EE392C Emerging Applications: Verification Applications

April 15, 2003
David Bloom
Suzanne Rivoire
John Whaley
Outline

- Motivation
- Beyond simulation and testing
- Model checking
- Theorem proving
- Hardware support
- Summary
Motivation

- Mission-critical systems
  - Ex. Space shuttle, medical instruments
- Complex, expensive systems
  - Ex. Telephone switching systems, arithmetic units in CPU
- Used widely for both software and hardware systems
Beyond Simulation and Testing

- Simulation and testing require the development of inputs (stimuli), and observation of outputs
  - Only as good as your test cases
- Adequate for many commercial applications, but not good enough for critical systems and such
- Formal verification exhaustively proves the correctness
  - Much more time consuming and complex
Model Checking

- Create a finite state description of a system to be verified
- Exhaustively search the finite state space to determine if a specification is true
- 3 main steps in model checking:
  1. Create the model
  2. Specify properties that must hold
  3. Verify model against specifications
Model Checking

- Verification should always terminate with a true or false condition
  - But, complexity of the model (number of finite states) can explode
  - Process of verification is automatic, but can be prohibitively long
    - A lot of research on state reduction, which is not of interest to us
    - But, perhaps we can exploit parallelism
Model Checking

- Large state space can be partitioned into subspaces
  - Subspaces can be processed in parallel – great for TLP
- Tend to be memory bound processes – large ratio of memory to arithmetic instructions
  - Access patterns mostly random – little to no locality to exploit
    - Perhaps software prefetching can help
Model Checking – Case Study

- Reduced Ordered Binary Decision Diagrams: a fundamental data structure in model checking
- ROBDDs are produced through the repeated application of:
  - Redundant test elimination
  - Equivalent sub-graph sharing
- We investigated the application characteristics of a popular BDD package
  - BuDDy package version 2.2
  - Compiled with Intel –O3 compiler
  - Intel P4-2.4 GHz using VTune
  - BuDDy test cases for model-checking
Model Checking – Results

- As expected, it was almost completely memory bound
  - 80% of time was spent on cache misses
  - CPI was 6.5
  - Read bus utilization: 8.38%
- To find equivalent nodes, code hashes all nodes into a large hash table
  - Table is too large to fit into the cache, and accesses are random

![Processor time chart](chart.png)
Theorem Provers

- Inductive and deductive verification techniques
- PROS: No state explosion
- CONS: It’s hard! (And requires more programmer intervention)
Case Study: ACL2

- Developed in 1989 at UT-Austin and AMD
- Shares ancestry with Stanford’s PVS
- Written in Common Lisp
TLP Opportunities in ACL2

Speculative Multithreading

Bar1

Bar2

TLP

Bazz

Make a change
Hardware support for bug detection (Oplinger & Lam 2002)

- Hardware support for fine-grained transactions
  - Software marks beginning of transaction
  - All further side-effects (memory and register) are buffered
  - Software decides when to either commit or abort the transaction

- Use Thread Level Speculation to parallelize monitoring code
  - Very effective because monitoring code is typically independent from original code
Procedural Thread-level Speculation (TLS)
Procedural Thread-level Speculation (TLS)

need data
dependence
checking

NORMAL SEQUENTIAL EXECUTION

fork

fork
Procedural Thread-level Speculation (TLS)
Procedural Thread-level Speculation (TLS)
Procedural Thread-level Speculation (TLS)

unobserved data dependence

re-execute
Using TLS to speed up Monitoring

![Diagram showing the process of using TLS to monitor and execute code parallelism between monitoring and original code.]

hopefully significant parallelism between monitoring and original code
Using TLS to speed up Heavy Monitoring

here, need independence between monitoring code invocations to get decent speedup
Future Directions

- Right now, performance not critical (or they’d be multithreading already!)
- As models to be verified get more complex...
- As verification programs get smarter...
Summary

- Largely memory bound
  - Ratio of memory to arithmetic operations is large
  - Little to no locality
    - Pre-fetching might be effective
- Good opportunities for exploiting TLP
- Currently, research on methods of reduction probably more important than exploiting hardware
- Complexity of verification systems will scale with growing complexity of systems to be analyzed