goal of the project reported in this paper was to design a simple event display tool that provided basic trace management support, user-definable event specification, user-customizable graphical presentation based on a standard Gantt chart (timeline) display, and a user-extensible analysis and display architecture. The project was not without an example. In fact, the BBN GIST tool supports some of these features for the Butterfly multiprocessor [4]. We enhanced GIST's functionality in the context of the Cedar multiprocessor system to allow events to be displayed relative to "logical" tasks instead of physical processors only, to provide multiple viewports into the trace, and to run under X Windows. We also provide the user with more flexibility in display customization.

The tool we developed is named JED for (J)ust an (E)vent (D)isplay tool. It uses event traces produced by the Cedar performance measurement system referred to above. The following sections discuss the organization and operation of JED. The mechanisms to extend the standard event analysis and display features of JED are also described.

## 7.2 Target Environment

The current implementation of JED is targeted at parallel, multitask programs running on the Cedar multiprocessor system. In this environment, parallel programs use the multitasking capabilities of the Xylem operating system to partition themselves into individual tasks for execution on the Cedar clusters. A task can further take advantage of hardware concurrency support on each cluster, an Alliant FX/8, to execute loops in parallel across up to eight processors.

The Cedar performance measurement facility allows the collection of trace data from multiple program tasks. The facility is implemented as a library of counting, timing, and tracing routines that use a trace buffering run-time system for storing performance data. Currently, trace buffers are implemented in software. Every task is assigned eight trace buffers, one for each potentially concurrent execution thread, to be used for tracing during execution.

The measurement facility produces a trace file for each task at program completion that is a time-ordered merge of a task's eight processor trace streams. The file contains a header portion that gives information about the task and about the file such as the number of events generated. The format of an event appearing in a trace file is shown below:

| |
|---|
| event id |
| processor id |
| cluster state |
| event data size (*dsize*) |
| timestanp |
| *dsize* bytes of event data |

All performance measurements made using the library routines are represented as events in the trace. The event data portion is used to store information associated with a particular event. This allows measurements such as task execution times, although not specifically an event, to be recorded as such using a special event id and using the event data field to store the timing data.

## 7.3 Organization

JED is an X Windows [5] (Version X11.R3) application organized into four separate components; see Figure 7.1. The Trace Control component is responsible for reading the trace, positioning within the trace, and searching for particular events. It provides these functions for each viewport open on the trace. The Event Control component provides event definition services. It allows the user to associate names and graphic icons to events. The event to graphic icon mapping can be controlled interactively as can the visibility of events in a task trace display. The Task Display component opens viewports onto the trace and allows events for tasks assigned to the viewports to be displayed in a Gantt chart-style form. Finally, the Event Display component controls how events are shown when "clicked" on in the task display. A standard event display is provided but the user can override this default by specifying event display modules that will be dynamically linked with JED at run-time; see Event Display section.

The implementation of JED is roughly 6000 lines of C code. It uses the Xt toolkit [6], the Athena widget set [7], and the HP widget set [8]. The current version works both in black and white and in color. The dynamic linking of event display modules is currently being implemented.

