A Graphics Facility Useful for Performance Monitoring on an NCUBE

Dave Tolle
Shell Development Company
3737 Bellaire Blvd.
Houston, TX 77025

1 Introduction

The Computer Science Department at Shell Development Company has an NCUBE-10 hypercube computer with a fast, powerful, and complex graphics board. See Figures 1 and 2.

Part of the complexity arises from the manner in which the columns of the framebuffer RAM are interleaved among 16 processors on the board: each processor has control over one pair of adjacent pixel columns in each group of 32 contiguous columns in the framebuffer (and thus on the display screen).

Figure 3 shows the correspondence between processors and pixel columns for the first 32 columns of the screen; the pattern repeats. This arrangement makes it awkward for an application program running on the processors of the hypercube to write raster data onto the screen, since the cooperation of several processors on the graphics board is typically required for each such writing operation.

For this reason, we have implemented a simple plotting server program that resides on each of the 16 processors of the graphics board and accepts various plotting commands from processors in the hypercube. The commands can be as simple as "plot a red vertical bar graph, 20 pixels wide, 95% of height 40, at pixel location (100,200) on the screen," or "put the following block of raster data, 10 pixels wide and 60 pixels high, at location (500,300) on the screen," or as complex as "plot the following time series, consisting of 400 samples, using format number 5, in a rectangle 6 pixels wide and 500 pixels high, at pixel location (100, 200) on the screen."

The client program running on the hypercube does not have to be aware of the mapping between graphics processors and pixel columns; it can send its requests to any of the 16 processors on the graphics board. The server that receives such a request decides which pixel columns are involved in the request and forwards a set of appropriately modified requests to the processors that control those pixel columns.

We have found this graphics facility to be quite useful in visual monitoring of programs as they execute on the hypercube. In one complex application, for instance, we compute several long time series (n-component vectors) per second in the hypercube and send them to the graphics board as they are produced, while at the same time sending several bar graph requests (representing aspects of the states of the hypercube processors) to the graphics board every two or three seconds from each of the several hundred hypercube processors. All of the hard work of plotting is done on the graphics nodes, and the run-time of the hypercube program is almost exactly the same as if no plotting were asked for.

2 The Programming Interface

The portion of the plotting server that plots a time series is considered proprietary by Shell and will not be further described here.

The following message types are supported by the plotting server program:

- *panel_start*: define the window for raster graphics.
- *raster_block*: write a rectangular block of raster data.
- *node_state*: put bar graphs into a small region associated with the hypercube node.
- *node_state1*: put bar graphs at arbitrary places.
- *node_state2*: put bar graphs one above another.

We now describe each of these message types.

2.1 The *panel_start* message type

The *panel_start* message specifies the window of the framebuffer into which raster data will be downloaded by the plotting servers. Regardless of the window specified in the message, the application program is free to request bar graphs and raster blocks anywhere on the screen; only the portions overlapping the specified window will actually be downloaded from the graphics nodes to the framebuffer, though.

The use of this message type is optional; the default is for the entire screen to be used for raster graphics from the plotting servers. However, if the user's program also displays graphics through the Hitachi chips on the graphics board (see Figure 2), it is necessary to restrict the window used by the plotting server, or the pixels written by the Hitachi chips will be overwritten each time the plotting servers download their raster data to the framebuffer.

In a typical application, the *panel_start* message is the first message sent to node 0 of the graphics board. Only node 0 has to receive messages of this type, since only node 0 controls the downloading of raster data from the nodes' local memories to the framebuffer.

A *panel_start* message consists of two integers specifying the top and bottom pixel rows of the window. Because of the way the pixel columns are distributed among the 16 processors on the graphics board, the window must extend across the entire screen.

2.2 The *raster_block* message type

A *raster_block* message contains arbitrary raster data to be displayed in a rectangular region of the screen. The first four words of the message specify the leftmost column, the width, the height, and the top row for the region. The rest of the message contains the raster data, one byte per pixel.

2.3 The *node_state* message types

There are three of these: *node_state*, *node_state1*, and *node_state2*, providing three varieties of vertical bar graphs.

The *node_state* message type gives the application program on a node in the hypercube an easy way to draw a few bar graphs in a small, roughly square region of the screen that the plotting server has arbitrarily assigned to that hypercube node.

The *node_state1* message type allows the application program to specify precisely where on the screen each bar graph should be drawn.

The *node_state2* message type makes it easy for an application program on a hypercube node to draw several bar graphs, all with the same height and width, one above another at a specified place on the screen.

We have found the last of these to be quite useful, since it makes it easy to arrange for a display of several rows of information, each row consisting of bar graphs representing one run-time characteristic of the entire set of hypercube nodes.

