HEWLETT
PACKARD
Concurrent Computing Department
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

Program replay in PVM

Milon Mackey

May 12, 1993

Introduction

REAPP = Research on Environments and Applications for Parallel Processing.

Current platform is clusters of workstations.

We are:

- Ming Hao
- Alan Karp
- Milon Mackey
- Vineet Singh

Working on IVD, integrates performance and data visualization with debugging.

One of our first goals is to have IVD work with PVM programs.

Outline of the talk

- Overview of program replay
- Interface to program replay
  - PVM hostfile
  - PVM console
  - PVM library
- Future work
- Related work

This work is still in progress.

Details may change.

Overview of program replay

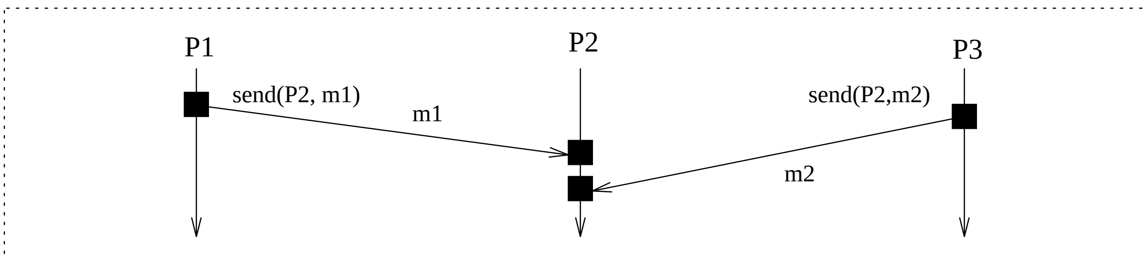
What is it?

Most parallel programs have:

— deterministic output

— non-deterministic internal behavior.

Caused by races.



This causes problems for debugging.

Program replay solves this problem through two steps:

1. *Trace* — program is executed — resolution of races recorded in a trace.
2. *Replay* — program is re-executed — trace guides the program to the same execution.

Internal behavior is now deterministic.

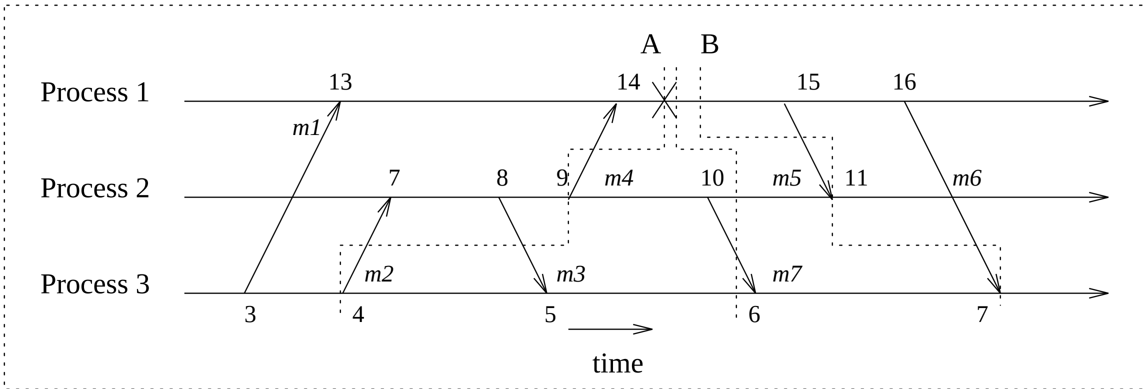
Overview of program replay

Why have it?

- Makes cyclic debugging work for parallel programs.
- Tracing has small probe effect.
- Compute Lamport's "happened before" relation.
- Synchronize with performance visualization.
- Gives support for causal distributed break points.
- Trace optimized code. Debug symbolically during replay.

