Virtual Execution Environments: Support and Tools

PIs: Bruce Childers#, Jack Davidson*, Mary Lou Soffa*

Apala Guha*, Jason Hiser*, Naveen Kumar#, Jing Yang*, Min Zhao#, Shukang Zhou*, Kim Hazelwood*

#University of Pittsburgh
*University of Virginia
Virtual Execution Environments

• Increasing interest in Virtual Execution Environments (VEEs)
• **Research focus**: Translation based VEE - examines and translates a program’s instructions
• **Our goals**
  - Improve performance and memory overhead
  - Develop tools to enable the widespread acceptance of VEEs
A Typical Translation-Based VEE

- **Application layer**
- **OS + hardware layer**
- **VEE layer**
  - Translation engine
  - Code cache
Techniques to improve performance

Performance
- Implemented VEE (Strata); explored overheads
  - Indirect branches expensive - context switch
- Indirect branches from conditionals
  - Indirect branch translation cache
  - Reduced overhead from 4.1X to 1.7X
- Indirect branches from returns
  - Reduced overhead from 1.7X to 1.3X
Reduce memory overhead

- **Reduction in memory footprint of code caches**

- **Exit stubs**
  - They are used very few times
  - They have standard functionality
  - They occupy a considerable percentage of code caches
Our Approaches

- Deleting exit stubs
- Avoiding generation of exit stubs
- Reducing the size of exit stubs
- Generating target address specific stubs
Evaluation – Stub Occupancy

**Standard implementation**
- Code Cache Occupancy as Percentages
  - Exit Stubs: 64%
  - Traces and Other: 36%

**After Technique 4**
- Code Cache Occupancy after Applying R+TAS
  - Exit Stubs: 43%
  - Traces and Other: 57%
Tools

- Instrumentor for various VEEs
- Dynamic Optimizer
- Debugger for dynamically optimized code
Trace-based Dynamic Optimizer

Program

Dynamic Translator

Code Cache

TDO

Host Machine

Fetch blocks

Control transfer

Execute

Optimize trace

Emit

Emit

Virtual Machine

NSF Next Generation Software Program
Challenges in Debugger

1. **Static debug information inconsistent**
   - Code is generated, modified, duplicated and deleted continuously during execution
   - Active debug environment needed
   - Code location problem - opt and duplication

2. **Re-optimization & trace combination**
   - Data-value problem - expected value

3. **Efficiency**
   - Frequent optimization of traces
   - Code duplication and code cache flushes
Debug Information Repository

Dynamic Optimizer

- Mapping Generator
- Annotator

Debug Engine

- DIR
- Record-replay Manager
- Execution Manager
- Breakpoint Manager

Native Debugger

NSF Next Generation Software Program

March 25-26 2006
Experimental Results

- **Dynamic Optimizer**: Strata-DO;
- **Native Debugger**: Gdb 5.3
- **SPARC v9**: Sun Blade 100; 500 MHz; 256 MB
- **SPECint2000**

- Can report all expected values except those deleted by optimizer
- **Performance overhead** - 2.6%
- **Memory overhead average** 685 KB

- Overheads are comparable to those debuggers for statically optimized code
Summary and future research

• Demonstrated that SDTs and tools can be efficient

• **Current and future research**
  - Limit study for dynamic optimizations to determine potential
  - Advanced execution system that automatically adapt application’s execution to resource landscape originating from process variation
Questions?

Thank You

For more information, please visit:

http://www.cs.pitt.edu/coco