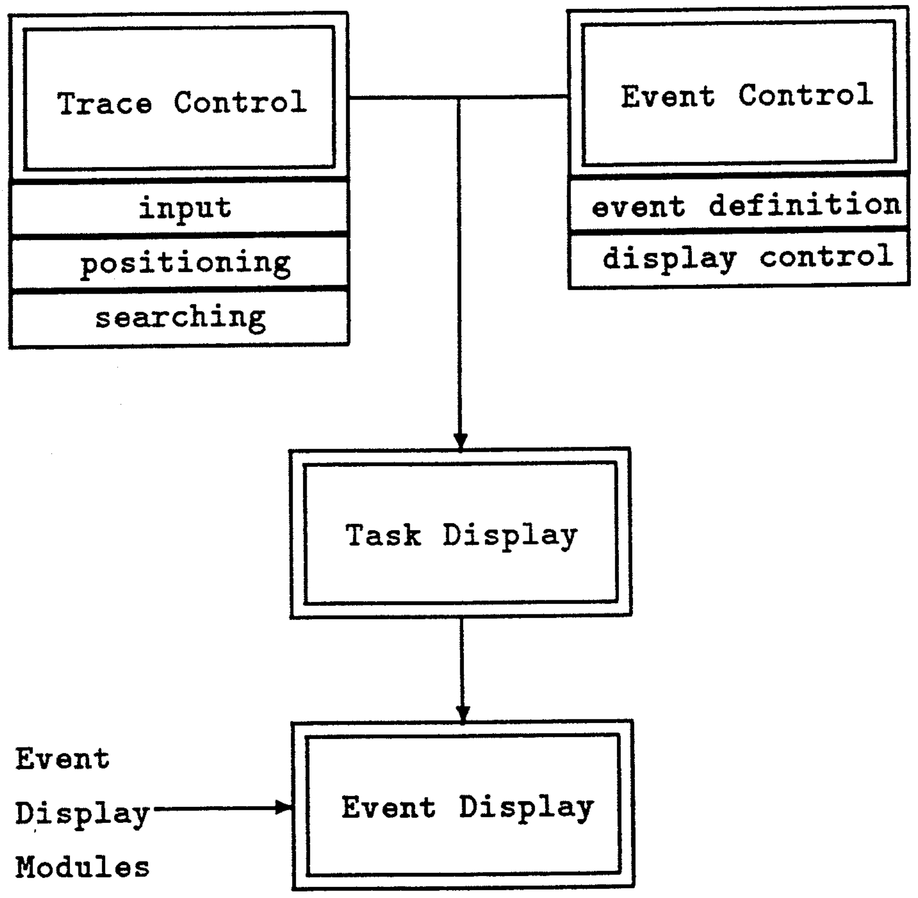
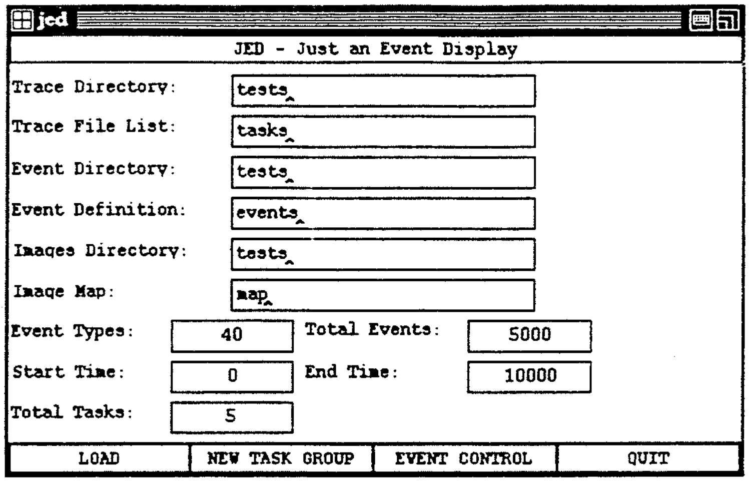


**FIGURE 7.1**
JED organization.

## 7.4 Top-Level Interface

The top-level interface to JED is shown in Figure 7.2. It allows the user to input information as to where the trace files are located and where to find information about the events that appear in the task traces. The LOAD, NEW TASK GROUP, and EVENT CONTROL command buttons provide access to the the trace control, event control, and task display components, respectively. These are discussed in the following sections. The QUIT button exits JED.



JED - Just an Event Display

|  |  |  |  |
|---|---|---|---|
| Trace Directory: | tests |  |  |
| Trace File List: | tasks |  |  |
| Event Directory: | tests |  |  |
| Event Definition: | events |  |  |
| Images Directory: | tests |  |  |
| Image Map: | map |  |  |
| Event Types: | 40 | Total Events: | 5000 |
| Start Time: | 0 | End Time: | 10000 |
| Total Tasks: | 5 |  |  |

**FIGURE 7.2**
JED top-level interface.

## 7.5 Trace Control

The trace control portion of JED implements various trace management functions. In particular, it reads task trace files according to the event trace format specification. All trace management functions operate on a per task basis. That is, trace control information is maintained for each task separately allowing operations such as trace positioning to occur on each task independently. The trace control component uses this technique to improve the efficiency of trace handling.