A message of any of these types can specify from 1 to 20 bar graphs. For each bar graph, the message specifies (in one way or another) a position on the screen, a width, a maximum height, a color, and a value to be graphed. The value is an integer from 0 through 99, which is taken to be the percentage of the maximum height to be graphed. If, for instance, the maximum height is 40 and the value is 50, a bar graph 20 pixels high will be drawn.

3 Difficulties

We describe here some of the difficulties encountered in implementing the system.

3.1 The time it takes to download the plotting servers

We would like to let the plotting servers run on the nodes of the graphics board all the time, making their facilities readily available to any application program running on the hypercube. This is not easy to do, since some programs open `/dev/graphics` in order to access the functions of the Hitachi chips, and NCUBE has decided that the graphics board will be reset (thus wiping out the plotting servers running on the graphics nodes) whenever `/dev/graphics` is opened. (The `/dev/graphics` device file gives access to the graphics board through a serial communications line tied to the Intel 80186 processor on the graphics board.)

This forces us to download the plotting servers whenever an application program that uses them starts up, and this takes nearly ten seconds.

With appropriate software changes to the plotting servers and to the application programs, however, the Hitachi chips could be accessed through node 0 on the graphics board, rather than through `/dev/graphics`. This would let us avoid ever opening `/dev/graphics`, and thus the plotting servers could run continuously.

3.2 The possibility of overwhelming the plotting servers

In the current implementation of the plotting servers, no handshaking is used between the application programs running on the hypercube and the plotting servers running on the graphics board. This simplifies the interface to the facilities, but it also exposes the application program to the danger of overwhelming the plotting servers by making too many plotting requests too quickly.

It may be a good idea to re-consider this decision and require handshaking in a future implementation. (Actually, in the version of the program currently in use at Shell, we do use handshaking for time-series requests, since these use long messages and a lot of computation, and therefore carry a high risk of overwhelming the servers.)

3.3 The question of when to download to the framebuffer

The hardware design of the graphics board gives the 80186 processor the authority to decide when to trigger a download of the raster data from the local memories of the 16 graphics nodes to the framebuffer (from which the data is automatically displayed on the CRT screen). Graphics node 0 has a serial communications line to the 80186 processor, and the software design has delegated the triggering authority to graphics node 0, which in our scheme runs a copy of the plotting server.

Since application programs running on the nodes of the hypercube send plotting requests to the plotting servers asynchronously, it is not obvious how graphics node 0 should decide when to trigger the download to the framebuffer.

Should node 0 trigger downloads at some regular time interval? Should node 0 wait for another graphics node to notify it that a graphics request has been completed? Should node 0 try to assure that all graphics nodes are quiescent before triggering a download, in order to avoid downloading while some bar graph is only partially drawn?

In the current implementation, graphics node 0 triggers a download whenever it has completed a graphics request or has waited for a quarter of a second without any messages arriving. It refuses to trigger more frequently than once every quarter second. Other schemes may well be better.

4 Availability

We hope to make a public-domain version of the source files for the plotting server program available in the near future.

