RAMSES: Reversibility-based Agent Modeling and Simulation Environment with Speculation-support

Davide Cingolani
Alessandro Pellegrini
Francesco Quaglia

High Performance and Dependable Computing Systems Group
Sapienza, University of Rome

PADABS 2015

What is RAMSES

- An x86 framework for agent-based discrete-event models
    - Clear distinction between *agents* and the *environment*
    - Models can be implemented according to the discrete event programming paradigm
    - No notion of parallelism in the actual model's code

- Parallel execution is based on speculative event processing
    - Synchronization is ensured via *software reversibility*
    - Reversibility is achieved via *software instrumentation*
    - Specifically targeting multi-/many-core architectures
    - Single queue shared among all threads

- Scheduling of events is *dominated by the environment*
    - An agent can act only when located in a given portion of the environment

Reference Programming Model

- The simulated phenomenon is decoupled only into two kinds of entities

- **Environment**
  - It can be of any size and shape, yet divided into regions
  - Each region must have a state, possibly scattered into the heap
  - The state can grow/shrink during the simulation run
  - At a given time, a region can host any number of agents
  - A region's state can be modified by an external event or by an agent
  - Regions can be connected to each other in any possible way

Reference Programming Model (2)

- **Agents**
  - Each agent must have a state, possibly scattered into the heap
  - The state can grow/shrink during the simulation run
  - Agents are always located in a region
  - Agents can move freely in the environment, according to the connection among regions
  - Agents can interact with each other, yet interactions can take place only if they are in the same region
  - Agents can interact with the environment — inspections and modifications must be supported

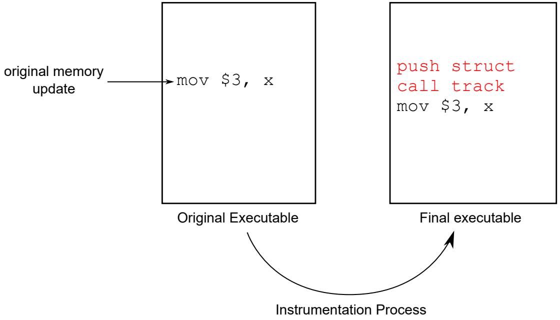
A glance at some of the API

- Functions to model the initial state of the model:
  - `Setup()`
  - `void InitialPosition()`
  - `void StartSimulation()`

- Functions to carry on the evolution of the system:
  - `void Move()`
  - `void AgentInteraction()`
  - `void EnvironmentInteraction()`
  - `void EnvironmentUpdate()`

Tracking Memory Updates for Reversibility

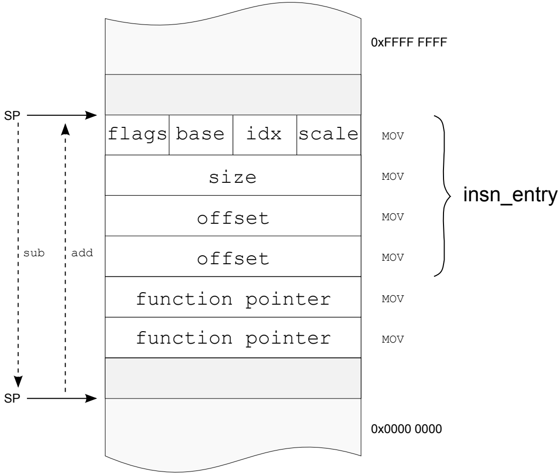
- Static software instrumentation is used to transparently modify the model's code
  - We rely on the Hijacker software instrumentation tool
- Memory updates are tracked to build packed versions of negative instructions



Tracking Memory Updates for Reversibility (2)

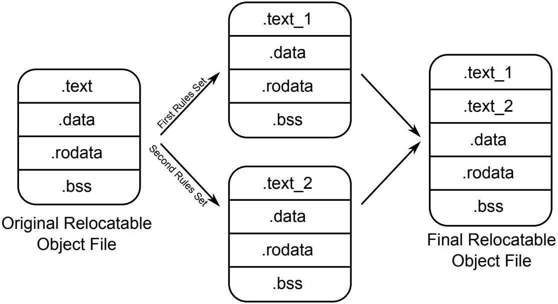
$$\left[\left\{\begin{matrix}\text{AX}\\\text{BX}\\\text{CX}\\\text{DX}\\\text{SP}\\\text{BP}\\\text{SI}\\\text{DI}\\\text{R8}\\\text{R9}\\\text{R10}\\\text{R11}\\\text{R12}\\\text{R13}\\\text{R14}\\\text{R15}\end{matrix}\right\}\right] + \left[\left\{\begin{matrix}\text{AX}\\\text{BX}\\\text{CX}\\\text{DX}\\\text{SP}\\\text{BP}\\\text{SI}\\\text{DI}\\\text{R8}\\\text{R9}\\\text{R10}\\\text{R11}\\\text{R12}\\\text{R13}\\\text{R14}\\\text{R15}\end{matrix}\right\} * \left\{\begin{matrix}1\\2\\4\\8\end{matrix}\right\}\right] + [\mathit{displacement}]$$

Tracking Memory Updates for Reversibility (3)

