IGOR: A System for Program Debugging via Reversible Execution

Stuart I. Feldman

Channing B. Brown

slides made by Qing Zhang

Abstract

- Typical Debugging
- IGOR
  - Reverse Execution
  - Selective Searching of Execution History
  - Substitution

Introduction

- Typical Debugger
  - Dumping
    - Octal dump
  - Breakpoint
    - Trace
    - Selective Checking

Example

tree *p, * q;

p = q;

- p-> right_child -> left_child = q; /*whoops*/
- (cyclic tree)
- Possible delayed side effect
- Messy!

IGOR

- Prototype Debugging System
- Snapshots
- Reverse Execution
- Portable

Earlier System

- COPE
  - Require expensive support
  - Full interpretation
  - Generation of code with inversion options
  - Special recompilation to threaded code

Implementation Environment

- In C
- Motorola-68000-based
- DUNE (OS)
  - Supports Unix System V

Aims and Limitations

- “Almost” no effect
- Random Memory Access
- Irreversible I/O
- Synchronous

System Changes

- Compiler
- Library
- Loader
- Modified Kernel

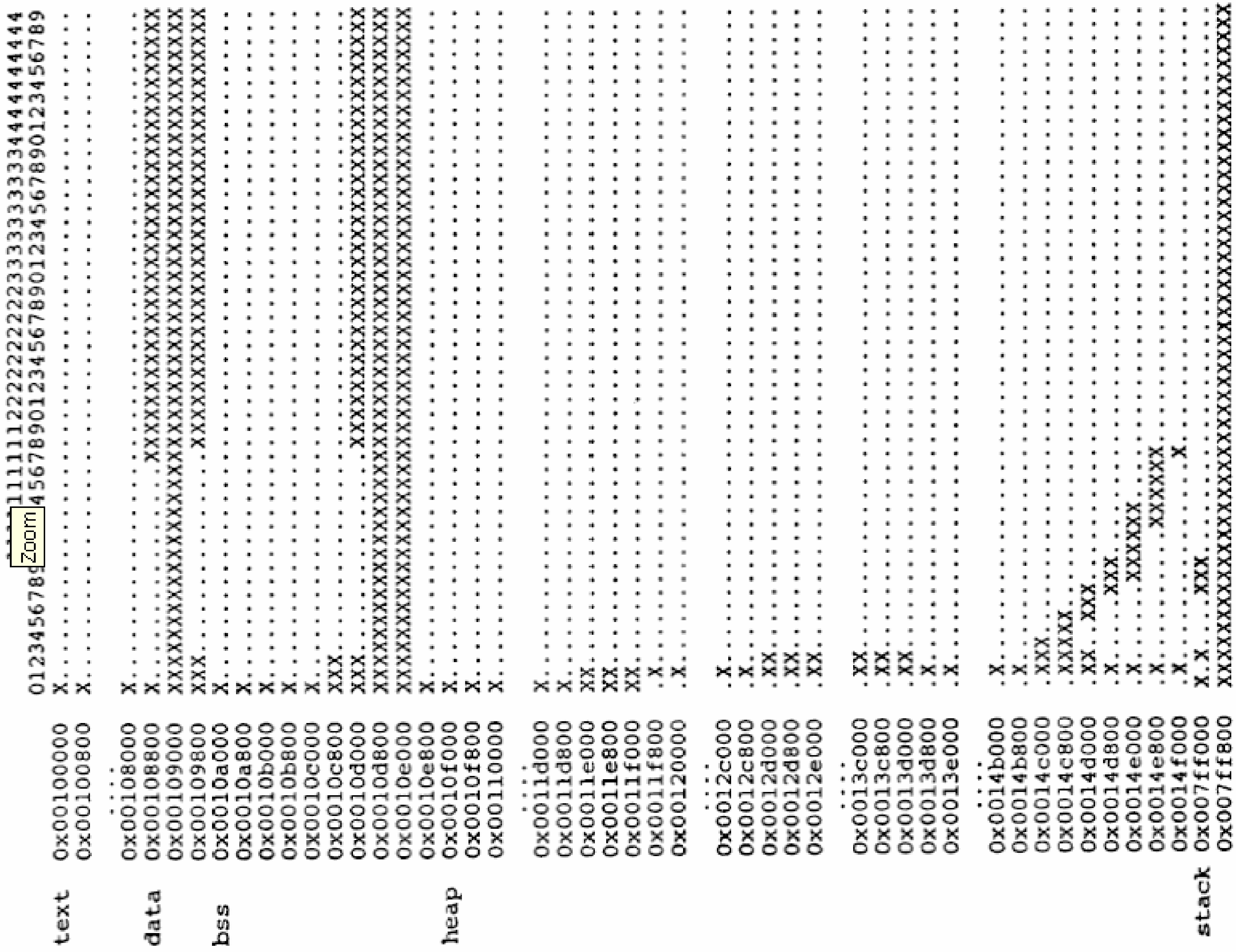
Reviving Dead Programs

- Illegal Termination
- Core Image
- Restart from a Core
- Picks up at next C instruction
- Only works for some Situations
  - i.e. running out of stack space

Checkpoints

- Save State
- Periodic Logging
- Size and Time
- Idea - not to save every page for every checkpoint
- Save used pages
  - i.e. Demand paging

Memory Reference



A Graph of Checkpointed Pages

Graph Analysis

- Unix “sort” program
- X – page dumped during the checkpoint
- Bottom of the Stack
- External variables
- Others aren’t as “nice”

Special System Calls

- Pagemod
  - List accessed memory pages
- Ualarm
  - Similar to Unix alarm syscall
  - Counts CPU time
- More costly
- Write protecting all pages
