Analysis and Debugging of PVM Programs with VISIT

Hermann Ilmberger, Sabine Thürmel, Claus-Peter Wiedemann
Siemens AG
Corporate Research & Development
D – 81730 München
Hermann.Ilmberger@zfe.siemens.de

Extended Abstract

October 5, 1994

1 Introduction

The popularity of PVM and similar parallel platforms can be further enhanced by offering adequate programming support. Our analysis and visualization environment VISIT (≫Visualize it! ≪) [IW93, ITW93] has proven to be an enormous help in the debugging and analysis phase of parallel program development. VISIT provides a detailed and high-level overview of the complex dynamic behaviour of a parallel program execution. It has been developed for parallel extensions of Common Lisp and C++ with explicit coarse grain parallelism running on a distributed memory machine. Later it was hooked to an implementation of discrete event simulation using TimeWarp. The tool consists of several visualizers related to the language's parallelism constructs, analysis tools as well as a window manager to control language related debugging tools. Besides its use for debugging, VISIT can be employed for granularity analysis and performance tuning on language level. Thus we judged it sensible to hook it to PVM, where such support was not available until recently. The flexible event-based approach and the modular and language independent design of the environment made this adaption quite straightforward.
VISIT takes a similar approach as Xab [Beg93] for event collection, but is much more visual. XPVM, a graphical console and monitor for PVM, adds a graphical user interface to the PVM console program and has a timeline display similar to VISIT's sequence chart display.

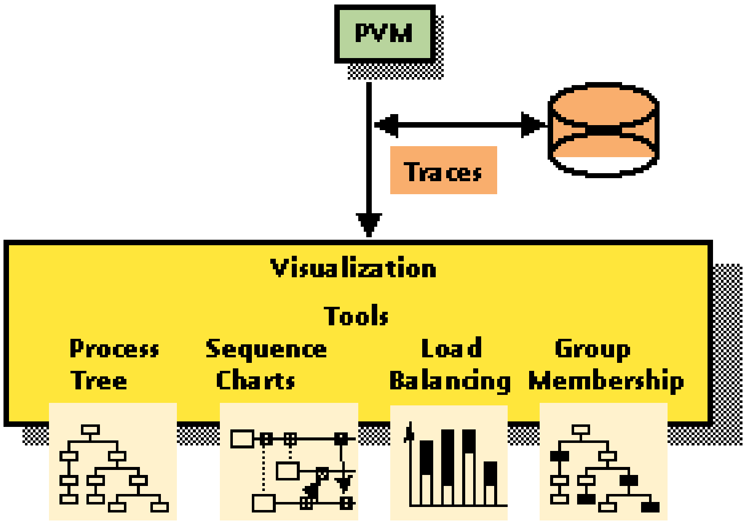

Figure 1: Trace based visualization for PVM with VISIT.

2 Architecture

The subset of the VISIT environment relevant for the visualization of PVM programs is shown in figure 1. VISIT runs on a workstation, the parallel program is executed on a multiprocessor or on a workstation network and communicates with VISIT via the network. To visualize a PVM based application program, the program only has to be recompiled and relinked. As an effect, the PVM library functions are wrapped by VISIT code which at runtime generates the corresponding trace events and send them to VISIT. The VISIT visualizers receive the events and are always informed about the current state of each PVM process.

The event messages generally include an event type, a time stamp, and event specific information. The format of the events is the same for all languages VISIT can display (EDS Lisp, PARC++, TimeWarp, PVM). We do not rely on the clocks of the workstations to be synchronized. A certain degree of time correction is done by VISIT to guarantee causality if desired by the user.

Visualization may either be online during the execution or offline (post game). For offline analysis the events are stored in a file from where they can be replayed as often as desired in arbitrary speed. This feature is specifically useful for programs which fail after a long runtime.

