A Symmetric Multi-threaded Architecture for Load-sharing
in Multi-core Optimistic Simulations

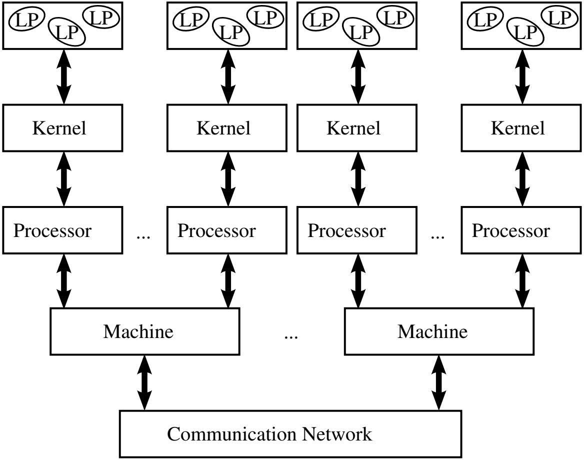
SAPIENZA
UNIVERSITÀ DI ROMA

Roberto Vitali
Alessandro Pellegrini
Francesco Quaglia

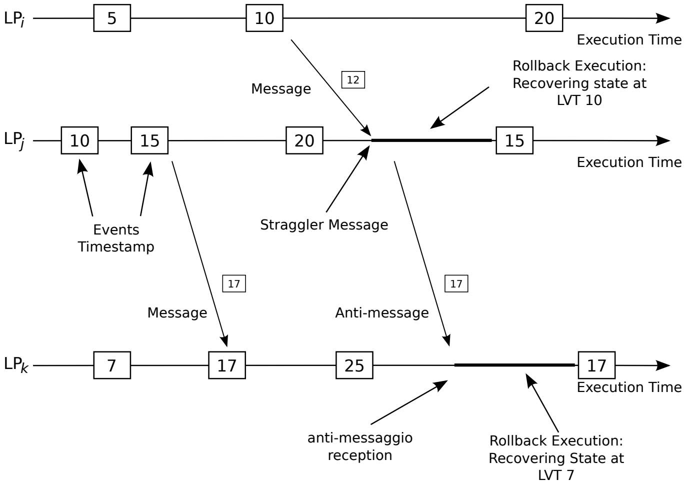
High Performance and Dependable
Computing Systems Group
Dipartimento di Ingegneria Informatica,
Automatica e Gestionale
Sapienza, University of Rome

InfQ - 2012

PDES Logical Architecture

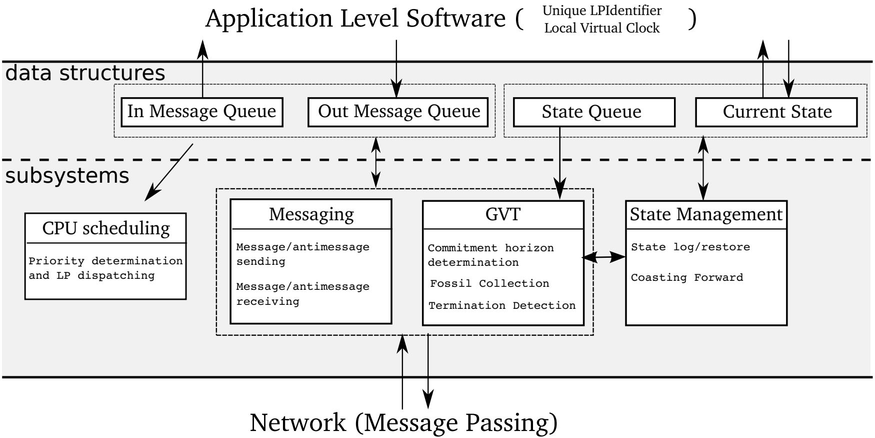


Rollback



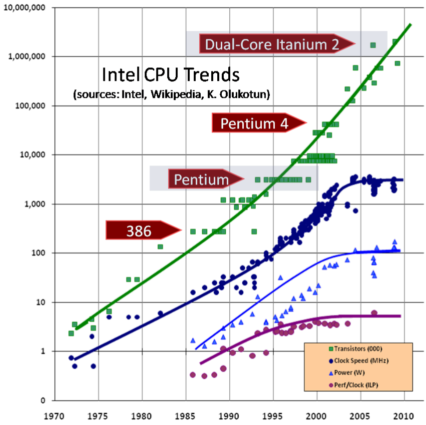
LP$_i$
5
10
20
Execution Time
Message
12
Rollback Execution:
Recovering state at
LVT 10
LP$_j$
10
15
20
15
Execution Time
Events
Timestamp
Straggler Message
17
Message
17
Anti-message
LP$_k$
7
17
25
17
Execution Time
anti-messaggio
reception
Rollback Execution:
Recovering State at
LVT 7

