Debugging Heterogeneous Applications with Pangaea†

Leesa Hicks*
leesah@credence.com

Credence Systems Corporation
9000 SW Nimbus Ave.
Beaverton, OR 97008

Francine Berman*
berman@cs.ucsd.edu

Department of Computer Science and Engineering, 0114
University of California, San Diego
La Jolla, CA 92093

Abstract

Heterogeneous computing environments pose special challenges for debugging. They present the same difficulties as parallel computing environments, including asynchronous communication, non-determinism, and increased state information. In addition, debugging in heterogeneous environments is complicated by the distribution of work across diverse computational platforms and heterogeneous communication networks.

In this paper we describe **Pangaea**, a tool for debugging heterogeneous applications targeted to distributed resources which use PVM. Nondeterminism in the system can cause varying event orderings for multiple executions. Pangaea assists in debugging software errors by providing a mechanism to ensure the consistent replay of the program. Replay ensures that the same event ordering will be enforced for each execution. Pangaea supplies an event logging capability for support of replay as well as a post-mortem display facility. Users may also use a modified version of XPVM to display events in any mode while running an application. Pangaea and XPVM provide a useful environment for debugging many types of errors that arise when executing parallel applications targeted to distributed heterogeneous systems.

1 Introduction

Debugging any program can be difficult and error-prone. Debugging parallel programs is particularly challenging. Asynchronous execution results in program nondeterminism, making it difficult to reproduce program executions. In addition, the enormous amount of information which is potentially relevant for debugging can contribute added difficulty to the debugging of codes on parallel platforms [17] [18].

In the last decade, the increase in network speeds has made it feasible to implement parallel applications on distributed heterogeneous networks of machines (*heterogeneous applications*). Experience has shown that some applications can utilize heterogeneous platforms to demonstrate improved performance over the performance achievable at any individual site [13] [10].

The implementation of heterogeneous applications creates new challenges for debugging. A debugger for these applications must provide information about an application's execution history both in terms of its usage of the distributed resources and its performance at individual execution sites in the system. The problems which arise in debugging applications on parallel machines must also be addressed in the heterogeneous environment, although the solutions must focus on the diverse administrative domains and communication paradigms of the distributed system.

One of the greatest sources of errors for both parallel and heterogeneous applications is communication event ordering. To illustrate, two tasks A and B may send messages to another task C. In one execution, a message from A might arrive first, whereas in another execution a message from B might arrive first. Task C might behave differently, and perhaps unintentionally, depending on which message arrives first. For example, task C might produce a segmentation fault when it receives task B's message first, but runs correctly when it receives task A's message first. Reproducing an error triggered by such a message ordering can be troublesome. In fact, stopping the execution of task B by setting a breakpoint in a debugger may slow it down sufficiently to allow task A's message to always arrive first.

Reproducibility is important in detecting data races, send and receive mismatches, buffer overflow, etc. Facilities are needed to capture the event ordering for erroneous behavior and to reproduce it as many times as necessary to diagnose the error. This reproduction of event ordering is known as *replay*. Although there have been several tools developed which provide replay for parallel applications on individual multiprocessors, there are few tools available which provide replay for parallel computations targeted to heterogenous platforms.

To address the need for a debugger that provides a replay capability for heterogeneous message passing systems, we have developed **Pangaea**. Pangaea targets networked resources which utilize PVM[1] [3], and combines the visual trace information provided by the XPVM tool [19] with a new facility for program execution replay. Pangaea's replay facility can be coupled with PVM's ability to access local debuggers at each execution site. In addition, a new Pangaea event graph view has been added to XPVM to support visualization of communication events. In this paper, we describe the Pangaea system design and implementation, and demonstrate an example execution.

2 Related Work

A number of tools have been developed for debugging parallel applications. In this section, we focus on those tools which target

*Supported in part by NSF contract number ASC-9301788.

†Available via WWW URL http://www-cse.ucsd.edu:/users/lhicks/pangaea.html or via anonymous ftp to cs.ucsd.edu:/pub/lhicks

SPDT '96, Philadelphia PA,USA

1. PVM (Parallel Virtual Machine) is a communication interface for heterogeneous networked resources. MPI [14] was also considered, but at the time of this research, did not provide task initiation capabilities.

parallel programs on distributed resources. There are a number of tools for debugging heterogeneous applications which do not provide replay. There are a few tools that provide some type of replay facility, but these have generally been limited in scope.

2.1 Tools Without Replay

The P2D2 debugger interface [5] supplies a centralized interface to remote, native debuggers for applications using PVM and MPI. P2D2 utilizes Application Interfaces (API) supplied with commercial debuggers to provide a consistent debugger interface for each platform that it supports. This approach eliminates the need to learn differing debugger interfaces, but relies upon the debugger vendors to provide APIs in addition to the user-interfaces.

The commercial ConvexPVM system from CXSOFT [7] includes a debugging tool, PVMdb, which also manages multiple debugging sessions for heterogeneous applications using PVM on HP 9000/700 systems[1].

Xab [2] supported run-time and post-mortem display of individual PVM task events. PVM routines were instrumented to send event information to the Xab monitor. The Xab display shows the most recent PVM event that each task performed. When performing post-mortem display of events (playback), the Xab display provides control mechanisms for stopping playback, resuming playback, single stepping, playback speed, and exiting the program.

The Xab system was replaced by XPVM along with instrumentation implemented directly into PVM. XPVM furnishes a graphical interface and monitors events for heterogeneous applications using PVM. XPVM supports several views: a network view, a space-time view, a utilization view, a call trace view, and a task output view. PVM trace data is collected into a trace file and is used to build and update the views. Information can be displayed real-time or as a post-mortem playback of the trace file. XPVM by itself has no replay facility, but as we describe herein, XPVM can be combined with Pangaea to provide replay facilities.

2.2 Tools Supporting Replay

Xmdb [8] provides a very limited set of replay facilities. It also provides execution control for a single process running on a modified version of PVM to aid in program development. Xmdb is intended to be a training tool for inexperienced developers, and may be useful for initial algorithm testing by more experienced developers. With Xmdb, the user specifies when messages are delivered to the controlled process. In addition, the event ordering for the controlled process can be replayed as an aid to debugging errors caused by differing event ordering between executions. Xmdb currently runs all processes on a single platform, limiting the number of tasks that can be practically run. The application source code may also require modifications to conform to Xmdb's message packing restrictions and to work around unsupported PVM functions.