### 7.5.1 Trace Loading

Clicking the LOAD button in the main panel activates the trace control component. The Trace Directory input string is used as the directory path to find the Trace File List file. The *trace file list* contains a list of task trace files produced by a parallel program execution. In the example in Figure 7.2, the file *tasks* looks like:

```
task.0
task.1
task.2
task.3
task.4
```

The total number of tasks for which trace files are present is reported in the Total Tasks field.

### 7.5.2 Trace Statistics

JED opens each task trace file and reads its header. From the header, the number of events generated for this task, the time of the START_TASK_TRACE event, and the time of the END_TASK_TRACE event can be determined; START_TASK_TRACE and END_TASK_TRACE are special events inserted by the performance measurement facility . The sum of all task events is reported in the Total Events field. The earliest event time and latest event time across all tasks are shown in the Start Time and End Time fields, respectively.

### 7.5.3 Task Trace Control

As mentioned earlier, JED maintains separate trace control information for each task. It does so to increase the efficiency of updating the task displays that have different viewports opened on the trace; this will become apparent later. Since the main display items are events from a task trace, JED must be able to quickly interrogate events of a particular task. Maintaining separate task trace files and control information allows this to occur.

The C structure for task trace control information is shown below:

```
typedef struct task
int   tid;                              /* task id for this trace */
int   tracesize;                        /* task trace size        */
int   indexsize;                        /* trace index size       */
int   indexfactor;                      /* trace index factor     */
int   eventcachesize;                   /* number in event cache  */
Hrc   begin, end;                       /* beginning, ending time */
Event event;                            /* current event          */
Event eventbegin;                       /* beginning event        */
Event eventend;                         /* ending event           */
Event eventsearch;                      /* search event           */
Event eventcache[MAX_EVENT_CACHE];      /* task event cache       */
Index index[MAX_TRACE_INDEX];           /* trace index array      */
FILE  *file;                            /* task trace file        */
 Task;
```

Because trace files can be large, JED maintains an index of each task trace file for rapid event searching. The *indexfactor* gives the number of events between each event index as determine by *tracesize* and the maximum size of the index array. Since the index factor is an integer, index array sizes less than the maximum may result; *indexsize* is the actual size. JED also stores in memory the beginning and ending event for each task trace. Storage for a search event is also provided.

Although indexed trace files improves the speed of positioning within a file, we would like to avoid constantly returning to a task trace file to find events. This is especially true when studying events local to each other in a trace. The solution is to cache events for each task. If when looking for an event it is not found in the cache, it is located in the task trace file and a new block of events from that event forward *MAX_EVENT_CACHE* events is read into the event cache. It is hoped that further accessing of events local to this one will already be present in the cache and will not have to be searched for on disk.

It was found that after implementing the event caching scheme, the speed of certain operations reflecting local movement within a trace file was significantly improved. These operations include scrolling forward, scrolling backing, zooming in, and zooming out. Of course, the maximum event cache size determines the locality of event reference that can be supported. This has to be traded off against the space required to hold the events in memory which additionally depends on the number of tasks.

## 7.6 Event Control

The event control component of JED maintains information about the events appearing in the traces and how they are to be shown in the displays. A definition file is provided at start-up for labeling events and indicating how data for events should be displayed. Additionally, a user-supplied image map for showing events in task displays is used at start-up to assign default graphic event icons. Interactive control over how events are displayed is also provided.