Overview of program replay

Causal distributed breakpoint



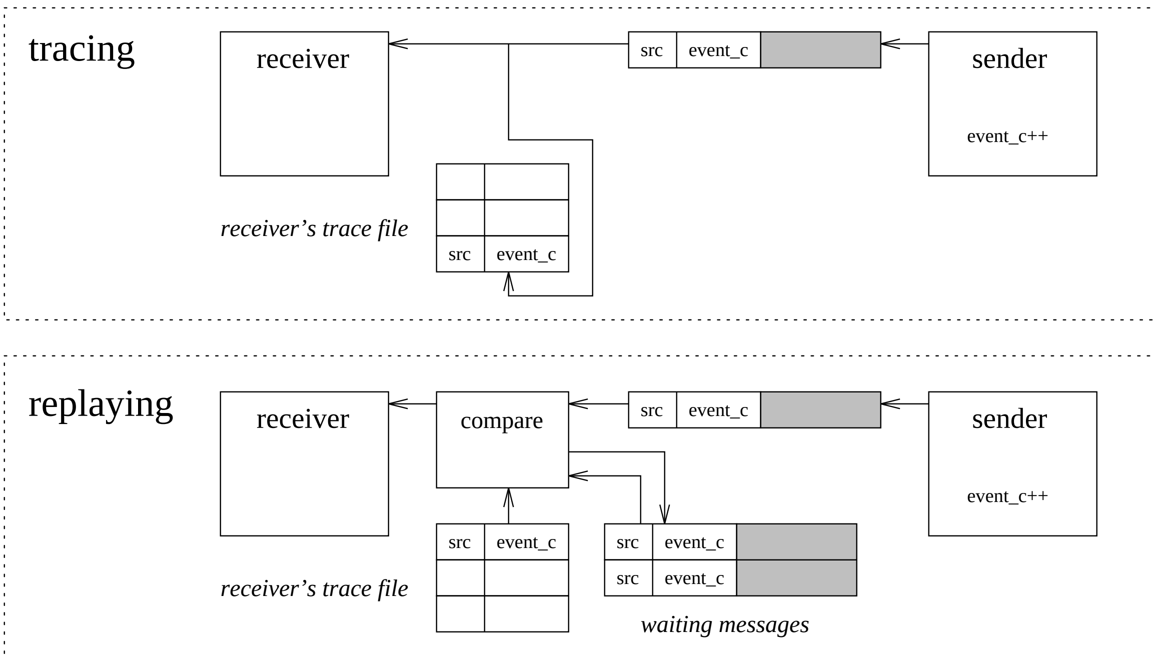
Overview of program replay

How does it work?

Each PVM task and daemon has its own trace file.

Writes it during tracing.

Reads it during replay.



Overview of program replay

How is it implemented?

Implemented in PVM 3.0.

Small changes to:

- PVM library
- PVM daemon
- PVM console

Most changes in PVM daemon.

About 2000 lines of new code.

Program replay in PVM

User interface

- PVM hostfile
- PVM console
- PVM library

PVM hostfile

Location of trace files

New "`tf=`" option controls the location of the trace files.

Example of a PVM hostfile:

```
# The trace files on the host "betti" are kept
# in $HOME/pvm3/traces.
betti

# The trace files on "pina" are kept in $HOME/traces.
pina tf=traces

# The trace files on "marina" are kept in /tmp/traces.
marina tf=/tmp/traces

* tf=/tmp/junk/monday

# The trace files on "giovanna" are kept
# in /tmp/junk/monday.
giovanna
```

Trace names

Traces have names.

Syntax: $\{dir/\}$+$name$, where $\{\}$+ means 0 or more occurrences of the enclosed text.

dir is the name of a directory

name identifies the trace.

Examples:

```
foo/bar
test.1
a/b/c/helpme
```

Modified and new commands

- `reset`

  Kills all PVM tasks and stops tracing and/or replay.
- `trace` *tr* [*topts*] [[*sopts*] *file* [*argv*]]

  Make a trace.
- `replay` *tr* [[*sopts*] *file* [*argv*]]

  Replay a trace.
- `replaytrace` *tr* *n-tr* [*topts*] [[*sopts*] *file* [*argv*]]

  Replay a trace and make a new trace.
- `rmtrace` *tr*

  Get rid of a trace.
- `traces`

  List the names of all the traces that exist.
- `traceinfo` *tr*

  Get information about a specific trace.
- `debug`

  Toggles running with debuggers.
- `status`

  Show the status of debuggers, tracing, and replay.

