UNIVERSITY OF TWENTE.

SHARED HASH TABLES IN PARALLEL MODEL CHECKING

IPA LENTEDAGEN 2010

ALFONS LAARMAN

JOINT WORK WITH MICHAEL WEBER

AND JACO VAN DE POL

23/4/2010
AGENDA

- Introduction
  - Goal and motivation
  - What is model checking?
  - Hash tables
- Related work
- Lockless hash table
- Experiments
GOAL AND MOTIVATION

Goal:
- Realize efficient multi-core reachability

Motivation:
- Multi-core is a necessity
- Reachability is a the basis of many verification problems
- Current model checkers do not scale as good as possible
- If you cannot parallelize reachability efficiently, then how do you parallelize more complicated algorithms:
  - full LTL model checking, symbolic reachability, POR, etc
WHAT IS MODEL CHECKING?

Shared Hash Tables in Parallel Model Checking  23/4/2010  3
### WHAT IS MODEL CHECKING?

<table>
<thead>
<tr>
<th>Process₁</th>
<th>Process₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( x \leftarrow x + 1; )</td>
<td>1 ( y \leftarrow y + 1; )</td>
</tr>
<tr>
<td>2 \textbf{if} ( y == 1 ) \textbf{then}</td>
<td>2 \textbf{if} ( x == 1 ) \textbf{then}</td>
</tr>
<tr>
<td>3 ( x \leftarrow 10; )</td>
<td>3 ( y \leftarrow 10; )</td>
</tr>
<tr>
<td>4 \textbf{end}</td>
<td>4 \textbf{end}</td>
</tr>
</tbody>
</table>

- PROMELA (SPIN)
- DVE (DiVinE)
- .NET (MoonWalker)
- C/C++ (terminator)
- Process algebraic (mCRL 1/2)
- Timed (UPPAAL)
- Hardware model checkers

---

*State space explosion*
WHAT IS MODEL CHECKING?

Data: Buffer $T = \{s_0\}$, Set $V = \emptyset$

1. while state $\leftarrow T$.get() do
2.     count $\leftarrow 0$;
3.     for succ in next-state(state) do
4.         count $\leftarrow$ count + 1;
5.         if $V$.find-or-put(succ) then
6.             $T$.put(succ);
7.         end
8.     end
9.     if 0 == count then
10.         // DEADLOCK, print trace..
11.     end
12. end

- States are arrays with variables
- Search in state space graph
- $V$ stores all seen states
- DFS or BFS depending on $T$
- Deadlocks and invariants

Parallelization:
- High throughput
- Synchronization points
HASH TABLES

A key is associated with data by using its hash as an index in a table

Hash collisions:
• Create overflow list (chaining) ← large memory working set
• Continue probing (open addressing) ← asymptotic behavior when full
  • Linear probing, double hashing
RELATED WORK

Fast model checkers:

- SPIN
- DiVinE 2.2
- DiVinE with Shared storage
RELATED WORK

DiVinE 2.2: static partitioning
BFS only, high comm. Costs, static load balancing

SPIN 5.2.4: shared storage + stack slicing
DFS only, multiple sync. points, specific case of load balancing
Barnat, Ročkai (2007) Shared hash tables in parallel model checking

- “Shared hash tables do not scale beyond 8 cores”
- “Could not investigate lockless hash table solution”

- Flexible reachability algorithm
- Flexible load-balancing
LOCKLESS HASH TABLE

- Design
- Lockless hash table
- Load balancing
Investigate requirements on shared storage
Investigate hardware (cache behavior)

Requirements
Find-or-put operation only
Scale by keeping a low memory working set
No pointers, no allocation
No resize!

Statically sized state vectors
LOCKLESS HASH TABLE

- Open addressing
- Hash memoization
- Separate data
- Walking the line
- Lockless (CAS + write bit)

See also Cliff Click JavaOne talk (2007)
LOCKLESS HASH TABLE

Data: size, Bucket[size], Data[size]
input : vector
output: seen

1) num ← 1;
2) hash_memo ← hashnum(vector);
3) index ← hash mod size;
4) while true do
   5) for i in walkTheLineFrom(index) do
      6) if empty = Bucket[i] then
         7) if CAS(Bucket[i], empty, ⟨hash_memo, write⟩) then
             8) Data[i] ← vector;
             9) Bucket[i] ← ⟨hash, done⟩;
             10) return false;
         11) end
      12) end
      13) if hash_memo = Bucket[i] then
          14) while ⟨-, write⟩ = Bucket[i] do ...wait... done
          15) if ⟨-, done⟩ = Bucket[i] ∧ Data[i] = vector then
              16) return true;
          17) end
      18) end
      19) num ← num + 1;
      20) index ← hashnum(vector) mod size;
   21) end
22) end

Walk-the-line
Linear probing

Wait for write in data array to complete

Double hashing
LOAD BALANCING

Static load-balancing
Workers can run out of work
Work stealing/handoff
Synchronized random polling

[Sanders97]
SUMMARY

As a summary, we implemented:

• The lockless hash table (in C)
• Reachability DFS + BFS
• Static load-balancing
• Synchronized random polling

Reused DiVinE next-state function
EXPERIMENTS

SETUP

- Using CMS 16-way AMD Opteron cluster
- linux 2.6.18, 2.6.32+patch
- 30+ models from BEEM database
  - Translated models for SPIN (same state count!)
- Statically sized hash tables
- Fair

Results →
Example, all model checkers, 3 models:
EXPERIMENTS

DiVinE 2.2:

Shared Hash Tables in Parallel Model Checking  23/4/2010  18
EXPERIMENTS

SPIN 5.2.4:

Shared Hash Tables in Parallel Model Checking  23/4/2010  19
EXPERIMENTS

LTSmin (lockless hash table):

Shared Hash Tables in Parallel Model Checking  23/4/2010  20
EXPERIMENTS

![Graphs showing time and speedup vs. number of cores for different tools: divine, hmin, and spin. The graphs demonstrate the performance improvement as the number of cores increases.]
EXPERIMENTS

Shared Hash Tables in Parallel Model Checking  23/4/2010  22
DISCUSSION / LIMITATIONS

• Statically sized states
• Reachability as a basis
• Slower sequentially
• Not 100% lock-free, but also not necessary
CONCLUSIONS

- Adapt applications to hardware (memory hierarchy)
- Centralized state storage scales better and is more flexible
- Scalable explicit exploration is a good starting point for future work on multi-core X. X ∈ {(weak) LTL model checking, symbolic exploration, space-efficient explicit exploration}

- Holzmann’s conjectures:
  - works only for unoptimized sequential code
  - works only for small state vectors/long transition delays
  SPIN has many features though, but backwards compatibility seems wrong starting point for scalable multi-core algorithms

- HTTP://FMT.CS.UTWENTE.NL/TOOLS/LTSMIN/
LTSMIN BFS SPEEDUPS  BASE CASE: LTSMIN BFS

Shared Hashtables in Parallel Model Checking  23/4/2010  25
LTSMIN DFS SPEEDUPS

BASE CASE: LTSMIN DFS

Shared Hashtables in Parallel Model Checking