### 7.6.1 Event Definition

The Event Directory and Event Definition input strings together indicate where the *event definition file* is found. The format on an entry in the file is:

```
event id      event name      "event data format"
```

The *event id* is the integer number of an event as it appears in the trace file. The *event name* is character string naming an event. The *event data format* is used to format data associated with an event in the default event display; see Event Display section. A few entries from the sample *events* file are shown below:

```
0     Routine_A_entry
1     Routine_B_entry
2     Routine_C_entry
6     Routine_A_exit
7     Routine_B_exit
8     Routine_C_exit
```

In this case, no event data format has been specified.

### 7.6.2 Event Image Map

The Images Directory and Image Map input strings together indicate where the *event image map* file is found. This file contains file names of bitmap images created by the X Windows application bitmap that will be assigned to events. The default event icon mapping is to assign successive images to successive events in a round-robin fashion until all events have been assigned images.

Some entries from the file *images* are shown below:

```
black
square
boxes
diamond
arrow_down
arrow_left
arrow_right
arrow_up
circle
dot
```

All the images in this example are shown in the *Image Map Window* in Figure 7.3. Naturally, the user can create her own set of images using the bitmap program. All images are assumed to be $16 \times 16$ pixels. Currently, only black and white images are accepted. We will be adding the ability to specify foreground and background colors for images in the future.

### 7.6.3 Event Control Window

The *Event Control Window* shows the user the current event display control for each event; see Figure 7.4. The name of the event is given together with the current icon assignment and a visibility control. The window is scrollable allowing the user to see all events.

Clicking on an event's icon opens the image map window. A new icon for this event can then be selected. This has the immediate effect of replacing the

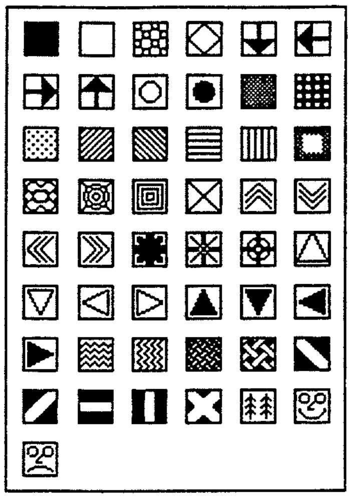
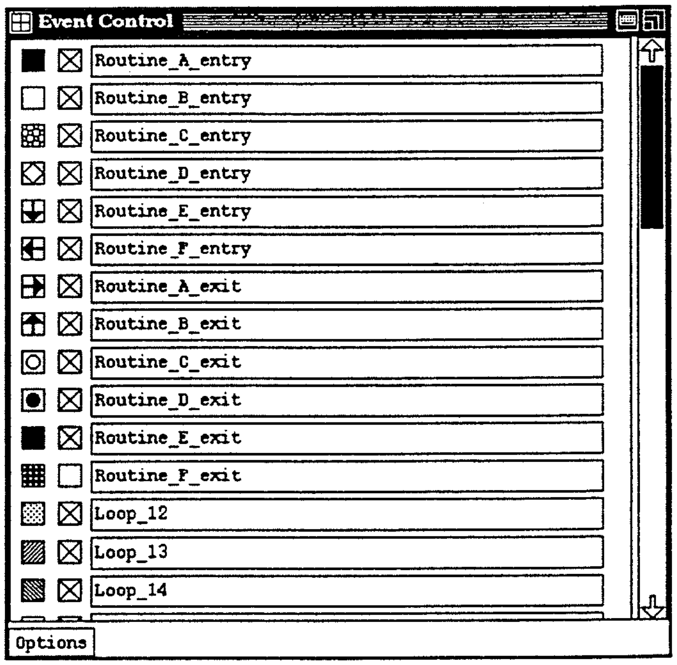


**FIGURE 7.3**
Event image map.



Event Control

| | | |
|---|---|---|
| ■ | ☒ | Routine_A_entry |
| □ | ☒ | Routine_B_entry |
| ▦ | ☒ | Routine_C_entry |
| ◇ | ☒ | Routine_D_entry |
| ⬇ | ☒ | Routine_E_entry |
| ⬅ | ☒ | Routine_F_entry |
| ➡ | ☒ | Routine_A_exit |
| ⬆ | ☒ | Routine_B_exit |
| ○ | ☒ | Routine_C_exit |
| ● | ☒ | Routine_D_exit |
| ■ | ☒ | Routine_E_exit |
| ▦ | □ | Routine_F_exit |
| ▨ | ☒ | Loop_12 |
| ▨ | ☒ | Loop_13 |
| ▧ | ☒ | Loop_14 |

**FIGURE 7.4**
Event control.

event's old icon with the newly selected on in all task displays; see Task Display section.