- Inadequate for debugging without ualarm

Dynamic Function Replacement

- Dynamically replace data elements
- Replace one or more C functions
- Restart execution from checkpoint
- Problems that arise

Dynamic Function Cont …

- Special Loader
- First few bytes replaced
  - (The old can be restored by replacing the initial bytes)
- Variables retain their value from previous version

Reversible Execution

- Selection Criterion
- Object Code Interpreter
  - i.e. x > 0;
  - Monotone value
  - Currently only comparison between a variable and a constant

Performance

- Compiler
  - 17 % greater than standard
    - “Typical” source file
  - 37 % greater for compilation
    - 4700-line program

Performance continued …

- Ran IGOR on existing functions
  - Make
  - C compiler’s main pass
  - Sort with 2853 line input file
- Table1 shows Execution overhead
- Table2 indicates the avg % of pages written out during execution.

Table 1: Execution Time Overhead

	Make		Ccom		Sort	
Checkpointing Interval (seconds)	Execution Time (seconds)	Overhead	Execution Time (seconds)	Overhead	Execution Time (seconds)	Overhead
none	4.7	-	12.1	-	22.2	-
1.0	8.0	70%	24.3	101%	31.7	43%
0.3	11.3	140%	34.3	183%	37.3	68%
0.1	22.7	380%	46.5	280%	47.5	114%

Table 2: Checkpointed Non-text Pages

Program	Checkpointing Interval (seconds)	Total Pages Accessed	Avg. Pages Written per Checkpt.	Percent Written
Make	1.0	46	34.0	74%
	0.3	47	27.9	59%
	0.1	47	23.9	51%
Ccom	1.0	89	41.2	46%
	0.3	89	36.5	41%
	0.1	89	30.7	34%
Sort	1.0	143	6.9	5%
	0.3	143	9.2	6%
	0.1	143	8.9	6%

Performance of Loader

- 4700-line program w/ 9 modules
- 4.1 s of CPU time
- ~21 % over the standard link

Future work

- Fancy Interface
- Second Storage Reduction
- Multiple Processors
- Extend to Asynchronous and Multi-Thread