Two other systems provide better replay facilities, but have other limitations. The Agora debugger [9] provides event logging and replay facilities for a particular heterogeneous system centered around a shared address space model. The Prospero Resource Manager (PRM) [16] provides replay with its debugging capabilities, but requires use of the Prospero operating system. This system is supported on Sun SPARCstations, Sun-3, and HP9000/700 platforms. Most heterogeneous systems, however, are typically based on PVM or MPI, support message passing communication, and span a wide variety of execution sites.

1. ConvexPVM is being discontinued after March 1996.

Unlike existing tools supporting replay, Pangaea supports a wide variety of applications and execution sites. There is no restriction to any particular heterogeneous system or operating system as with Agora and PRM. Unlike Xmdb, Pangaea runs in a production PVM system rather than a test environment. Pangaea supports replay for all of the application tasks rather than a single task. Tasks run where the user specifies them in place of a single platform; thus there is no limitation on the number of tasks that can be run. Also, Pangaea has no message packing and unpacking restrictions, and all relevant PVM functions are supported by Pangaea.

3 Overview

Heterogeneous applications are typically composed of a few coarse-grained components targeted to different machines in the system. Execution of each of these computations is asynchronous, and computation and communication can be overlapped to amortize network latency [1]. Debugging such applications requires a mechanism for debugging each of the components at each of the execution sites in the system (i.e. local debuggers) as well as a mechanism for tracing and displaying communication between components on the network. Replay of the application is especially important due to the non-determinism inherent in network communication.

Pangaea is targeted to support applications which use PVM. An application running in Pangaea must first be runnable from XPVM. This prohibits the application from using standard input (*stdin*) to obtain data, since all tasks spawned by PVM have */dev/null* opened for *stdin*. Programs are instrumented, requiring that each task 1) include a Pangaea source file header, 2) invoke a Pangaea initialization function prior to invoking any PVM functions, 3) invoke a Pangaea exit function just prior to exiting PVM, and 4) be bound with the Pangaea library. Pangaea supports all C language functions supported by PVM, although Pangaea requires that PVM buffer initialization be performed prior to every message sent. Replay support in Pangaea requires that the application send and receive the same messages given the same inputs.

The instrumented applications can be initiated by a modified version of XPVM or by Pangaea without XPVM. XPVM is used to display information in the event graph real-time in *record* or *replay* mode, and post-mortem in *display* mode. If the application is run by the Pangaea modified version of XPVM, then the developer selects and enters, as with unmodified XPVM, the same information for starting a task from the XPVM Spawn Dialog menu. In addition, the developer selects the desired Pangaea mode (record, replay or display, defined below). XPVM automatically initiates the application as needed to run the selected Pangaea option. Pangaea automatically detects that XPVM is running and supplies the event visualization information to XPVM.

If the application is run with Pangaea options from a shell without using XPVM, then the overhead of communicating and displaying events is avoided. In this case, the developer runs the Pangaea top-level task rather than the application, and supplies information required for executing the application on the command line. The record and replay modes are available in this type of execution and the visualization capability is suppressed.

The record/replay facility of Pangaea is based on the strategy used by LeBlanc and Mellor-Crummey in Instant Replay [11]. When sending messages, a header is inserted into the message that contains enough information to uniquely identify it. The message header is extracted when it is received to record or replay the event. Within Pangaea, messages are identified using Pangaea logical task ids and message identifiers to enable matching of events to tasks in subsequent replay executions. In the record mode, the ordering of the task spawn and communication events of a hetero-

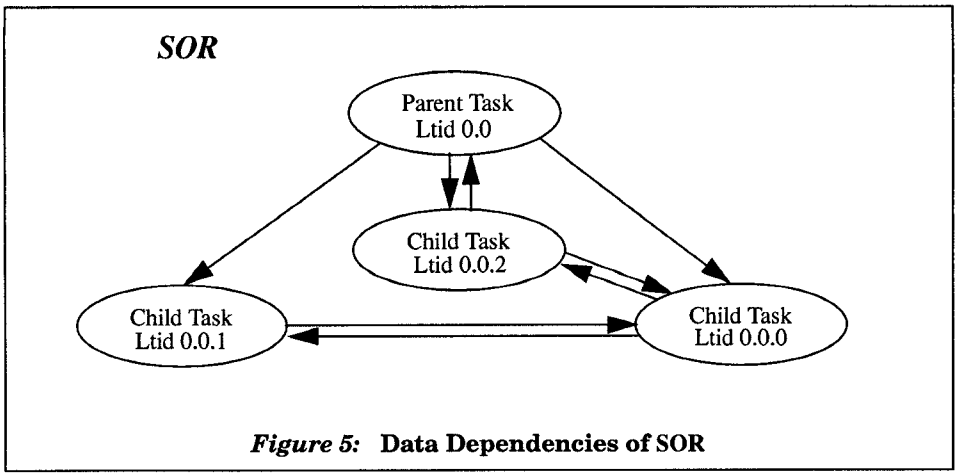

Figure 5: **Data Dependencies of SOR**

geneous application are captured. In the replay mode, the event ordering from the previously recorded session is displayed while the application is re-executed using the captured state information. The replayed session can be repeated as often as necessary to detect the cause of failure. In addition, a functional interface allows application tasks to suspend and resume communication logging and replay activities ("selective logging and replay") for the purpose of minimizing the size of log files and reducing overhead. This approach is based on [15] and [6] where they were used successfully in the parallel setting. Selective logging and replay are especially useful when the developer or user suspects that a bug is related to a particular message or message type, and recording a subset of events is adequate for reproducing an error.

The Pangaea system provides a graphical visualization of PVM communication events as they are being recorded or replayed. A new Pangaea event graph view has been added to XPVM to display send and receive events and their relationships to one another. All other existing XPVM displays are also available. The Pangaea event graph view complements the XPVM space-time view. The event graph view supports the record, replay, and post-mortem display features of Pangaea. It was implemented as a separate display for several reasons. First, tasks in the event graph are identified by Pangaea logical task id instead of by PVM task id as in the space-time view. Second, communication events in the event graph are represented with different shapes for different types of events, whereas the space-time view makes no such distinction. Third, the space-time view can be active at the same time as the event graph view. Finally, separating these two views retains all the existing features of XPVM while simultaneously supporting the new Pangaea features. The following section illustrates this view and describes how Pangaea works.