The visibility of an event in a task display can be controlled. An event is visible if the *visibility box* is crossed out. This is true of the Routine_A_entry event in Figure 7.4. An empty visibility box indicates the event is invisible as in the case of Routine_F_exit. Clicking on the visibility box toggles the visibility setting.

The items in the Options pop-up menu allow further event display control. They are:

| |
|---|
| All Visible |
| All Invisible |
| All Image |
| Reset |

The All Visible and All Invisible items set all the events to be visible and invisible, respectively. All Image sets all the events to the same graphic image; the image map window is popped-up and the user selects an icon. Reset returns all event image assignments to their original defaults.

Clearly, more could be added to the Options menu. In particular, more sophisticated "event coloring" choices would be useful. The idea is that events could be classified by type, such as routine entry and exit, and assigned an icon based on its type; for instance, all routine entry events might be shown as a black-filled box and all routine exits as a white box. This would enable the user to obtain a more graphically abstract view of the events in the task display. As it is currently, events are classified with respect to only one type, i.e., "an event".

## 7.7 Task Groups

To see the trace for a task in JED, the user must first open a *Task Group Window*; see Figure 7.5.

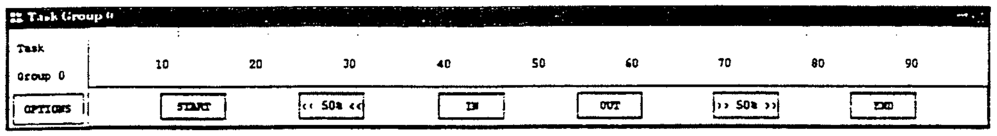


**FIGURE 7.5**
Task group.

A *task group* is essentially a viewport (time window) into the trace files for all tasks assigned to the task group. A viewport is defined by beginning and ending execution times. The task group window implements operations allowing the attributes of the time window to be changed by scrolling forwards or backwards in time, or by zooming in or zooming out in time. Multiple task groups can be created allowing task events from different times for different tasks to be viewed simultaneously.

### 7.7.1 Time Ruler

The *time ruler* shown in a task group window shows divisions of the current task group time interval in units of clock ticks. JED makes no assumptions about the resolution of a clock tick.[1] The time values shown are global times across the entire program execution. The difference between the beginning and ending times for a time window defines the current time window *resolution.*

### 7.7.2 Time Commands

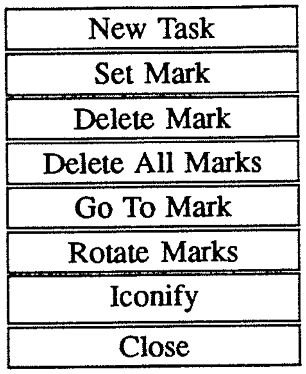
At the bottom of a task group window are commands buttons for changing the attributes of the time window. The START button sets the beginning time to be that of the first event generated across ALL task traces. The time window resolution is maintained so the ending time value is set to be the beginning time plus the resolution; for instance, if the first event occurred at time 100 and the current time window resolution is 1000, clicking START would set the beginning time to 100 and the ending time to 1100. The END button is defined similarly except the ending time is set to the time of the last event generated across ALL task traces.

The $\ll$ 50% $\ll$ button scrolls the time window to the left by 50% of the current resolution with the resolution maintained. The $\gg$ 50% $\gg$ button scrolls the time window to the right by 50% of the current resolution with the resolution maintained. The IN button zooms in on the current time window by dividing the time interval in half and maintaining the beginning time. The OUT button zooms out (grows) the current time window by twice its size while maintaining the beginning time.

