InstantCheck: Checking the determinism of parallel programs using on-the-fly incremental hashing

Adrian Nistor, Darko Marinov, Josep Torrellas

MICRO `10
Problem: Nondeterminism

- Parallel programming is difficult
  - Threads race over synchronization and data
  - One fixed input \(\rightarrow\) many different outputs
  - Difficult to test, maintain, debug
  - Less likely to be reused
- The problem: Nondeterminism
  - *Uncertain* of what your own code does
- If possible, programmers prefer determinism
  - One input \(\rightarrow\) one output — like serial code
One Solution: Internal Determinism

• **Fix** the order of inter-thread communication
  – E.g.: grab locks in a predefined order
• Very intuitive for programmer \(\rightarrow\) serial code
• Inherently **costly**
  – High runtime overhead
    • Instrumentation, load-imbalance
  – Significant hardware extensions
  – New languages that restrict expressiveness
• More **relaxed** forms of determinism?
External Determinism

• **Relaxed** form of deterministic execution
• **Idea:** Same input -> same output **MEM state**
• Program executes/communicates any way it wants
  – No fixing inter – thread communication order
    • No performance hits
  – Little additional hardware, or none at all
  – Regular C code
• **Our current proposal:** **InstantCheck**
  – Checks if a program is externally deterministic
  – Does not enforce external determinism (future work ?)
**Example: Internal vs External Det.**

<table>
<thead>
<tr>
<th>CODE</th>
<th>EXEC—1</th>
<th>EXEC—2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>G = 2</td>
<td>G = 2</td>
</tr>
<tr>
<td></td>
<td>thread 0</td>
<td>thread 0</td>
</tr>
<tr>
<td></td>
<td>thread 1</td>
<td>thread 1</td>
</tr>
<tr>
<td></td>
<td>L_0 = 4</td>
<td>L_0 = 4</td>
</tr>
<tr>
<td></td>
<td>L_1 = 3</td>
<td>L_1 = 3</td>
</tr>
<tr>
<td>LOCK</td>
<td>G = 2 * 4</td>
<td>G = 2 * 3</td>
</tr>
<tr>
<td></td>
<td>G = G * L</td>
<td></td>
</tr>
<tr>
<td>UNLOCK</td>
<td>G = 8 * 3</td>
<td>G = 4 * 6</td>
</tr>
<tr>
<td></td>
<td>G = 24</td>
<td>G = 24</td>
</tr>
</tbody>
</table>

- Same state → External determinism
- Order really does not matter

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
InstantCheck

• Checks external determinism
• Very fast checking of memory–state equality
  – On-the-fly incremental hashing
  – Extremely small and simple hardware hashing module
• Run program multiple times during testing
  – Check if executions end up in deterministic state
  – Check several times during execution:
    • Program end, barriers, other points in the program
  – More checks → higher confidence
• Piggy–back on the testing done for parallel code
  – Such testing already runs the program many times
  – Simply insert very fast checks for external determinism

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Example: InstantCheck

- Run program many times
- Start from same state S0
- Fix the input
- In the end ?? S1 = ... = S77 ?? → yes → External Determinism
- Do we actually run 77 times ? NO!! → 2 – 3 times is enough
- Turns out that some programs even go through deterministic states \((T_1, T_2, T_3)\)

EXEC—1

\[
\begin{array}{c}
S_0 \\
S_1
\end{array}
\]

EXEC—2

\[
\begin{array}{c}
S_0 \\
S_2
\end{array}
\]

EXEC—77

\[
\begin{array}{c}
S_0 \\
S_{77}
\end{array}
\]

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Overview

Capturing program state

HW & SW System

Evaluation

Conclusions
Hashes – Efficient State Comparison

• A hash distills the memory state in just **64 bits**
• Efficient to compute, compare and store
• Efficiency does not come at the cost of accuracy
• False positive:
  – Same mem states, different hash → **impossible**
• False negatives:
  – Different mem states, same hash → **statistically rare**
  
  (for 64 bit hash, probability is ~~ 1 / 2^64 )
On-the-fly Incremental Hashing

- Compute the hash incrementally in hardware
  - On-the-fly at every store operation
- Each thread has a Hash Register
- When a thread executes STORE, it updates its Hash Reg:
  \[ \text{HashReg} = \text{HashReg} \quad \oplus \quad H(\text{old value}) \quad \oplus \quad H(\text{new value}) \]
- Consolidate the hashes of all the threads in software
  - End of program
  - Whenever you want to check determinism (e.g., barriers)
- The consolidated hash summarizes the memory state
  - No need to traverse the entire memory
How It Works