Relation of PVM console to program replay

Messages sent and received by PVM consoles are not traced.

These PVM console commands cannot be used during tracing or replay:

- `add`
- `delete`
- `kill`
- `replay`
- `replaytrace`
- `sig`
- `spawn`
- `trace`

PVM console

Example

```
> trace junk/r1 wmod 57 93
>                       # Trace the application.
>
> ps -a                 # Verify that wmod is finished.
> reset                 # Stop the trace.
> replay junk/r1        # Replay the application.
>
> reset                 # Stop the replay.
> debug                 # Run all tasks with debugger.
> replay junk/r1        # Replay the application again.
>
> reset                 # Stop the replay.
> replaytrace junk/r1 junk/r1.a to=TSEND
>                       # Replay the application and
>                       # this time make a trace that
>                       # also records when messages
>                       # were sent.
>
> reset                 # Stop the replay.
> replay junk/r1.a      # Replay it from the new trace.
>
> reset                 # Stop the replay.
> spawn wmod 57 93      # Run the application with no
>                       # tracing or program replay.
```

Modified and new routines

- `void pvm_console()`

  Distinguishes a PVM console from a normal PVM task.

- `pvm_pgmreplay(...)`

  Controls tracing and replay.

- `pvm_traceinfo(char *`*tr*`, struct traceinfo *`*ti*`)`

  Returns information about a trace.

- `pvm_traces(char ***`*tr-names*`)`

  Returns a list of all the existing traces.

- `pvm_reset()`

  Modified to stop tracing and replay.

- `pvm_rmtrace(char *`*tr*`)`

  Removes the trace *tr*.

- `pvm_spawnflags(int `*flags*`, int `*op*`)`

  Controls flags used when a task is spawned.

- `pvm_status(struct pvmstatus *`*st*`)`

  Returns the status of tracing, replay, and the spawn flags.

The details of pvm_pgmreplay()

pvm_pgmreplay(pgmreplay_t **p*,
char **task*,
char ***argv*,
int *flag*,
char **where*, int *ntask*, int **tids*)

```
typdef struct flag_t_ {
  int task;         /* tid of task */
  int on;           /* flags to set */
  int off;          /* flags to un-set */
} flag_t;

typedef struct stop_t_ {
  int task;         /* tid of task to stop */
  event_t event_c; /* event it should stop after */
} stop_t;

typedef struct pgmreplay_t_ {
  char *replay;     /* R name of trace to be replayed */
  int checkpoint;   /* R check point to start from */
  flag_t *flags;    /* R control of spawn flags */
  stop_t *stop;     /* R tasks to stop during replay */
  char *trace;      /* T name of trace to make */
  topt_t topts;     /* T what should be traced */
  error_t *errors; /* B what went wrong */
} pgmreplay_t;
```

Program replay in PVM

Future work

- Finish the implementation of program replay.
- Experiment with it.
- Measure and tune it.
- Add checkpoint and restart.

Related work

Program replay

- Thomas J. LeBlanc and John M. Mellor-Crummey. Debugging parallel programs with Instant Replay.
- Eric Leu and André Schiper. Execution replay: a mechanism for integrating a visualization tool with a symbolic debugger.
- Robert H. B. Netzer and Barton P. Miller. Optimal tracing and replay for debugging message-passing parallel programs.

Using the "happened before" relation.

- Robert J. Fowler, Thomas J. LeBlanc, and John M. Mellor-Crummey. An integrated approach to parallel program debugging and performance analysis on large-scale multiprocessors.

Causal distributed break points.

- Jerry Fowler and Willy Zwaenepoel. Causal distributed breakpoints.