4 Example Execution

An application that solves Laplace equations using a red-black Successive Over-Relaxation (SOR) algorithm is used to demonstrate Pangaea. This algorithm splits a matrix into strips, with one strip per task. Each task executes on its strip, shares the proper rows with its neighbors, and then communicates the maximum result to all other tasks after each iteration. The SOR application was run with Pangaea using four tasks with a problem size of 100, and was executed on a heterogeneous network of Sun and RS6000 workstations. The Pangaea top-level task, *pangaea*, was run on the user's Sun workstation. The data dependencies of the SOR application tasks are shown in Figure 1. (Since there are no data dependencies between *pangaea* and the SOR tasks, *pangaea* is not shown in Figure 1.)

During execution, SOR is first run in record mode (Log button is selected), and Pangaea uses XPVM for visualization (see Figure 2). With XPVM, the spawn parameters are entered the same way whether Pangaea is to be run or not. The user enters the application invocation in the command entry, sets the PVM flags, and then specifies both where to run the application and how many tasks to run. The Pangaea top-level task spawns the parent task of SOR as specified in the spawn parameters, and the SOR parent task spawns three child tasks. The parent and child tasks each process a strip of the matrix. In addition, the parent task is responsible for broadcasting data required by all tasks. If the user needs to start up a task in a local debugger, the PvmTaskDebug flag can be selected with the appropriate debugger initialization mechanism [3].

Pressing the Pangaea Options button extends the Spawn Dialog with the Pangaea selections, as shown in Figure 2. Selecting one of the Replay, Log, or Display mode buttons activates Pangaea and directs XPVM to execute the Pangaea top-level task in place of executing the application directly. The host on which the Pangaea top-level task is to be run can be specified, as can the host on which the Pangaea task coordinator is to be run. (These components of Pangaea are described further in Section 5). If the *None* mode button is selected, Pangaea is not invoked for the application.

In the XPVM example invocation shown in Figure 2, the Pangaea top-level task (*pangaea*) is initiated on *seuss.ucsd.edu*. This in turn spawns the Pangaea task coordinator on *trini.ucsd.edu* and SOR as specified by the Spawn Dialog options. Two log files are created for each SOR task, one for task spawns and one for communication events. (Placing task spawns into a separate log file allows the user to modify where tasks run in a replay execution, as discussed in Section 6.2.) In addition, Pangaea notifies XPVM of each relevant PVM event occurrence for posting to the Pangaea event graph view, which is visible when the view is activated from the Views menu.

Figure 3 shows the first few communication events in the Pangaea event graph that occurred between the tasks of SOR while logging events with Pangaea. (It is infeasible to show all of the communication events in this static format.) Each task and its events are displayed in a horizontal line in chronological order. Task identification includes the host name, task name, and its Pangaea logical task id.

Circles (red in a color display) in the Pangaea event graph represent send events, triangles (blue in a color display) represent receive events, and squares represent broadcast and multicast