### 7.7.3 Options

The Options pop-up menu provides functions for adding tasks to a task group, marking interesting points in time, and iconifying or closing the task group window. The menu items are:

[1]For the Cedar machine, a clock tick is 10 $\mu$seconds.



The New Task item lets the user assign a task to a task group. A list of tasks are shown from which the desired one is selected. A new task display is then created; see Figure 7.6 and the Task Display section. Any or all tasks can be assigned to a task group; notice, it does not make sense to assign the same task to a task group more than once.

JED lets the user to "mark" interesting points in time for a task group. The intent is to let the user return to these interesting points in the future. Setting a mark is done by selecting the Set Mark menu item and then clicking at a point on the current time ruler. When a mark is set it becomes the *current mark.* JED maintains a queue of marks for each task group. A mark icon (a triangle) is shown on the ruler at the time where the mouse click occurred; a black triangle indicates the current mark. Marks can be seen in Figure 7.6.

A variety of things can be done with marks. Delete Mark deletes the current mark. Delete All Marks does what it says for a task group. Go To Mark goes to the current mark. The time window is moved such that the beginning time is that of the mark and the resolution stays the same. Rotate Marks places the current mark at the tail of the mark queue, makes the next mark in the queue the current mark, and goes to that mark.

The Iconify menu item unmaps a task group window and shows it iconified in the top-level window. This feature is currently not implemented although it would be easy to add. Close closes the task group window and all assigned task displays.

## 7.8 Task Display

A *task display* is a region of the task group window that gets created when a task is assigned to a task group; Figure 7.6 shows a task group window with one task display and Figure 7.7 shows one with two task displays.

The graphics part of the region displays events for the particular task occurring in the time window as defined by the task group. Events are shown by their graphic icons. In the case of our Cedar implementation, task-level concur-