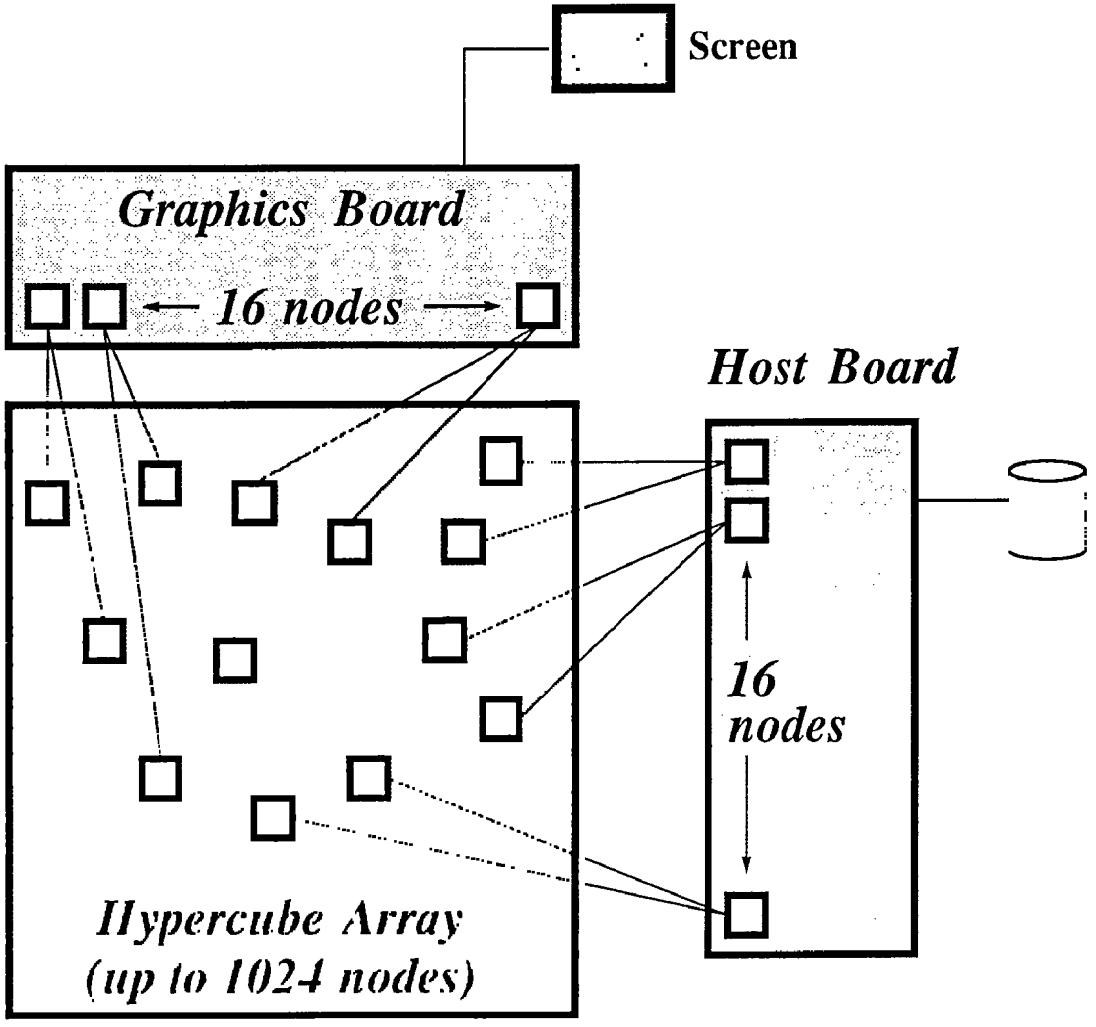

FIGURE 1 : NCUBE Overview

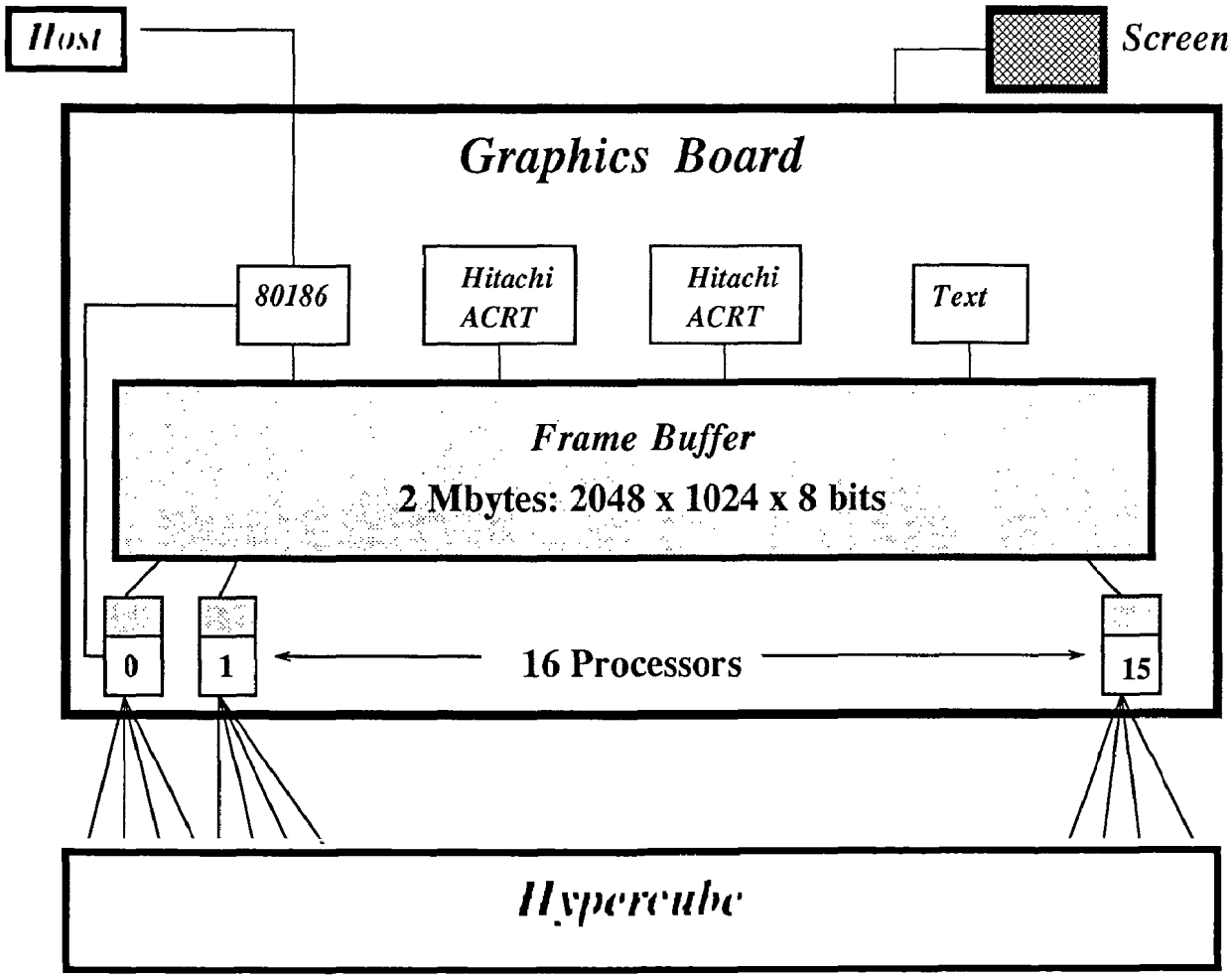