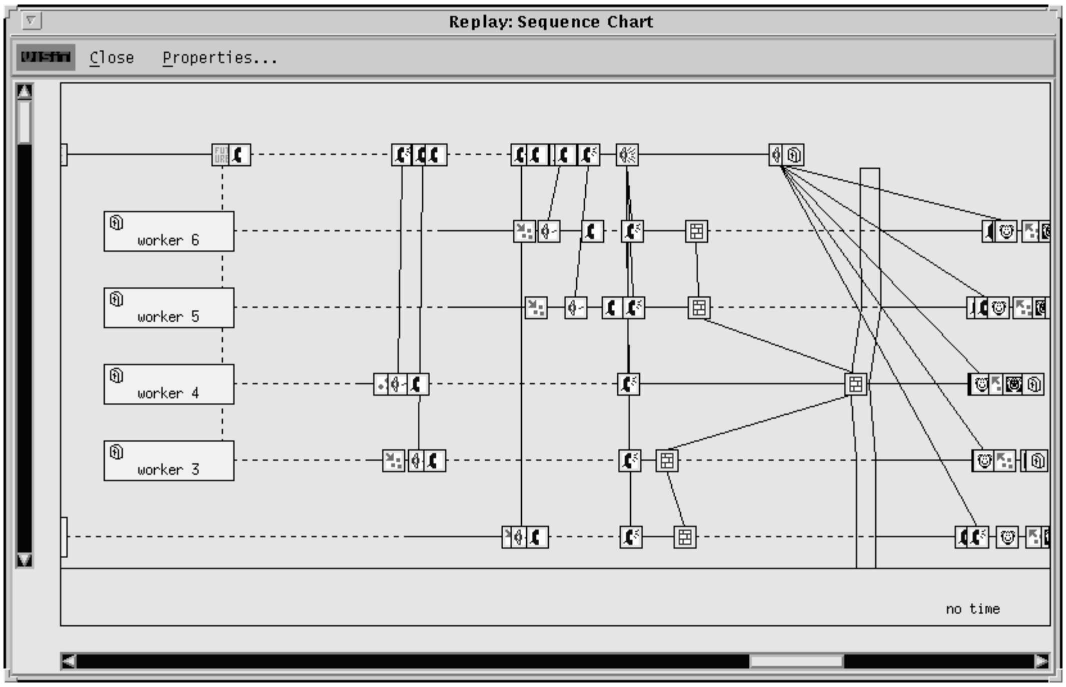

Figure 2: PVM visualization including message passing and barrier synchronization

3 Functionality

VISIT consists of several visualizers related to parallel constructs. The main component is the sequence chart visualizer (cf. figure 2). It displays the hierarchy of spawned processes and their important events as little icons along the timeline. Process-to-process communication is displayed by connecting lines. Special icons exist for multicast and broadcast. The PVM barrier mechanism is displayed by a wall icon for barrier-entry, and a wireframe for barrier-exit (see right hand side). Although the exit signal is given to all processes blocked in a barrier at the same time, this exit-frame in general is not rectangular because the time is taken by the VISIT process separately when the exit is signalled by each participating PVM process.

Other visualizations include the tree of spawned processes and a display visualizing the load on each processor. The process tree display also acts as group membership display. Members of the same group can be highlighted, and the visualizers can be restricted to processes in a certain group.

In PVM debugging mode, VISIT opens a separate window for every process debugger. In order to navigate easily between these debuggers, their windows can be controlled via the process icons in the process tree. They can be popped up and down, thus reducing the amount of windows on the screen. The question "which window belongs to which process" is no longer a problem.

4 Applications

A principal user of VISIT is our simulation group. They perform parallel circuit simulations on workstation clusters using the PVM platform [WZ94]. Major goals of their parallelization strategy are the achievement of automatic load balancing and fault tolerance. The visual proof that their implementation had the desired properties was easy with VISIT: the sequence chart display shows how workpieces are distributed, and when a workstation is switched off during a program run, a new process is automatically spawned to take over the workpieces processed by the failed process.

5 Work in Progress

VISIT is currently extended to be employed in regression testing for parallel programs. Divergences between different runs of the same program (either identical or corrected version) will be made visible. Since PVM (version 3.3 and higher) now is able to generate trace events by itself, we will look into using this trace directly for VISIT.

References

[Beg93] Adam L. Beguelin. Xab: A Tool for Monitoring PVM Programs. In *Proceedings Workshop on Heterogeneous Processing WHP'93*, pages 92–97, Los Alamitos, CA, April 1993. IEEE Computer Society Press.

[ITW93] Hermann Ilmberger, Sabine Thürmel, and Claus-Peter Wiedemann. VISIT: A Visualization and Control Environment for Parallel Program Debugging. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 199–201, San Diego, California, May 1993.

[IW93] Hermann Ilmberger and Claus-Peter Wiedemann. A Visualization and Control Environment for Parallel Program Debugging. In *26th HICSS: Hawaii International Conference on System Sciences*, volume II, pages 387–396. IEEE Computer Society Press, January 1993.

[WZ94] Utz Wever and Qinghua Zheng. Parallel Circuit Simulation on Workstation Clusters. In *8th Conference of the European Consortium for Mathematics in Industry (ECMI 94)*, University of Kaiserslautern, Germany, September 1994.