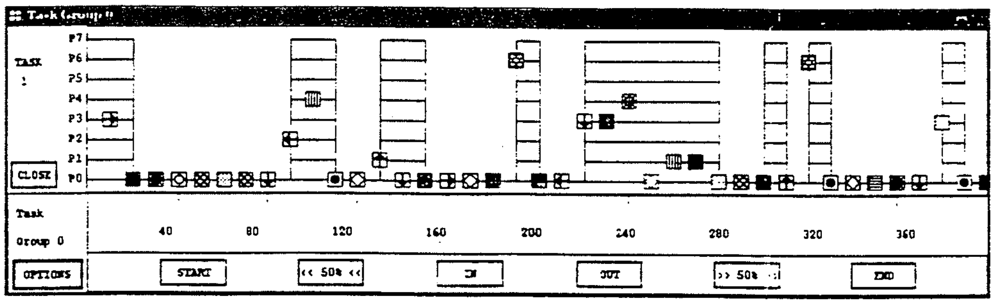


**FIGURE 7.6**
Task display—one task.

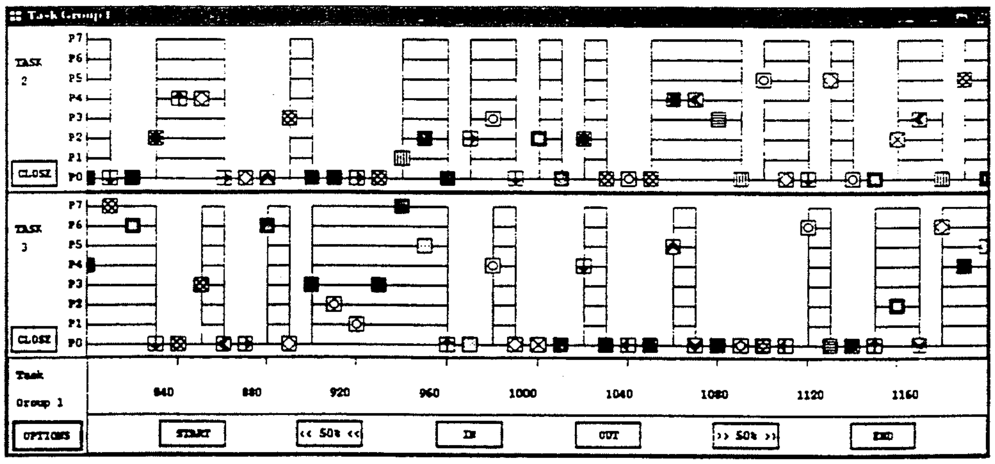


**FIGURE 7.7**
Task display—two tasks.

rency is also shown in the form of lines of sequential and concurrent activity. The CLOSE button de-assigns a task from a task group.

### 7.8.1 Gantt Chart Display

The type of display chosen to show task events is a Gantt chart. The intent was to have a simple display that would show event history. The task display does this by showing events for each task assigned to a task group occurring within the task group time window. The horizontal placement of event icons for a particular task display reflect the time relationship between events occurring for that task. The vertical stacking of task displays allows one to see the time relationship between events occurring on different tasks.

### 7.8.2 Cedar Implementation

The task display for Cedar tasks needs some explaining. Because each task executes on an Alliant FX/8, the cluster component of Cedar, it can take advantage of special concurrency hardware to have up to eight processors working concurrently. Each processor can be generating events. Therefore, there are eight possible event streams for each task.

The events for each processor are shown separately in the task display. Further, concurrency lines are shown to distinguish between periods of sequential and concurrent execution.[2] Sequential execution events are always shown on the processor 1's event line.

When interpreting the task display for Cedar programs it is important to understand that JED can only use the event information to determine sequential versus concurrent state. It cannot assume that at the time a sequential to concurrent transition occurs, as occurs in Figure 7.6 at time 220, only the processor generating the event is active. Similarly, JED must assume all processors remain active until a concurrent to sequential transition is noticed, as at time 280 in Figure 7.6.

## 7.9 Event Display

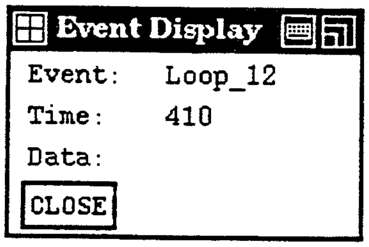
Clicking on an event icon in a task display opens an *event display window*. This provides detailed information about that particular event: the event name, the time the event occurred, and some representation of the data field, if any. JED provides a standard textual event display. Additions are being implemented that will allow a user to link to JED his own special displays for certain events.

### 7.9.1 Standard Event Display

An example of the standard event display is shown in Figure 7.8. It simply gives the event name, the time of the event, and the event data textually formatted using the format string specification from the event definition file.[3] In this example there are no data associated with the event. The CLOSE button closes the

[2]Notice, by setting all events to be invisible, only the concurrency lines will be shown allowing the user to observe sequential/concurrent transitions.

[3]At this time, textual event data formatting is not implemented.



```
Event:   Loop_12
Time:    410
Data:
```

**FIGURE 7.8**
Event display.

event display window. An event display window can also be closed by again clicking on the event icon.

The BBN GIST tool also has this capability of popping-up a textual event display window. In fact, the idea of using a format specification string to format the event data was borrowed from GIST.

### 7.9.2 Custom Event Displays

Often the event data can be represented in ways other than textually. Also, a large amount of event data can pose problems with textual event layout. For instance, an event trace may contain events recording the number of entries in a work queue. It might be desired to show these data in the form of a bargraph with the amount the bargraph is filled relative to the number of queue entries.

To provide the capability for extending the standard set of event displays, we are taking the following approach in JED; see Figure 7.9. An interface is specified in JED that allows user-defined event displays to be linked with the JED program. Essentially this interface passes an event structure from JED to a user's event display through a special *create event display* routine specific for that display. The event display interprets the event information and presents the data accordingly. JED also requires the event display modules to support a *close event display* routine to be used for closing event display windows from JED.