FIGURE 2: NCUBE Graphics Board

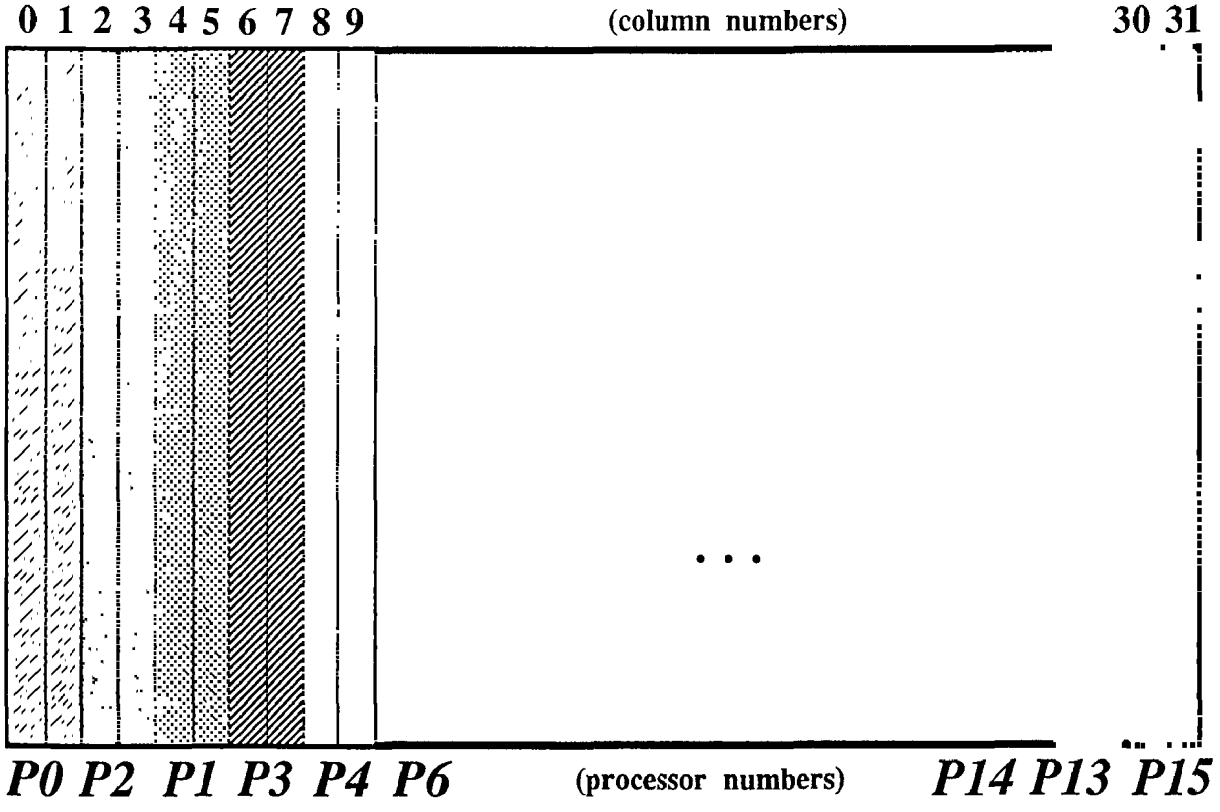

FIGURE 3 : Interleaving of First 32 Pixel Columns