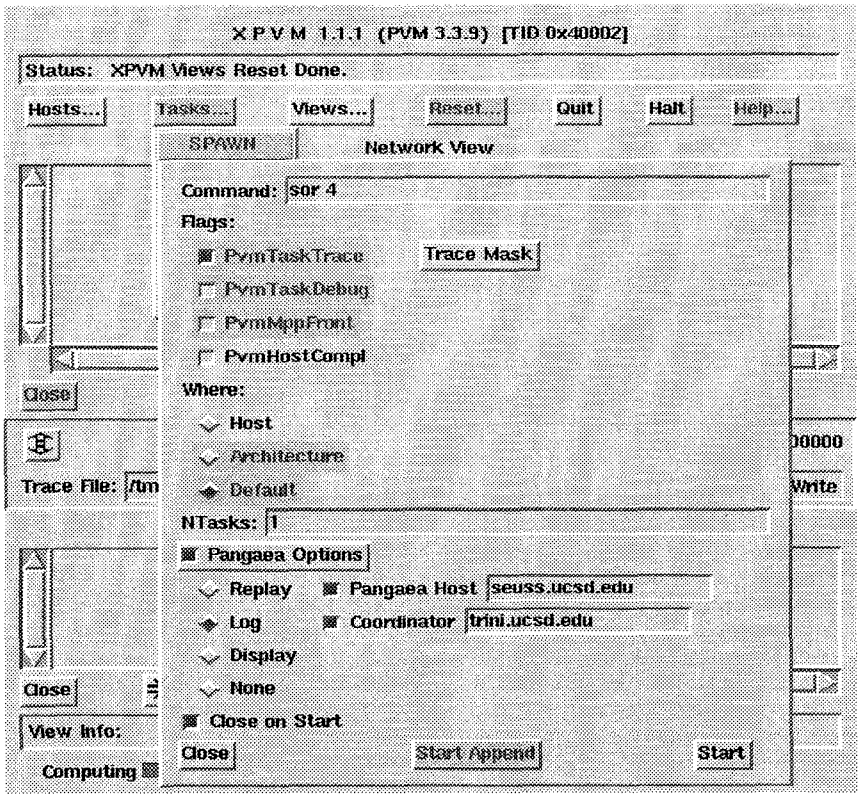

Figure 2: **XPVM Spawn Dialog**

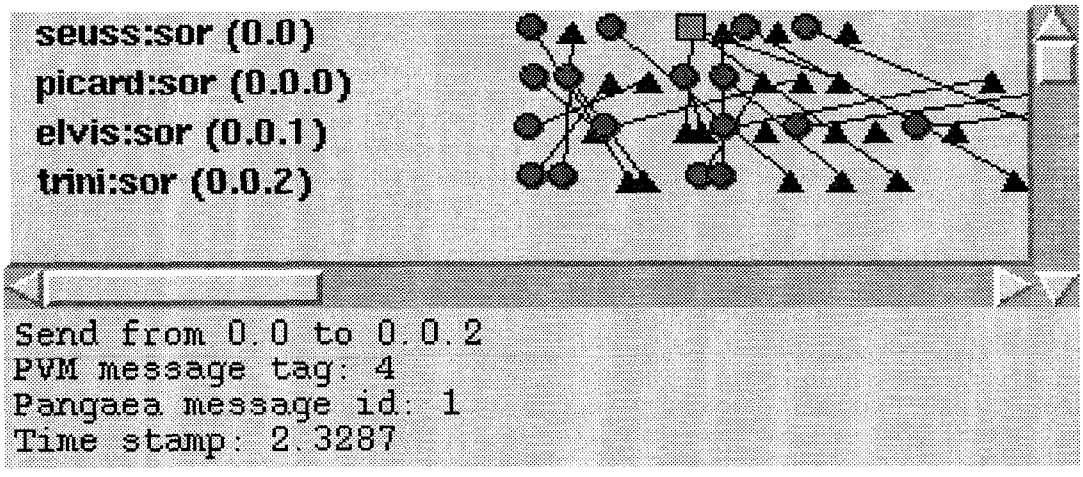

Figure 3: **Pangaea *Event Graph* View**

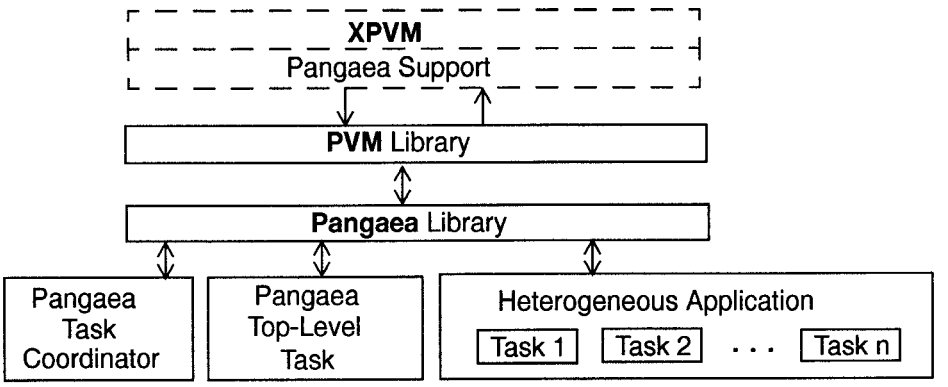

Figure 4: **Pangaea Composition**

events (green for broadcasts and yellow for multicasts in a color display). Lines drawn between corresponding sends and receives show communication. With broadcasts and multicasts, there may be multiple lines connecting different receive events. Information for an event or communication is displayed below the view when the mouse button is pressed over a send, receive, broadcast, multicast, or communication. In Figure 3, a send event has been selected. The detailed information for this event is displayed below the horizontal scroll bar, and includes the sending *ltid*, the receiving *ltid*, the PVM message tag, the Pangaea message id, and the time-stamp.

The scaling of events can be modified in the event graph to increase or decrease the spacing between events. Pressing mouse button 2 increases the spacing between events by decreasing the time scale, and pressing mouse button 3 decreases the spacing by increasing the time scale. This capability allows users to adjust the relative positions of events for better readability or to explore communication patterns.

To reproduce the recorded execution ordering, the application can be initiated through XPVM by selecting the Replay mode button instead of the Log mode button in the Spawn Dialog box or by substituting *-replay* for *-log* in the shell invocation. In replay mode, the application will re-execute on the heterogeneous system in such a way that PVM events occur in the same order as they were recorded in the log files. In particular, event ordering is enforced by matching a current event to a pre-recorded event. If an irreconcilable difference occurs, such as a task sending a message to a destination different from that recorded in the log file, an error message is reported.

The same process is used to run in display mode, although this option is only available while running from XPVM. In this case, events will be posted to the Pangaea event graph directly and only from the log files, bypassing the application code. The display will look identical to the one created in record mode since all data, including the time stamps, is extracted from the log files.

In the following, we provide a more detailed look at the structure and functionality of Pangaea.

5 Pangaea's Design and Structure

Pangaea's design supports record, replay, and post-mortem display of communication events for distributed applications using PVM for task initiation and communication. The initial implementation supports C programs using PVM. XPVM provides the graphical display interface for Pangaea. Pangaea is comprised of a new C code library that supports the record, replay, and display capabilities for PVM events, a program to run as the top-level task, a program to coordinate task identification, and new C and Tcl/Tk code to implement the graphical display capabilities in XPVM. Figure 4 shows the structure of Pangaea. The following sub-sections describe selected components of Pangaea.

5.1 Pangaea Top-Level Task

Replay of communication events relies on unique, deterministic task identification, both to identify events and to name log files. PVM assigns unique task identifiers (*tid*), but they are not deterministically generated; PVM task identifiers can and do vary for the same task from execution to execution. Assume, for example, that a receive event is to be logged for Task A (PVM *tid* 40025) that was sent from Task B (PVM *tid* c0008). The log entry needs to include the sender's task id for unique identification of the receive. Yet with PVM, the task identifiers for Task A and Task B can differ from one execution to the next. So while the logged entry can record the sender as being Task B's *tid* c0008, a subsequent execution of task B may assign a new PVM task id (say c0012, for example), and there is no means of identifying the logged *tid* c0008 as the current *tid* c0012, both of which represent Task B.

Reproduction of event ordering from a previously recorded session relies on deterministic identification of tasks; consequently, Pangaea assigns each task a unique, deterministic logical task id (*ltid*). This is accomplished by using a top-level task to act as a single ancestor to all of the tasks of an application in order to generate logical task identifiers. *Ltid* assignment requires that the application's tasks must all be spawned either from the Pangaea top-level task or by the application's tasks.

