Deterministic Parallel Programming

Concepts and Practices

Li Lu
04/2011
How hard is parallel programming

• What’s the result of this program?
• What is data race?
• Should data races be allowed?

Initially $x = 0$

Thread 1: $x = 1$
Thread 2: Output $x$
How hard is parallel programming

• What’s the result of this program?

• Still data races?

• Is this program “correct”?

Thread 1:
lock
x = 1
unlock

Thread 2:
lock
x = 2
unlock
Output x
How about debugging a parallel program?

• In debug, we usually...
  • Check variable values at break points
  • Check register values for each processor
  • Monitoring certain variable’s values
• Does scheduling matters?

```plaintext
parallel for i in [1..100]
f(A[i])
```
Deterministic Parallel Programming

- General sense: behaves “in the same way”
- Easier to...
  - Construct and debug
  - Understand and maintain
- Scenario 1: Debug a parallel program
- Scenario 2: Verify an existing program
- Scenario 3: Estimate time/space consumption
What’s following?

• What is deterministic parallel programming
  • A formalized framework
  • Multiple interesting definitions

• Any implementations?
  • Deterministic Programming Languages
  • Deterministic Implementations
What is deterministic parallel programming?

We need a formal definition.
Possible Definitions

• Computes the same output on a given input
• Terminates in the same state
• Computes in the same way
  • same time & space consumption
  • same visible states in debugger
  • as easy as sequential code to test and understand
• What exactly does “in the same way” mean?
Semantics & Implementations

Source Program and Input

Language semantics

Program Executions

Language implementation

Target Executions

Correspond to
Executions

• A 3-tuple $E$: ($OP$, $<_p$, $<_s$)
  • $OP$: set of operations ($op_{name}$, $val_{list}$, $tid$)
  • $<_p$: Program order
  • $<_s$: Synchronization order
  • $<_{hb}$: Irreflexive transitive closure of $<_p$ and $<_s$
  • Input & output: external events, ordered by $<_s$
  • $ext(E)$: the sequence of external operations in $E$, without their thread ids
Proposed Framework

- A *program* is deterministic iff all of its program executions on a given input are *equivalent*.
- A *language* is deterministic iff all its programs are deterministic.
- An *implementation* is deterministic iff all the target executions of a given program on a given input correspond to program executions that are mutually *equivalent*.
Example definitions of equivalence
What’s Following?

• Provide and analyze 5 possible definitions
• All definitions guarantee the same output on the same input
• Assumptions
  • I/O operations operate on two unbounded vectors
  • I/O operations are fully synchronized
Singleton

• Two executions $E_1$ and $E_2$ are equivalent iff they are exactly the same

• Strictest definition of equivalence

• Rules out “benign” differences of any kind among program executions
Singleton

• Good for repetitive debugging
• Possible to guarantee same state when rerunning to a breakpoint
• Monitored variables change deterministically
• Possible idiom: Independent split-merge
• Access disjoint sets of variables
• e.g., with parallel iterators or cobegin
Dataflow

- Two executions $E_1$ and $E_2$ are equivalent iff there is a bijection between their set of operations that preserves the dataflow:
  - the content other than thread id in each operation
  - the values each read returns
Dataflow

- Still good for repetitive debugging
  - Same values for same data watchpoints
  - Two executions may not reach the same global state when a breakpoint is triggered
- Without special purpose hardware, scalable performance for general programs may be difficult
- Idiom: bag of independent tasks
**SyncOrder**

- Two executions $E1$ and $E2$ are equivalent iff they synchronize in the same way.
- For example: grab a lock, send/recv or resolve conflicting transactions.
SyncOrder

- Allows benign changes in data flow
- Not as good for repetitive debugging
  - No guarantee on global state at breakpoints
  - No guarantee on variable value changes
- Potentially cheaper to implement
  - instrument only synchronization ops
- Idiom: chaotic (racy) convergence
**ExternalEvents**

- Two executions E1 and E2 are equivalent iff they have the same output on the same input.
- The minimal definition of determinism?
- Not very good for repetitive debugging.
- Implementation cost is hard to predict.
- Maximum flexibility for language design and implementation.
- Possible idiom: parallel iterator with reduction.
FinalState

• Two executions are equivalent iff they both terminate and terminate with the same state

• A variant of ExternalEvents

• Cannot handle non-terminating programs

• Disadvantages similar to ExternalEvents
benign synchronization races: e.g., bag of tasks

benign non-trivial data races: e.g., racy convergence

e.g., parallel loop with reduction
Future Work

• Formalize definitions and proofs
• Consider additional definitions, connect them with programming idioms
• Accommodate conditional synchronization, spinning, operations like `rand`
• Consider deterministic implementations
• Implement and assess representative languages
What’s following?

• What is deterministic parallel programming
• A formalized framework
• Multiple interesting definitions

• Any implementations?
• Deterministic Programming Languages
• Deterministic Implementations
Deterministic Programming Languages

Source Program and Input

Target Executions

Language implementation

Language semantics

Correspond to

Program Executions
Deterministic Parallel Java

- Based on original Java language
- A changed type and effect system
  - Regions: Partitioning memory locations for disjoint constraints
  - Effects list: annotated memory effects
- Expressing parallelism: cobegin and foreach
- Not so easy to use...
Intel CnC

- A completely new parallel programming model
  - Data flow style programming
  - Programmer specifies restrictions
  - Data and control dependencies
  - With built-in collections (item, tag, step)*
  - Parallelism is hidden in collections’ semantics
- Not strictly deterministic... (Until now)
Deterministic Implementations

Source Program and Input

Language semantics

Program Executions

Target Executions

Language implementation

Correspond to
Examples

• Pure software implementations
• Determinator: OS support (special region and APIs)
• CoreDet: Compiler and runtime extension
• Hardware-software hybrid
• DMP and RCDC: architectural supported determinism for various memory models
• Problem: slow or expensive
One more thing...

• Deterministic *replay*
  • Different to deterministic *executions*

• Different focus
  • Deterministic executions: always produce equivalent executions
  • Deterministic replays: produce exactly the same execution as the recorded one
Thank you!

Q&A