User-defined event display modules will be specified as part of the event definition file. For each event, there will be an indication as to what event display to use—standard or user-defined—and if user-defined, where the event display object code module resides. JED will make use of an object code instrumentation tool developed for Cedar to perform the module linking.

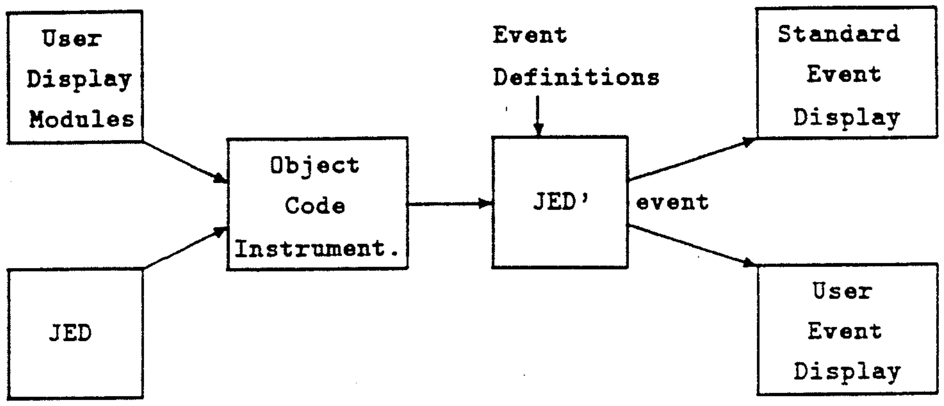


**FIGURE 7.9**
User-defined event displays.

## 7.10 Conclusion

The JED tool attempts to fill a gap between rudimentary performance reporting tools and sophisticated performance analysis and visualization systems. JED concentrates on managing and displaying event traces produced from parallel, multitasked programs running on multiprocessor systems. Currently, JED is working for traces from programs running on the Cedar machine.

Schemes have been implemented in JED to improve the efficiency of browsing multiple, potentially large, task event traces. These include per task trace control, indexing of task trace files, and event caching

Certain decisions about task trace displays and event displays have been made in JED. A Gantt chart-style display was selected because it shows event history and the time relationship between events. Events are represented in this display as graphic icons. JED support multiple viewing ports onto the traces allowing events from multiple tasks to be seen from different time periods simultaneously. Details of an event can be shown using the standard textual event display.

JED can be customized in several ways. First, there are no assumptions about events except for the event format. All event information is provided by the event definition file set up by the user. This includes the event id, event name, textual format specification, and special event displays. Second, the user can control how events are shown in the task displays using special image maps. Finally, the user can replace the standard textual event display with a custom display. For the Cedar implementation, we are building event display modules to show counting, timing, and virtual memory statistics.

## References

1. Allen D. Malony, *Program Tracing in Cedar,* CSRD Report No. 660, University of Illinois at Urbana-hampaign, April 1987.
2. K. Gallivan, W. Jalby, A. Malony, and P.-C. Yew, *Performance Analysis on the Cedar System,* CSRD Report No. 680, University of Illinois at Urbana-Champaign, June 1988.
3. A. Malony, D. Reed, R. Aydt, B. Totty, J. Arendt, and D. Grabas, *An Integrated Performance Data Collection, Analysis, and Visualization System,* 4th Conf. on Hypercubes, Concurrent Computers, and Applications, March 1988.
4. *GIST User's Manual,* Bolt, Beranek and Newman, 1988.
5. R. Scheifler and J. Gettys, *The X Window System,* ACM Trans. on Graphics, Vol. 5, No. 2, April 1986, pp. 79–109.
6. J. McCormak, P. Asente, R. Swick, *X Toolkit Intrinsics – C Language Interface,* MIT, 1988.
7. Ralph R. Swick and Terry Weissman, *X Toolkit Athena Widgets – C Language Interface,* MIT, 1988.
8. *Programming With the HP X Widgets,* Hewlett-Packard, Nov. 1988.