In contrast, tasks of a normal PVM application aren't required to be related in order to communicate. As an example, tasks A and B can each be executed from a shell, join a group, obtain each other's *tid*, and then proceed to send messages to each other. It does not matter whether task A or task B is executed first when using PVM, because there is no requirement that PVM task ids be assigned deterministically. It does matter to Pangaea, however, since tasks must be consistently identified between different executions. Requiring that all tasks be related imposes a deterministic execution ordering on each task, thereby enabling deterministic assignment of logical task identifiers. Users may comply with this relational requirement by modifying the Pangaea top-level source code to spawn the tasks of the application that were previously run from shells. In particular, tasks A and B could be spawned by the

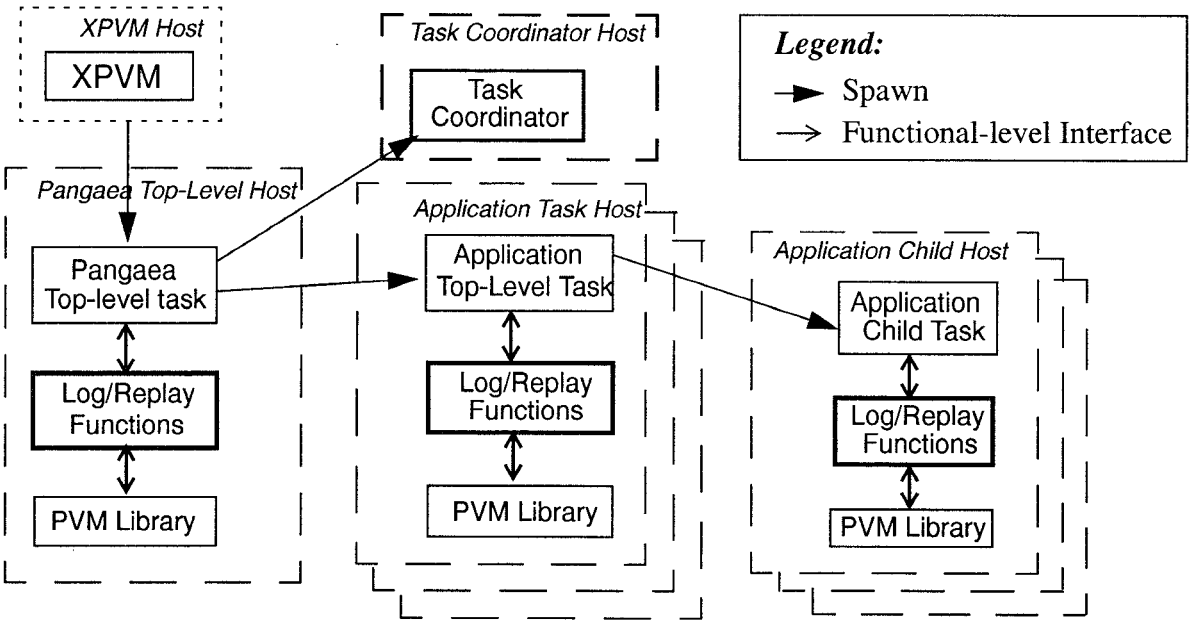

Figure 5: **Pangaea Run-Time Hierarchy**

Pangaea top-level task in the appropriate order. By adding these task initiations to the top-level task, the user can meet the task relationship requirements for Pangaea without modifying the application.

The Pangaea top-level task is a C program that initializes the Pangaea support routines and spawns the application's top-level task(s) as specified by the command line arguments. When running Pangaea options with the XPVM interface, new code in XPVM executes the Pangaea top-level task automatically, although the developer may specify the host on which this task runs (the default is to run on the same host as XPVM). When running directly from a shell, the developer must invoke the Pangaea top-level task instead of the application's top-level task and supply the appropriate command line arguments specifying how to spawn the application.

5.2 Pangaea Task Coordinator

The Pangaea task coordinator provides translation facilities between PVM task ids and Pangaea logical task ids. The Pangaea top-level task spawns the task coordinator during its initialization. Each task registers its PVM task id and Pangaea logical task id with the task coordinator during Pangaea initialization. The task coordinator collects this information to process translation requests.

When first sending a message to a particular task (destination), the sending task knows the destination's PVM task id (*tid*), but it may not know the corresponding logical task id (*ltid*). (An *ltid* is known once the first translation request for the corresponding *tid* has been processed). The sending task must have the destination's logical task id for recording or replaying the event. If the *ltid* is not already known, then the instrumented PVM send functions in the Pangaea library (*pvm_send, pvm_bcast, pvm_mcast*) initiate a translation request for a PVM task id from the task coordinator to satisfy this requirement.

When requesting a *tid* for a group member with *pvm_gettid*, the group member's *ltid* is required for recording the event. If the *ltid* is not already known, then the Pangaea library function for *pvm_gettid* initiates a translation request for a PVM task id from the task coordinator to satisfy this requirement. When replaying *pvm_gettid*, it may be necessary for the Pangaea library function to request a translation from an *ltid* to a *tid*. This occurs when the *tid* for the logical task to be returned doesn't correspond to the PVM instance id for the current execution and the recorded *ltid/tid* association is not already known.

The task coordinator is a C program that is spawned automatically during the initialization of the Pangaea top-level task. The developer may specify the host where the task coordinator is to run; if no specific host is specified, it is executed on the same machine as the Pangaea top-level task. The task coordinator is a simple program which waits for messages to arrive and processes them. The Pangaea initialization procedure registers a new PVM task id and its associated logical task id, after which the task coordinator may process translations for that task. If a translation request arrives before the associated task's registration, the request is queued and processed once the registration actually occurs.

5.3 Run-Time Hierarchy

Figure 5 shows the run-time hierarchy of Pangaea. The Pangaea top-level task spawns the application's top level tasks(s), which in turn may spawn additional child tasks, etc. If XPVM is used to spawn the application, then XPVM will spawn the Pangaea top-level task and supply it with the parameters needed to spawn the application.

Table 1: PVM Functions Instrumented by Pangaea

Task initiation	Sends	Receives	Group Operations
pvm_spawn	pvm_initsend	pvm_recv	pvm_gettid
	pvm_mkbuf	pvm_nrecv	
	pvm_send	pvm_trecv	
	pvm_bcast	pvm_probe	
	pvm_mcast		

5.4 Selective Logging and Replay

The Pangaea library contains a function that users can invoke to suspend and resume record and replay. When event recording is suspended, send and receive events are not recorded in the communication log files. Pangaea message headers are still added to outgoing messages, however, since the sending task does not know whether recording is suspended in the receiving task. Message headers are always unpacked by the receiving function for the same reason. Task spawns and group identification events are always recorded, as replay cannot be performed without this information. When event replay is suspended, send and receive messages are not matched with the contents of the communication log files, but message headers are still added and removed by the sending and receiving tasks.