EXEC—1

Thread Hash

thread 0

TH_0 = 0

\[ G = 2 \]

\[ G = 2 \times 4 \]

\[ TH_0 = TH_0 \oplus h(G, 2) \oplus h(G, 8) \]

thread 1

TH_1 = 0

\[ G = 8 \times 3 \]

\[ TH_1 = TH_1 \oplus h(G, 8) \oplus h(G, 24) \]

State Hash combines the THs

\[ SH_0 = TH_0 \oplus TH_1 = \oplus h(G, 2) \oplus h(G, 24) \]

Hash Address & Value

MINUS old

PLUS new

CANCEL each other

MINUS initial

PLUS final

INDEPENDENT of intermediate values

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
How It Works

**EXEC—1**

- $G = 2$
- thread 0
- $G = 2 * 4$
- $TH_0 = TH_0 \oplus h(G, 2) \oplus h(G, 8)$
- $G = 8 * 3$
- $TH_1 = TH_1 \oplus h(G, 8) \oplus h(G, 24)$
- $SH_0 = TH_0 \oplus TH_1 = \oplus h(G, 2) \oplus h(G, 24)$

**EXEC—2**

- $G = 2$
- thread 0
- $G = 2 * 3$
- $TH_1 = TH_1 \oplus h(G, 2) \oplus h(G, 6)$
- $G = 4 * 6$
- $TH_0 = TH_0 \oplus h(G, 6) \oplus h(G, 24)$
- $SH_0 = TH_0 \oplus TH_1 = \oplus h(G, 2) \oplus h(G, 24)$

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Issues and Questions

• Why do we hash the address?
  — HashReg = HashReg ⊕ H(G, 2) ⊕ H(G, 8)
  — Removes ambiguity: two addresses, same values

• Why does it work?
  — Modulo addition is associative and commutative
  — Modulo subtraction cancels modulo addition

• If needed, how do we delete some variables?
  ⊕ H(initial value) ⊕ H(final value)

• False positives? → impossible

• False negatives? → statistically rare
Overview

Capturing program state

 HW & SW System

Evaluation

Conclusions

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Hardware System – Basic Design

- Hardware extension in the L1 cache controller

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Flexible Implementation Choices

- Commutative & Associative
  - In parallel
  - Out-of-order
- Optimize for area or speed
- Clusters compute:
  - Intermediate results
- Sum intermediate results

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Lightweight Hardware

- Migrate
- Virtualize
- Context-switch
- Scalable
- Flexible implementation choices

1 register: save & restore
Core – local operations
Checking determinism with on-the-fly incremental hashing
Sources of Nondeterminism (1)

- We may want to detect, control, or ignore some special sources of nondeterminism
- Software bugs
  - Can detect them if they create nondeterminism
  - Concurrency bugs do create nondeterminism
  - We found a real bug in PARSEC
Sources of Nondeterminism (2)

• Floating point precision limitations
  \[(A \times B) \times C \neq A \times (B \times C)\]

• Maybe the user wants to ignore this
  – Include rounding unit in the Hashing Module
  – We offer several options
  – By default: round to the nearest 0.001
  – In our experiments (many FP) this was effective
Sources of Nondeterminism (3)

- Malloc(), rand(), get_time_of_day()
  - Solution: control them
- Small auxiliary data structures
  - Cholesky: size and order of free task nodes list
  - Solution: delete from hash with $\oplus$ / $\ominus$ technique
- Truly nondeterministic algorithms
  - E.g.: simulate annealing

$\rightarrow$ Many programs are deterministic
Overview
Capturing program state
HW & SW System
Evaluation
Conclusions
Experimental Setup

• Simulate HW and SW implementations
• Simulate the HW Hashing Module with PIN
• 17 applications:
  – Sphinx3, PBZip2, PARSEC, SPLASH–2
• Run each application 30 times
• Compare states at program end, barriers, end of loop iteration (2 applications)
• Randomizing thread scheduler
  – Similar to CHESS, PCT from Microsoft Research
## Characterizing Determinism

<table>
<thead>
<tr>
<th>Type of Determinism</th>
<th># of Apps</th>
<th>Application</th>
<th>First NDet Run</th>
<th># Dyn Checking Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit-by-bit deterministic</td>
<td>7</td>
<td>streamcluster</td>
<td>—</td>
<td>12928</td>
</tr>
<tr>
<td>Deterministic if FP rounding</td>
<td>4</td>
<td>fluidanimate</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>Deterministic if ignoring small structures</td>
<td>3</td>
<td>cholesky</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Nondeterministic</td>
<td>3</td>
<td>canneal</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>17</strong></td>
<td><strong>2–3</strong></td>
<td><strong>0–10,000</strong></td>
<td><strong>0–70</strong></td>
</tr>
</tbody>
</table>