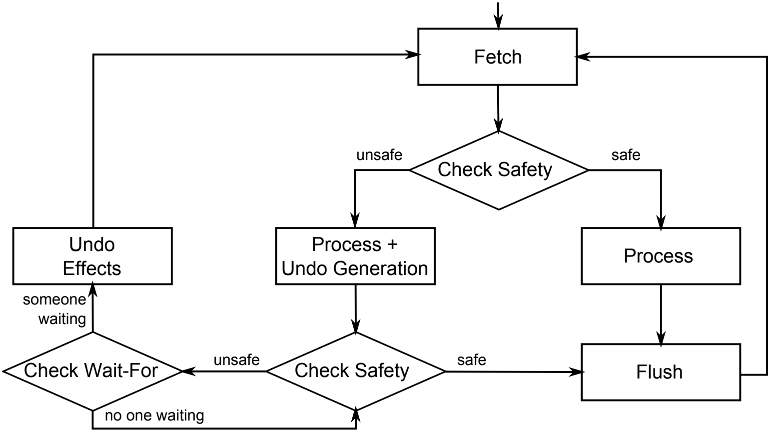


Tracking Memory Updates for Reversibility (4)

- It is not always necessary to pay the cost of generating reverse instructions
- We can then use *multi-coding* to quickly switch at runtime among instrumented/non-instrumented code



Runtime Execution Support



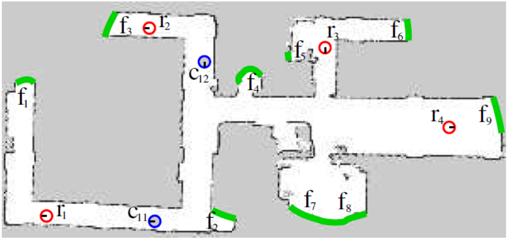
FETCH operation

```
procedure FETCH(last_event e) RETURNS: event
    if e = NULL then
        SPINLOCK(global_lock) //this branch is atomic via a globally shared lock
        e ← GETMINIMUMTIMESTAMPEVENTFROMCALENDARQUEUE( )
        processing[i] ← T(e)
        SPINUNLOCK(global_lock)
    end if
    if ¬TRYLOCK(region_lock[e.destination]) then
        repeat
            reupdateMin ← false
            minWait ← wait_time[e.destination]
            if T(e) < minWait then
                if ¬ CAS(wait_time[e.destination], minWait, T(e)) then
                    reupdateMin ← true
                end if
            end if
        until reupdateMin
        while TRUE do
            SPINLOCK(region_lock[e.destination])
```

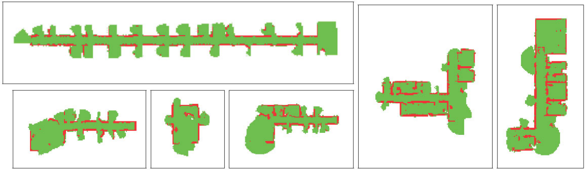
FETCH operation (2)

```
            if T(e) ≤ wait_time[e.destination] then break
            end if
            SpinUnlock(region_lock[e.destination])
        end while
    end if
    return e
end procedure
```

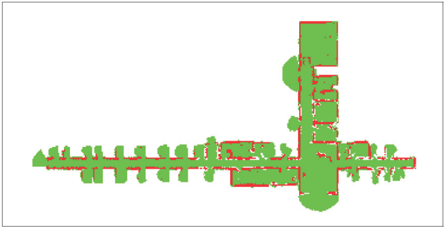
Performance Study: Distributed Multi-Robot Exploration and Mapping



Performance Study: Distributed Multi-Robot Exploration and Mapping



(a) input maps



(b) merged map

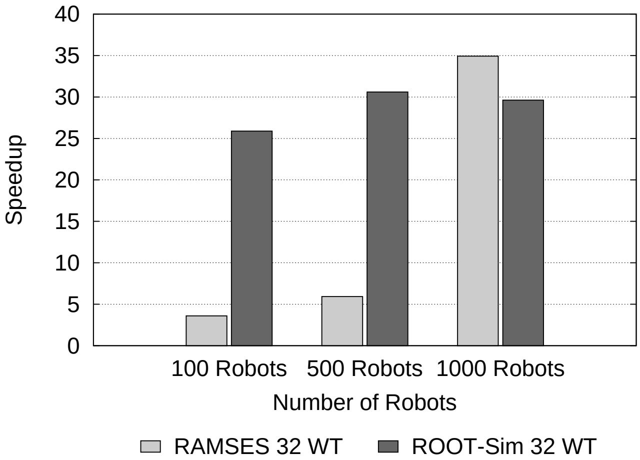
Performance Study: Distributed Multi-Robot Exploration and Mapping

- The map is constructed online
- Robots explore independently, until they accidentally meet:
  1. they use their sensors to estimate their mutual physical position
  2. they create a rendez-vous point to verify the estimation's goodness
  3. if the hypothesis is verified, they exchange the so-far acquired data
  4. they form a cluster
- Clusters allow to explore collaboratively:
  - jointly define the next targets (reduce mapping time)
  - make a guess on the position of other robots (enlarge the cluster)

Performance Evaluation: Set up

- Hardware configuration:
  - HP ProLiant server equipped with 64GB of RAM
  - 4 8-cores CPU (32 cores total)
- Benchmark configuration:
  - 4096 regions
  - Random obstacles
  - Variable number of robots between 100 and 1000
- Comparison with ROOT-Sim, a Time-Warp-based general purpose PDES simulation engine
  - in ROOT-Sim agents are not bound to regions
  - the implementation is more complex (50% more lines of code)

Experimental Results



Thanks for your attention

Questions?

pellegrini@dis.uniroma1.it
http://www.dis.uniroma1.it/∼pellegrini
http://github.com/HPDCS/RAMSES