Selective record and replay can be particularly useful when an error is correlated with a particular message type, or with messages received by a particular function, etc. In general, the communication log files can become quite large. When recording events for a quantum chromo dynamics application [4], for example, a single task recorded over 9,000 events in 45 seconds, using an average of 36 bytes per event (9,207 events used approximately 329K bytes). Restricting the events recorded and replayed to a small subset of all events can significantly reduce the space required by the log files.

Because record/replay suspension and resumption are derived from code annotations, a replay execution must use the same static selection as a previously recorded execution. Use of this feature must be done with care, since the ordering of unrecorded and non-replayed events cannot be guaranteed.

Use of static selection may result in unconnected communication events in the Pangaea event graph. A receive event displayed will have no matching send when the sender of message has record suspended while the receiver is recording. Similarly, a send event displayed will have no matching receive when the sender is recording while the receiver is not. This is not a problem per se, but the user needs to be aware of this possibility.

6 Pangaea Functionality

We now discuss the functionality implemented in Pangaea for the instrumented PVM functions for record and replay mode. These functions are listed in Table 1. Note that although *pvm_psend* and *pvm_precv* are defined in the PVM library standard, these functions are not actually supported by PVM and therefore are not instrumented by Pangaea[1].

1. Use of *pvm_send* and *pvm_recv* in PVM 3.3.9 results in an error message that these functions are not implemented.

6.1 Record Mode

Record mode logs all relevant PVM events for use during replay. Two log files are created for each task in record mode. The execution log contains task initiation events (spawns), and the communication log contains group identification requests, send events, and receive events.

A task initiation event is recorded for each invocation of *pvm_spawn*. The information recorded is used to guarantee that the same tasks will be executed on the same hosts during a replay execution.

The PVM buffer initialization functions *pvm_initsend* and *pvm_mkbuf* are instrumented to add a message header for unique identification of each message. No events are recorded for these functions. The message header includes the sender's *ltid* and message identifier.

Tasks may join named groups with the *pvm_joingroup* function. Each task joining a group is assigned a group instance id. While the behavior of group joins are deterministic, the assignment of instance identifiers depends on the order of completion by the joining tasks. After joining a group, tasks may broadcast messages with *pvm_bcast* to other group members without specifically identifying the receiving tasks. Since the destination of a broadcast message is a group name and group joins are deterministic, no special processing for *pvm_joingroup* is required.

Sending a message to a particular group member, however, requires that the sender specify the destination *tid*. Tasks must call *pvm_gettid* to obtain the *tid* of a specified group member (identified by group name and instance id). Since instance id assignment is non-deterministic, Pangaea must provide record and replay facilities for *pvm_gettid*. In record mode, the requested group and the *ltid* of the *tid* returned by *pvm_gettid* are recorded for use by replay.

A send event is recorded for each invocation of *pvm_send*, *pvm_bcast*, and *pvm_mcast*. The destination *ltid*(s) and message identifier are recorded for *pvm_send* and *pvm_mcast*. The destination group name and message identifier are recorded for *pvm_bcast*.

A receive event is recorded for each invocation of *pvm_recv*, *pvm_nrecv*, and *pvm_trecv*. These instrumented functions extract the Pangaea message header and record the sender's *ltid*, the sender's message identifier, and the message length.

Although *pvm_probe* is instrumented in Pangaea, no events are actually recorded for this function as results are irrelevant. Since a positive result from *pvm_probe* must be followed by a call to *pvm_recv*, it is sufficient to record and replay only the associated receive event.

6.2 Replay Mode

Replay mode reproduces the event ordering captured in a previously recorded session.

For task spawns (*pvm_spawn*), replay mode insures that the same tasks run on the same machines as a previously recorded session. For example, if Task A spawned Task B on the Sun workstation *seuss.ucsd.edu*, and Task C and Task D on the RS6000 workstation *trini.ucsd.edu*, then replay also spawns Task B on *seuss.ucsd.edu* and Task C and Task D on *trini.ucsd.edu*. This guarantees that each task has access to the log files created by a previously recorded execution. Logically, it would be sufficient to execute a spawned task on a machine with a file system that has access to the task's log files, but this is difficult to determine automatically. The user may, however, modify the execution log to control where tasks are executed during replay. This would be useful, for example, when the target system(s) for the application are expensive to use or are unavailable and alternate systems could be used for debugging. In this case, the user may need to copy or move log files to the appropriate systems to ensure that the appropriate files will be available for each task.

The PVM buffer initialization functions *pvm_initsend* and *pvm_mkbuf* are identical to record mode; they simply add the message header used to uniquely identify messages.

Group operations pose an interesting challenge for replay. PVM tasks may join groups, obtain other group members' PVM task ids, and send messages to each other. A complication for Pangaea occurs because PVM assigns each group member a nondeterministic group instance id, and tasks identify a group member by its instance id. Since the tasks execute asynchronously, this order cannot be predetermined. This has a subtle (and rather nasty) effect for the design of replay support, because the logical tasks of a group will need to have the same instance ids as for the recorded execution. Pangaea manages this group identification problem by instrumenting *pvm_gettid* to return the PVM task id of the task assigned the logical task id recorded for this invocation of *pvm_gettid*. As a result, the PVM task id returned by *pvm_gettid* may actually belong to a different instance id for the current execution, but will now be the same logical task as the recorded session.

Replay processing for send events (*pvm_send*, *pvm_bcast*, and *pvm_mcast*) insures that the same message identifier is sent to the same destination(s) as recorded in the communication log.

Replay of receive events comprises the heart of Pangaea. This is where the complexity of event order reproduction resides. Task initiation (*pvm_spawn*), group identification (*pvm_gettid*), send buffer initialization (*pvm_initsend* and *pvm_mkbuf*), and send events (*pvm_send*, *pvm_bcast*, *pvm_mcast*), merely enable the replay of receive events. To perform replay for *pvm_recv*, a synchronous receive, Pangaea invokes *pvm_recv* with the user's parameters until the message received matches the expected (previously recorded) receive event. Any messages received out-of-order are queued until the caller again invokes *pvm_recv*. If the expected message is in the receive queue, the queued message is returned to the caller in place of invoking *pvm_recv*.