- Most programs are externally deterministic (14 out of 17)
  - Though they were not written for determinism (but for high performance)
- Nondeterminism is detected fast (2 – 3 runs)
- Determinism points even inside the program, not just at the end
  - Checking determinism just at the end may miss bugs (e.g., streamcluster)
Comparing Different Implementations

- Native: no determinism checking
- HW-InstantCheck-Inc: discussed so far
- SW-InstantCheck-Inc
  - Incremental hashing in software
- SW-InstantCheck-Tr
  - Traverse the data structures and hash in software

→ Compare the number of instructions
Total Instructions Executed

- HW scheme has negligible overhead → always-on
- SW schemes are reasonable alternatives:
  - Geometric Mean: 3X or 5X
  - For some apps, one SW scheme clearly better than the other → choose

Adrian Nistor  Checking determinism with on-the-fly incremental hashing
Overview
Capturing program state
HW & SW System
Evaluation
Conclusions
Hardware Hashing — Discussion

• Versatile and lightweight primitive for programmability

• Other uses of fast on-the-fly state hashing:
  – Detecting bugs
  – Prune the search space in systematic testing
  – Filtering out benign data races
  – Deterministic replay

• Other uses of hardware hashing:
  – Record execution history rather than execution state
  – Hashes read(data) rather than write(data, addr)
  – Detect data races during systematic testing
Conclusions: InstantCheck

- Technique to check external determinism
- Lightweight HW primitive:
  - On-the-fly incremental hashing
  - Instantly compare memory states
- Advantages of checking external determinism:
  - Negligible execution overhead
  - No extensive HW that limits the communication
  - Does not require new specialized languages
- Analyzed determinism characteristics of existing apps
  - 14 out of 17 are externally deterministic
- Found a bug in PARSEC
- The HW primitive has several other uses
Characterizing Determinism

| Det Type | Application | Source | FP? | Det as is? | First NDet Run | Impact of FP rounding on Det. | First NDet Run after FP | Isolating Small Structures on Det. | # Dyn Checking Points | Det at the End |
|----------|-------------|--------|-----|------------|----------------|-----------------------------|-------------------------|-------------------------------|----------------------|----------------|----------------|
| bit by bit | blackscholes ft | parsec | Y | Y | – | Det $\rightarrow$ Det | – | – | 101 | Y |
|           | lu         | splash2 | Y | Y | – | Det $\rightarrow$ Det | – | – | 13 | Y |
|           | radix      | splash2 | Y | Y | – | Det $\rightarrow$ Det | – | – | 68 | Y |
|           | streamcluster | parsec | Y | Y | – | Det $\rightarrow$ Det | – | – | 12 | Y |
|           | swaptions  | parsec | Y | Y | – | Det $\rightarrow$ Det | – | – | 12928 | Y |
|           | volrend    | splash2 | N | Y | – | Det $\rightarrow$ Det | – | – | 2501 | Y |
|          | fluidanimate | parsec | Y | N | 2 | NDet $\rightarrow$ Det | – | – | 6 | Y |
|          | ocean      | splash2 | Y | N | 3 | NDet $\rightarrow$ Det | – | – | 41 | Y |
|          | waterNS    | splash2 | Y | N | 3 | NDet $\rightarrow$ Det | – | – | 871 | Y |
|          | waterSP    | splash2 | Y | N | 2 | NDet $\rightarrow$ Det | – | – | 21 | Y |
|          | cholesky   | splash2 | Y | N | 3 | NDet $\rightarrow$ NDet | 3 | NDet $\rightarrow$ Det | 4 | Y |
|          | pbzip2     | openSrc | N | N | 2 | NDet $\rightarrow$ NDet | 3 | NDet $\rightarrow$ Det | 1 | Y |
|          | sphinx3    | alpBench | Y | N | 2 | NDet $\rightarrow$ NDet | 2 | NDet $\rightarrow$ Det | 4265 | Y |
| small struct | barnes | splash2 | Y | N | 2 | NDet $\rightarrow$ NDet | 2 | – | 2 | N |
|           | canneal    | parsec | N | N | 2 | NDet $\rightarrow$ NDet | 2 | – | 0 | N |
|           | radiosity  | splash2 | N | N | 2 | NDet $\rightarrow$ NDet | 2 | – | 0 | N |

Adrian Nistor  
Checking determinism with on-the-fly incremental hashing