Verifying Software for Multi-core Systems

Presented by:
Michael McDougall
mcdougall@grammatech.com

GrammaTech, Inc.
317 N Aurora St.
Ithaca, NY 14850
Tel: 607-273-7340
E-mail: info@grammatech.com
Motivation

Old: software got faster as hardware improved
New: must parallelize software to benefit

Challenges:

› Concurrency bugs subtle, hard to diagnose
› Verification hard for concurrent systems
› Multi-core less forgiving
   • Parallelization must be fine grained
   • Shared memory behaves in odd ways

• DARPA project:
  › Flag bugs in lock-free algorithms
GrammaTech, Inc.

- ~25 people, including 10 phds
- Founded by Tim Teitelbaum (Cornell) and Tom Reps (Wisconsin)
- Locations: Ithaca NY, San Jose CA, Madison WI, Rochester NY
- Expertise: software analysis (static & dynamic) of source and binary
- Applications:
  - Software assurance (correctness, bug finding, malware detection)
  - Software re-writing (legacy software, anti-reverse-engineering)
- Research projects: NASA, Army, AF, Navy, OSD, NSF
- Products:
  - CodeSonar: bug finding for C/C++/Ada
  - CodeSurfer: program understanding + analysis library
- Customers: 150+
  - Lockheed Martin, FDA, Qualcomm, LG Electronics, NASA
Focusing on *Formal Verification*

- This talk focuses on verification by *automatic analysis of software*
  - Ideally, exhaustive exploration of software behavior without running the software
  - In practice, partial exploration that complements other verification methods
- Not covering
  - Traditional testing
  - System testing
  - Inspections/audits
  - ...though formal verification techniques can be applied there too
Software Analysis Overview

Rigorous
Complete
Exhaustive

- **Model checkers**
  - Proof of correctness
  - DARPA project

- **Bug finders**
  - Detect buffer overflows, NPD
  - False positives/negatives
  - Product: CodeSonar

- **Enforce Best practices**
  - NASA/JPL SBIR Phase II

Requires special expertise
Can give to engineering team

Easy to apply
Fast
Scalable
Sequential Code

Want to consider all inputs

Step 1 -> Step 2 -> Step 3

4 -> 144
5 -> 342
6 -> 452

...
Why concurrency is hard

Concurrent code: want to consider all inputs +all interleavings

Extra burden for the writer & the verifier
Multi-core verification inherits a lot

- Verification community has long focused on concurrency problems
  - Classical problems: dining philosophers
  - Protocol verification
  - Cache-coherence
- Multi-core verification inherits long list of tricks/techniques
  - Partial-order reduction
  - Exploiting symmetry
    - In State
    - In Threads
What’s new with multi-core?

• Urgency
  › Software developers have no choice now
  › Performance has to come from keeping more cores busy

• Focus on performance / ease of porting
  › Shared memory more important than message passing

• Game developers hitting this already
  › Sony Playstation 3 : 8 cores
  › Tom Leonard, VALVE:
    • High-level concurrency primitives too slow
      – Mutex, semaphores, etc
    • Extensive reliance on lock-free algorithms
Bridging the Gap

Single-core application

Lock-free synchronization

Model checking for multi-core

Multi-core application

Slow

Correct
Lock-Free Algorithms

Queue data structure

Synchronization data structures (locks, semaphores)

Compare-and-swap instructions

Traditional Concurrent Queue

Compare-and-swap instructions

Lock-free Concurrent Queue
Relaxed Memory Models

- Multicore systems do not respect sequential consistency
Consequences

• Relaxed memory models
  › “Proven” algorithms fail on multi-core system
  › Impact not well understood

• Generally
  › Multi-threaded code works on single core, but not multi-core
  › Multi-core parallelism much finer grain
  › Multi-threaded code works faster on quad core when you turn 3 cores off

• Safer (& sometimes faster) to turn cores off
GrammaTech’s DARPA project

- Lock-free data structure + harness

  Serial model

  - All possible serial executions
  - Relaxed memory model
  - All possible relaxed executions

- SMT Solver

  Equal: OK
  Not equal: counterexample
Software Analysis: What is needed

Rigorous
Complete
Exhaustive

- Model checkers
  - Proof of correctness
  - DARPA project

Better understanding of multi-core-specific issues

- Bug finders
  - Detect buffer overflows, NPD
  - False positives/negatives
  - Product: CodeSonar

Make aware of concurrency, with minimal impact on scalability

What are the best practices for multicore?

- Enforce Best practices
  - NASA/JPL SBIR Phase II

New languages, abstractions, memory models that help verification

Easy to apply
Fast
Scalable
Conclusion

• Verification for multi-core inherits from long tradition of concurrent verification
• These techniques need to be adapted for the concrete problems of multi-core
  › Tight coupling of CPUs
  › High-performance use of shared memory
  › Discrepancy between source POV and memory op POV not well understood
The End

• Acknowledgements:
  › Thomas Reps (GrammaTech + U Wisconsin)
  › Sebastian Burckhardt (Microsoft Research)

• Questions?