The asynchronous receive functions *pvm_nrecv* and *pvm_trecv* require slightly different processing for replay since they don't necessarily return a message to the caller. As for *pvm_recv*, Pangaea returns the expected message from the receive queue if it is present. Otherwise *pvm_nrecv* (or *pvm_trecv*) is invoked. If *pvm_nrecv* returns a negative result (no message received), Pangaea immediately returns to the caller. Otherwise Pangaea determines whether the message received matches the message expected. If it does match, the message is returned to the caller. Otherwise the message is stored in the receive queue and a negative result is returned to the caller.

The *pvm_probe* function determines whether the next expected message has arrived. In replay mode, this function determines whether the expected message is in the receive queue, and if so, returns a positive result (message arrived). Otherwise Pangaea invokes *pvm_recv*, stores the received message in the receive queue, and determines whether the expected message was received. If the expected message was received, then a positive result is returned; otherwise a negative result is returned. After the caller obtains a positive result from *pvm_probe*, the subsequent call to *pvm_recv* returns the queued, expected message.

Typically, the asynchronous functions *pvm_nrecv*, *pvm_trecv*, and *pvm_probe* need not be specifically replayed. In some applications, however, the algorithm may be modified slightly if a message is not immediately available, as in exponential back off and retry. With the current implementation of Pangaea, the application could have a run-time error that is not reproducible in Pangaea. To support these types of applications, Pangaea would need to be extended to explicitly replay these functions.

7 Pangaea Overhead

Pangaea sends additional messages and requires additional computation and space. This increased overhead need only be incurred when actively debugging with Pangaea, however; normal executions need not utilize Pangaea.

To support record and replay, additional messages are generated by the Pangaea routines, mostly during task initiation and completion. When spawning a task, the parent sends each child one message containing the child's Pangaea logical task id (*ltid*). During task initialization, each task registers its PVM task id (*tid*) and its corresponding *ltid* with the task coordinator, generating one message per task. As each task finishes, it sends a completion message to its parent. At the conclusion of the top-level task, one message is sent to the task coordinator instructing it to exit. The first (application) send to an unknown destination generates two messages, one to request translation from a *tid* to an *ltid* from the task coordinator and one to answer the translation request. When requesting the *tid* of a group member, it may be necessary to request translation from an *ltid* to a *tid* from the task coordinator, generating one message for the request and one message for the reply.

When running Pangaea record and replay with the XPVM visual display, messages are sent from the Pangaea routines to XPVM for event notification: one message for each task spawned (*pvm_spawn*), one message for each send event (*pvm_send*, *pvm_bcast*, *pvm_mcast*), and one message for each receive event (*pvm_recv*, *pvm_nrecv*, *pvm_trecv*). This essentially doubles the communication volume for each task. Because of this high overhead, users are more likely to use record and replay mode without XPVM, and use Pangaea's post-mortem display facility for visualization of communication events. Fortunately PVM sends are asynchronous, so at least the application tasks are not blocked while sending event notifications to XPVM.

A test application was run with and without Pangaea to measure the added computational overhead. All timings were run independently of XPVM. The application was run using four tasks, each running on a separate Sun4 or RS6000 workstation. Timings were obtained for 500 to 1 million invocations of each PVM function. Executables were compiled with full optimization. Most messages were approximately 44 bytes in length. All timings were performed on idle systems, apart from Pangaea, PVM, and the application. Table 2 contains the resulting measurements and comparisons of the application run without Pangaea, with Pangaea logging (record mode), and with Pangaea in replay mode. All CPU utilizations are reported in milliseconds and include both user and system time. Values reported are averages.

Table 2: Pangaea Overhead

PVM Function	Dependent Function	Without Pangaea (msec)	Logging		Replay	
			With Logging (msec)	Added Overhead (msec)	With Replay (msec)	Added Overhead (msec)
pvm_bcast	Buffer Init and Msg Packing	18.014	21.873	3.859	22.460	4.446
pvm_mcast	Buffer Init and Msg Packing	3.659	3.889	0.230	4.182	0.523
pvm_nrecv	Unpacking	0.118	0.201	0.083	0.644	0.526
pvm_probe	pvm_recv	0.110	0.142	0.032	0.902	0.792
pvm_recv	Unpacking	0.624	0.989	0.365	1.514	0.890
pvm_send	Buffer Init and Msg Packing	1.279	1.588	0.309	1.874	0.595
pvm_spawn	None	17.996	31.425	13.429	22.331	5.335
pvm_trecv	Unpacking	0.225	0.385	0.160	0.756	0.531

Most of the measurements include calls to other PVM functions which are always invoked in conjunction with the particular function listed. These functions are identified as "dependent functions" in Table 2. For *pvm_send*, for example, this includes the CPU utilization of the buffer initialization and message packing that is a prerequisite to *pvm_send*. Similarly, once *pvm_probe* indicates that the requested message has arrived, *pvm_recv* must be called to actually receive the message. It was not feasible to measure CPU time separately for *pvm_send* and the buffer initialization (*pvm_initsend* or *pvm_mkbuf*), as there was insufficient precision in the timing information available to measure individual function calls. More importantly, it is not necessary to measure them separately since they must be invoked together to complete a communication activity.

As expected, Pangaea does add computational overhead to PVM library calls for recording events. Most of this is due to I/O to the log files. There is one record created for each send event (*pvm_send*, *pvm_bcast*, *pvm_mcast*), one for each successful receive event (*pvm_recv*, *pvm_nrecv*, *pvm_trecv*), one for each group member identification (*pvm_gettid*), and several for each task initiation (*pvm_spawn*). The relative impact of this added overhead will depend on the ratio of computation to communication in the application. Assuming that a task invokes *pvm_spawn* once, *pvm_send* 10,000 times and *pvm_nrecv* 35,000 times, then the Pangaea overhead would add 6.1 seconds of CPU utilization in record mode and 24.4 seconds in replay mode. With a computation to communication ratio of 10:1, this results in a 3.3% increase in CPU utilization in record mode and 13.1% in replay mode.

Another component of increased overhead includes the message header added to each send event. The average size of the message header is nine bytes, four for the Pangaea message id and five for the sender's *ltid*. The message header increases the length of each message sent and adds overhead for packing and unpacking the header.

There is also a fixed amount of overhead incurred by using the Pangaea system in any mode. Each application must invoke Pangaea initialization and cleanup functions, which use total CPU utilization of 20.4 msec and 17.5 msec, respectively. The Pangaea top-level task uses 243.3 msec total CPU time, and the Pangaea task coordinator uses 168.0 msec total CPU time. The task coordinator and the top-level task spend most of their time waiting, so it is not surprising that their CPU utilization is negligible.

