Support for Collaboration, Visualization, Monitoring and Debugging of Parallel Applications using Shared Windows

Daniel Stødle[1], John Markus Bjørndalen[1], and Otto J. Anshus[1]

Dept. of Computer Science, University of Tromsø, N-9037 Tromsø, Norway
{daniels, johnm, otto}@cs.uit.no

Abstract. Data produced by individual parts of a parallel computation is typically collected into a data set or stream, and then visualized into one window. We have developed a system providing each part with one or several visualization windows. These windows can be simultaneously shared onto many displays, allowing a computation to be visualized at several locations simultaneously, or for supporting interactive, collaborative visualization. To further support collaboration, the system incorporates support for multiple cursors, allowing researchers to work simultaneously on visualization tasks on a display wall. We have used the system in a parallel implementation of Mandelbrot, as well as to share native windows from non-cluster applications on Mac OS X.

1 Introduction

Current systems for runtime visualization on clusters are not sufficiently flexible to support collaborative applications [1], and there are no out-of-the-box desktop environments for display walls that offer multiple cursors [2]. This paper presents a system for sharing windows with several users, and combines it with support for multiple cursors to enable collaboration on a wall-sized, high-resolution, tiled display ("display wall"). The system can share any window from Mac OS X with other computers running Mac OS X or Linux, and its potential use for visualization, debugging and monitoring is exemplified by sharing windows visualizing the current state of a parallel Mandelbrot computation running on a Linux cluster.

We present two scenarios to further motivate the system presented in this paper. In the first scenario, a group of researchers work on visualizing a set of results on their own computers. To share their data, they can quickly and simply share their visualization window with each other or to a display wall. The other users can interact with the window, modifying the view or change other settings as if the window was local. On the display wall, several users can interact simultaneously using multiple cursors.

The second scenario concerns runtime inspection of parallel applications. We propose using shared windows, created by each part of the parallel application, as a means of visualizing partial results, or as a monitoring or debugging tool. By subscribing to a shared window, it becomes simple to inspect the progress of a computation, or otherwise monitor the application. Since shared windows support interaction, the behaviour

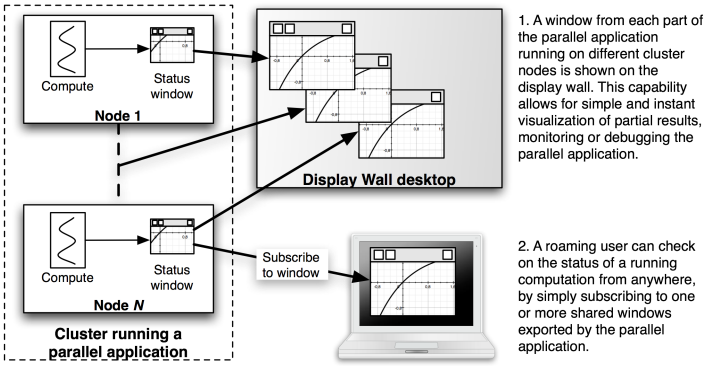

Fig. 1. Windows shared by a parallel application are accessed both for collaboration on a display wall and simple monitoring on a laptop.

of the application can even be changed at runtime. Figure 1 illustrates this scenario, in both a roaming and collaborative setting.

To meet the demands from the above two scenarios, our system should 1) support the use of tens of cursors, to accommodate tens of users simultaneously working or collaborating on a display wall, and 2) support platform independent sharing of windows, in order to facilitate sharing of windows between different window systems and hardware platforms. Our main contribution with this paper is the novel combination of window sharing with a system supporting multiple cursors, and the use of shared windows as a means for instant and simple visualization of partial results, monitoring and debugging of parallel applications running on a cluster. We also evaluate the performance of the window sharing system.

2 Design and implementation

The window sharing system consists of a server, and clients that connect to the server over TCP for publishing or subscribing to shared windows. The server supports network discovery using multicast, allowing both publishers and subscribers to discover the server on a LAN.

The current implementation lets a client on Mac OS X publish native Mac OS X windows (with no changes to application code) and subscribe to shared windows, while the Linux client only supports subscribing to shared windows. Additionally, an implementation of a parallel solver for the Mandelbrot set running on Linux was modified to share a window containing the current progress of the computation. Parallel applications wanting to leverage shared windows in this way, must be modified to export windows containing the desired information.

To achieve platform independence, windows are shared by sharing their pixel representation, as opposed to drawing operations like “fill rectangle” or “draw line.” On Mac OS X, a window’s contents are polled periodically (30 times per second), while

contents from the Mandelbrot window are sent to subscribers only when part of it is actually updated.

