Turning Nondeterminism into Parallelism

Omer Tripp
Tel Aviv University
Israel

Eric Koskinen
New York University
USA

Mooly Sagiv
Tel Aviv University
Israel

Turning Nondeterminism into Parallelism

- Nondeterminism and Parallelism
- Approach of **`TANGO`**
- Possible Applications / Restrictions
- Progress and Preservation
- Benchmarks

Nondeterminism and Parallelism

- Tasks have multiple possible behaviours
- Hard to parallelize due to possible conflicts
- Optimistic and pessimistic approaches fails
- Every method of task 1 must commute with each of task 2
- But nondeterminism has hidden parallelism

Task 1	**Thread 2**

`...`
`Iterator it=elems.iterator();`
`Object o1=it.next();`
`...`
	`...`
	`Object o2=...;`
	`elems.remove(o2);`
	`...`

TANGO's Approach

- Tool for loop parallelization for nondeterministic programs
- Static may-modify analysis at compile time (WALA)
  - ∀ iterations (task) it stores a link to the affected substate
  - Available as **may-modify oracle** during runtime

    → Consider possible future behaviours rather than history
- **Specialization** (select mutually compatible behaviour)
  - Tasks store their modifications in a log (except for the task with highest priority)
  - Guards (property checks) are evaluated over a substate
- Very easy to use: put `Monotonic` and replace ADT

Possible Applications / Restrictions

- Algorithms in which each iteration tests for some property (query, guard) and modifies (subset of) data if property holds.
- Guards: **nondeterministic**, **pure** and **monotone**
- Tasks: intentions have to be **predictable**, constrainted and must have a compact representation.
- Working set of a task is a small subset of shared state
- Various ADTs supported (Map, Graph, Tree, Set, …)
- IBM App Scan (web vulnerability detector)
  - Identifies injection points and tries to exploit them
  - Greedy pruning with nondet. monotonic guard

Guaranteed Preservation

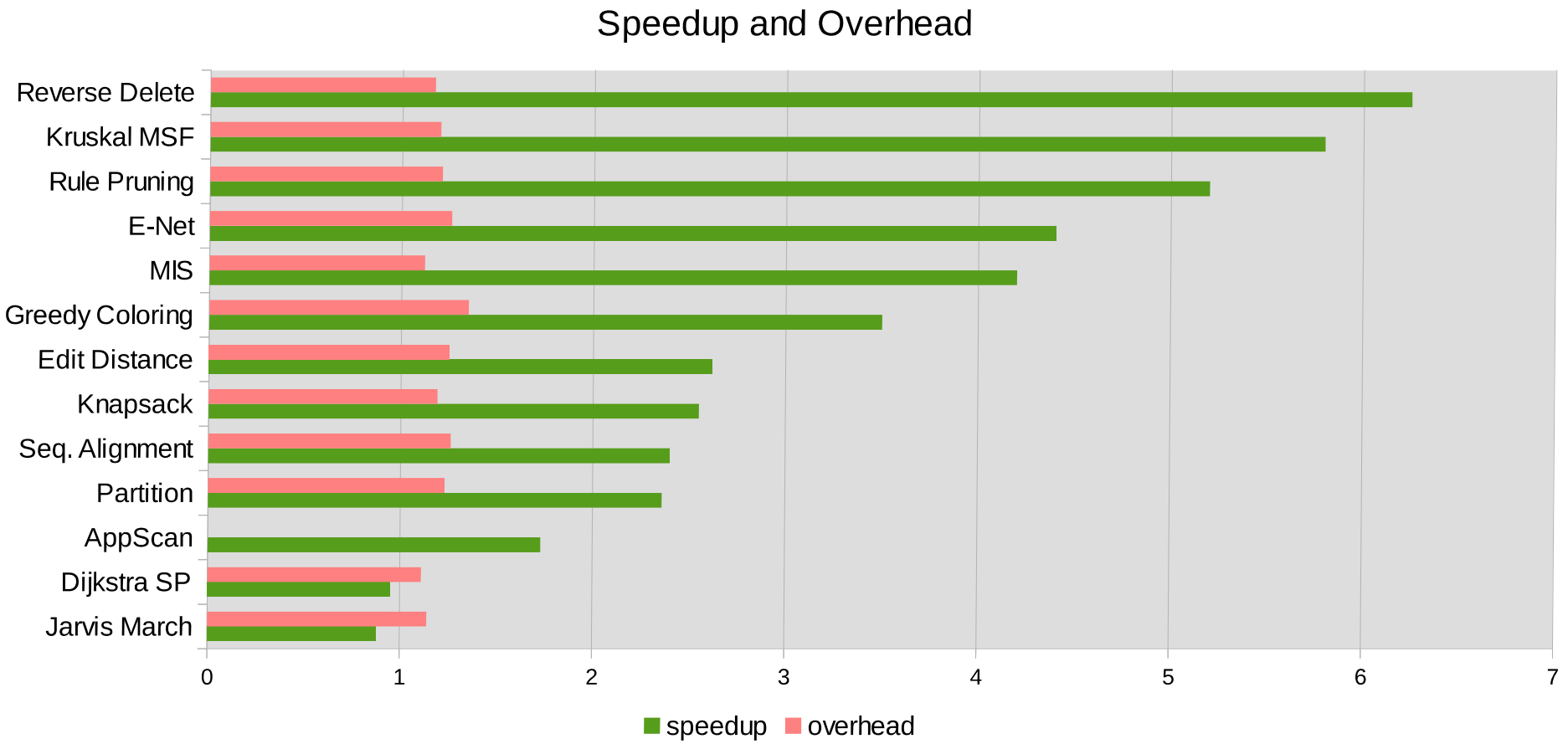
- **Theorem:** Method and guard specialization **preserves behaviour and serializability** of the original system.
- Proof sketch:
  - For methods: specialization causes less states, but still all needed.
  - For guards: evaluates to true on substate $\rightarrow$ also evaluates to true on entire state (monotonicity)
  - Local preservation $\rightarrow$ global preservation

Guaranteed Progress

- Prioritize tasks to guarantee progress
  - Breaks symmetrical dependency
- **Theorem:** For tasks* t1, t2, ..., tn and priority relation (total order) << either:
  - a) all tasks have completed or else
  - b) ∃ ti (with highest priority) that can perform a method or guard

*each of them has to terminate if executed sequentially

Benchmarks for **TANGO**



Speedup with **16 threads**
Overhead with **1 thread**

Questions