The actual size of the log files created depends on the number of events recorded. The average number of bytes per event for SOR in the example execution is 29. The execution logs will always be small since they only contain spawn events. Each log file contains a header describing the format of the contents, and thus has a small, fixed, minimum size. The Pangaea execution log will always contain at least one spawn event since it initiates the application, although its communication log will be empty. No log files are generated for the Pangaea task coordinator.

Replay mode incurs the same amount of fixed overhead as record mode. The same message headers are added to each send event. The main difference between replay and record mode is that replay reads the log files instead of writing to them, and it caches into memory the incoming messages that are received before they are expected. This data shows that all instrumented PVM functions, with the exception of *pvm_spawn*, add more overhead for replay than for record. This result was expected since there is more processing to perform for replay than for record mode for these functions. Replay processing for *pvm_spawn* is actually simpler than that for record, and thus has lower overhead.

8 Conclusion

In this paper, we have described Pangaea, a debugger for parallel applications implemented on heterogeneous distributed systems. Pangaea combines the visualization and trace facilities of XPVM with a new replay facility and a new event graph view. Pangaea provides the means to capture the event ordering of a particular execution and reproduce that same ordering as many times as needed. This capability aids in debugging software errors caused by non-deterministic communication patterns produced by heterogeneous applications.

There are several ways in which the scope of Pangaea can be expanded. The initial implementation supports only C programs; other language interfaces, particularly Fortran, need to be developed. In addition, it would be useful to add additional views to

Pangaea, or even to add a facility for customizing views, like that available in Panorama [12]. The new Pangaea event graph view in XPVM could be enhanced to allow more sophisticated re-size and zoom capabilities. A dynamic selection capability could be added to allow selective logging and replay at run-time in addition to the existing static selection capability. Pangaea could also be implemented for MPI applications once MPI supports task initiation. This would increase the availability of Pangaea's capabilities to heterogeneous application developers.

To improve ease of use, the Pangaea modifications to XPVM could be incorporated into the official XPVM release. It is also feasible to implement the Pangaea facilities for record, replay, and display directly into PVM. With Pangaea features implemented in PVM and XPVM, there would be no need to install a separate Pangaea package.

Pangaea, coupled with XPVM, provides a useful tool for replaying and visualizing executions of parallel applications on heterogeneous distributed systems. Heterogeneous systems are becoming more and more prevalent, and tools such as these play an important role in the development of efficient applications which can exploit the performance potential of such systems.

9 References

[1] Anglano, C., Schopf, J. Wolski, R., and Berman, F. *Zoom: A Hierarchical Representation for Heterogeneous Applications.* University of California at San Diego, Computer Science and Engineering Technical Report CS95-451, 1995

[2] Beguelin, A. *Xab: A Tool for Monitoring PVM Programs.* In Workshop on Heterogeneous Processing, Los Alamitos, California, April 1993, pp. 92-97.

[3] Beguelin, A., Dongarra, J., Geist, G.A., Manchek, R., and Sunderam, V. *A User's Guide to PVM Parallel Virtual Machine.* Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[4] Bernard, C., Ogilvie, M. C., DeGrand, T. A., DeTar, C. E., Gotlieb, S. A., Krasnitz, A., Sugar, R. L., and Toussaint, D. *Studying quarks and gluons on MIMD parallel computers.* International Journal of Supercomputer Applications 5, 4 (Winder 1991), pp. 61-70.

[5] Cheng, D. and Hood, R. *A Portable Debugger for Parallel and Distributed Programs.* Supercomputing, 1994 (URL http://wk122.nas.nasa.gov/NAS/Tools/Projects/P2D2/).

[6] Choi, J. and Stone, J. M. *Balancing Runtime and Replay Costs in a Trace-and-Replay System.* In Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, pp. 13-22, May 1991.

[7] CXSOFT's ConvexPVM System. URL http://www.convex.com/prod_serv/cxsoft/Products/pvm_inf.html.

[8] Damodaran-Kamal, S.K. *Xmdb Version 1.0 User Manual 1.2.* Los Alamos National Laboratory, 1995 (URL http://www-c8.lanl.gov/dist_comp2/mdb/mdb.html).

[9] Forin, A. *Debugging of Heterogeneous Parallel Systems.* International Workshop on Parallel and Distributed Debugging, 24(1) 130-141, January 1989.

[10] Kuppermann, A., and Wu, M. *Quantum Reaction Dynamics on a Gigabit/Sec Network.* Proceedings of the Gigabit Testbed Maxijam, November 1995.

[11] LeBlanc, T., Mellor-Crummey, J. *Debugging parallel programs with Instant Replay.* IEEE Transactions on Computers C-36, 4, April 1987, 471-482.

[12] May, J.M. *An Extensible, Retargetable Debugger for Parallel Programs.* Technical Report Number CS94-375, Department of Computer Science and Engineering, University of California at San Diego, June 1994.

[13] Mechoso, C., Farrara, D., and Spahr, J. *Achieving Superlinear Speedup on a Heterogeneous Distributed System.* IEEE Parallel and Distributed Technology, pp. 57-61, Summer 1994.

[14] Message Passing Interface (MPI). URL http://www.arc.unm.edu/workshop/mpi/mpi.html.

[15] Miller, B. and Choi, J. *A Mechanism for Efficient Debugging of Parallel Programs.* Proceedings of the ACM SIGPLAN '88 Conference on Programming Language Design and Implementation, published in ACM SIGPLAN Notices 23, 7, pp. 135-144, July 1988.

[16] Neuman, B. C., and Rao, Santosh. *The Prospero Resource Manager: A Scalable Framework for Processor Allocation in Distributed Systems.* Concurrency: Practice and Experience, Vol 6(4), pp. 339-335, June 1994.

[17] Pancake, C. M. *Why is there such a mis-match between user needs and tool products*? Presented at the 1993 Workshop on Parallel Computing Systems, Keystone, Colorado, April 1993.

[18] Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, 1991, 1993.

[19] XPVM: *A Graphical Console and Monitor for PVM.* (URL http://www.netlib.org/utk/icl/xpvm/xpvm.html).