Support for multiple cursors was added to Window Maker[1], a window manager for the X Window System. A separate thread receives input events from the network over a number of TCP sockets, which are generated by users wanting to interact with the display wall. Multiple cursors are emulated by time-sharing the system cursor, providing a number of "virtual" cursors in its place. When a user clicks the mouse button, the system cursor is moved to the position of that user's virtual cursor, and a mouse click event is posted. The virtual cursors are distinguished by giving them different colors.

3 Experiments

We have evaluated the performance of the window sharing system by sharing a single window from a PowerMac Dual-G5 @ 2.5 GHz, 4GB of RAM running Mac OS X 10.4.2. The subscribers ran on a 28-node cluster, each node equipped with Intel P4 EMT-64 @ 3.2 GHz processor, 2GB of RAM, with hyperthreading enabled and running Rocks 3.3[2]. The interconnect was a switched, gigabit Ethernet. The shared window was sized at 508x519 pixels in 32-bit color, and contained a constantly-moving image.

The Mac OS X publisher was benchmarked with up to 28 simultaneous subscribers, with each subscriber running on different nodes in the display wall cluster. Performance seen by each subscriber decreased linearly with the number of subscribers, from 28 frames per second (fps) with one subscriber, to 3-4 fps with 28 subscribers. The publisher's bandwidth and CPU consumption increased linearly. The bandwidth usage maxed out at about 65 MB/s with 10 subscribers, while CPU usage went from 45% with one subscriber, to about 80% with 16 subscribers (max CPU load is 200%). After this, additional subscribers incurred little additional load, as the network was already saturated.

We also measured the impact of integrating shared windows with the Mandelbrot application. With one subscriber for each window shared by each part of the application, we could only measure a very small impact on the running time (less than one second). The key difference between the Mandelbrot publisher and the Mac OS X publisher, is that each part of the Mandelbrot application knows when it updates a window, whereas the Mac OS X publisher must poll for updates. Each part of the Mandelbrot computation updates the window 10 times, hence requires only 10 updates.

4 Related work

VNC [3] allows one to share an entire desktop, with many free implementations available, including some that allow sharing only regions of the desktop. Recently, SharedAppVNC [4] was made available, which integrates work derived from our window sharing prototype with VNC. Microsoft's Messenger and their older NetMeeting software support application sharing [5], but only work between computers running Windows.

[1] http://www.windowmaker.org/

[2] http://www.rocksclusters.org/

For the X Window System, there are many application-sharing solutions, with XTV [6] being one example. WinCuts [7] can support window sharing on Windows, but does not support interaction, and lacks in performance (one update per second).

Time-sharing the system cursor was introduced in [8], where a multi-cursor window manager similar to our own is presented. Their implementation is limited in that it supports at most seven cursors. Our implementation handles input focus differerently, does not impose any limitations on the number of cursors, and does not require events to pass through the X server more than once.

5 Conclusion

A system supporting shared windows and multiple cursors has been presented. It can support tens of users simultaneously collaborating on a display wall, interacting with native and shared windows. An implementation sharing windows on Mac OS X demonstrates this.

Sharing windows can be a powerful way for simple and instant runtime visualization, monitoring and debugging of parallel applications, exemplified by a parallel solver for the Mandelbrot set. Sharing a window from a parallel application for this purpose incurs very little additional load, and hardly affected the overall running time of the solver. The Mac OS X implementation fared much worse, due to its need for polling windows 30 times per second for updates.

Acknowledgements

Thanks to Vera Göbel, Espen Skjelnes Johnsen, Tore Larsen, Kai Li and Grant Wallace for their discussions, suggestions, help and support. This work has been supported by the NFR funded project No. 159936/V30, SHARE - A Distributed Shared Virtual Desktop for Simple, Scalable and Robust Resource Sharing across Computer, Storage and Display Devices.

References

1. A. Chan, W. Gropp, and E. Lusk. MPE: MPI Parallel Environment. http://www-unix.mcs.anl.gov/perfvis/software/MPE/index.htm.
2. R. E. Faith and K. E. Martin. Xdmx: Distributed, multi-head X. http://dmx.sourceforge.net/.
3. T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1), January 1998.
4. G. Wallace. SharedAppVNC. http://shared-app-vnc.sourceforge.net/.
5. Microsoft Corporation. Netmeeting. http://www.microsoft.com/windows/netmeeting/.
6. H. Abdel-Wahab and M. Feit. XTV: A Framework for Sharing X Window Clients in Remote Synchrounous Collaboration. IEEE Tricomm, April 1991.
7. D. S. Tan, B. Meyers, and M. Czerwinski. WinCuts: Manipulating arbitrary window regions for more effective use of screen space. In *CHI '04 extended abstracts on Human factors in computing systems*, pages 1525–1528, New York, NY, USA, 2004. ACM Press.
8. G. Wallace, P. Bi, K. Li, and O. Anshus. A MultiCursor X Window Manager Supporting Control Room Collaboration. Technical Report TR-707-04, Princeton University, Computer Science, July 2004.