Debugging Highly-Parallel Programs

João M. Lourenço, José C. Cunha and Vitor Duarte
CITI / Universidade Nova de Lisboa

joao.lourenco@fct.unl.pt
Why do programs have errors?

Problem

- Domain knowledge errors
- Devise a computational solution

Solution

- System failure
- Inadequate performance
- User errors
- Run-time errors
- Compilation errors
- Algorithmic errors

Write a computer program

Devise a computational solution
What about parallel programs?

Problem

Domain knowledge errors

Devise a computational solution

Specification errors

Algorithmic errors

Synchronous execution errors

Interleaving errors
All men are equal!
What about errors?

Yields a correct result, although it takes longer than acceptable

Unwanted side effects caused by non-reentrant code and shared data

Ordering failures and deadlocks

Violations of precedence or mutual exclusion relations

Byzantine

Performance

Interleaving

Synchronization

Ordering

Sequential errors

Non fail-stop errors

Harder

Easier
Parallel program

Multicore system

Expected behavior

Observed behavior

?
Parallel computations

Parallel program

Running

generates

Multicore system

Parallel computations

What?

How?
Program histories

- **Local History**: $h_i$ – sequence of events generated by executing the program “$p_i$”
  - $h_i = e_i^0, e_i^1, ..., e_i^f$
  - $k^{th}$ event in $h_i$ ($e_i^k$) produces the local state $s_k$

- **Global History**: $H$ – union of the local histories of $N$ processes
  - $H = h_1 \cup h_2 \cup ... \cup h_N$
Parallel Computation

• A parallel computation is a partially ordered set \((\text{poset})\) defined as

\[ C^D = (H, \rightarrow) \]

\[-\rightarrow = \text{Global history}\]

\[-\rightarrow = \text{Lamport’s } \textit{happens before} \text{ relation}\]
A cut of a parallel computation is a subset $C$ of its global history $H$ that contains an initial prefix for each of its local histories

$$C = \{h_1^x, h_2^y, ..., h_n^z\}$$
The **frontier of a cut** is the set of the last states/events in a cut

\[ F = \{s_1^x, s_2^y, ..., s_n^z\} \]

The frontier of a cut defines a global state
A cut is consistent if for all events in its frontier, all their past events are also included in the cut.
• A **global state is consistent** if it corresponds to the frontier of a consistent cut.
Runs

7 states

6 states

30 states

Evolution of P_1

Evolution of P_2
Observing a parallel program

- Process (P₁)
- Process (P₂)
- Process (Pₙ)

Events / states

Local histories

Run

Consistent run

Observation

Consistent observation

Internal & interaction events

Time precedence

Time precedence

Permutation

Casual precedence constraints

Global history

Union

Arbitrary total order

Causal precedence constraints

Subset

Parallel computation

Cut

Consistent cut

Frontier of a consistent cut

Casual precedence constraints

Last events in cut
Observing a parallel program

Process (P₁)
Process (P₂)
Process (Pₙ)

Events / states

Local histories

Global history

Cut

Consistent cut

Parallel computation

Program execution

Consistent run

Developer perspective

Consistent observation

Frontier of a consistent cut

Internal & interaction events
casual precedence constraints

Global history

union
subset
casual precedence constraints
arbitrary total order
time precedence
casual precedence constraints
permutation
casual precedence constraints

Consistent program state
Observing and debugging

- Interactive debugging of remote processes
  - to obtain reproducible behavior

- Trace, replay and debugging
  - to analyze alternative paths

- Combined testing, steering and debugging
  - to evaluate correctness properties

- Global predicate detection

- State based debugging
  - observation of program states

- Deterministic re-execution
  - repeatable observations

- Systematic state exploration
  - alternative observations

- Global program properties
  - observation of consistency
The scaling challenge

• How to deal with hundreds (or thousands) of threads?
  – Collect, store and gather observations / logs
    • What shall be the detail level?
    • Logs may be huge
  – Combining the logs
  – Reason about global observations
    • Visualize large amounts of information
    • Evaluate global predicates on the program state
    • Evaluate global predicates on the program run
  – Map observation points to the original program
    • Dealing with code-generators
    • Supporting high-level abstractions, DSLs
Happy debugging! 😊