Time Warp Fundamentals



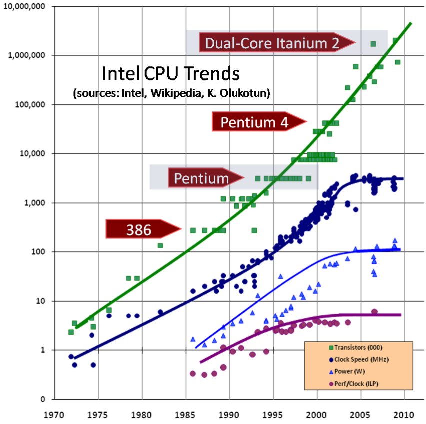
Rationale

- Processors speeds are no longer following Moore's Law
- Multi-core machines are the industry's answer to the increasing need in computational power



Rationale

- Processors speeds are no longer following Moore's Law
- Multi-core machines are the industry's answer to the increasing need in computational power
- Yet, parallelizing an optimistic simulation kernel entails a hard synchronization effort



Goals

- Propose a Paradigm Shift towards Symmetric Multi-threaded Optimistic Simulation Kernels
  - Reshuffle of their internal organization
  - Rely on the worker-thread paradigm to concurrently run any LP hosted by a given kernel instance

Goals

- Propose a Paradigm Shift towards Symmetric Multi-threaded Optimistic Simulation Kernels
  - Reshuffle of their internal organization
  - Rely on the worker-thread paradigm to concurrently run any LP hosted by a given kernel instance
- Exploit this new organization to support load sharing
  - Orthogonal to load balancing
  - Computational power is reassigned to kernel instances
  - Any kernel instance can activate/deactivate a certain number of worker threads

Kernel-Level Synchronization

- Avoid *lock-everything effects* in kernel mode
  - Reduced set of operations/data structures
  - Inherent strict coupling among the LPs

Kernel-Level Synchronization

- Avoid *lock-everything effects* in kernel mode
  - Reduced set of operations/data structures
  - Inherent strict coupling among the LPs
- Frequent updates involve *input/output queues*
  - Core of the cross-LP dependencies
  - Updates by the worker thread currently running the owener LP
  - Additionally, by worker threads running other LPs

Kernel-Level Synchronization

- Avoid *lock-everything effects* in kernel mode
  - Reduced set of operations/data structures
  - Inherent strict coupling among the LPs
- Frequent updates involve *input/output queues*
  - Core of the cross-LP dependencies
  - Updates by the worker thread currently running the owener LP
  - Additionally, by worker threads running other LPs
- Critical sections' duration is dependent on actual time-complexity of the queue-update operation.

Top/Bottom Halves (1)

- Operations involving data-structures updates are logically considered as *interrupts*
- Upon the receival of an interrupt, the task is not immediately finalized
- A light (constant time) top-half module is executed, for registering the operation to be performed

Top/Bottom Halves (1)

- Operations involving data-structures updates are logically considered as *interrupts*
- Upon the receival of an interrupt, the task is not immediately finalized
- A light (constant time) top-half module is executed, for registering the operation to be performed

- Three situations can explicitly produce an interrupt:
  1. A worker thread is running a locally hosted LP_j in **forward** mode, which produces a new event to be scheduled for the locally hosted LP_i

Top/Bottom Halves (1)

- Operations involving data-structures updates are logically considered as *interrupts*
- Upon the receival of an interrupt, the task is not immediately finalized
- A light (constant time) top-half module is executed, for registering the operation to be performed

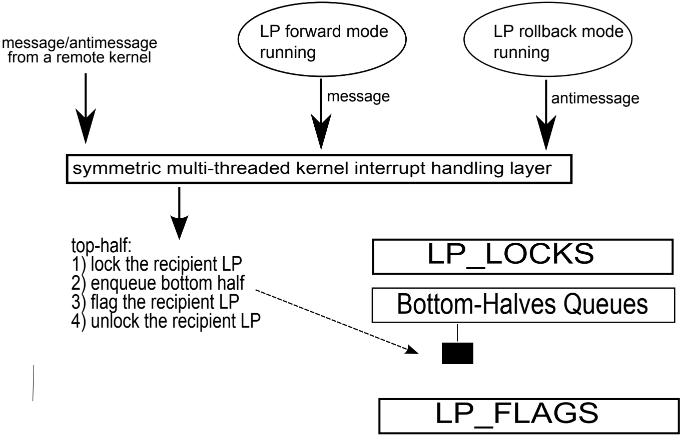
- Three situations can explicitly produce an interrupt:
  1. A worker thread is running a locally hosted LP_j in **forward** mode, which produces a new event to be scheduled for the locally hosted LP_i
  2. A worker thread is currently running a locally hosted LP_j in **rollback** mode, giving rise to the production of an antimessage destined to LP_i

Top/Bottom Halves (1)

- Operations involving data-structures updates are logically considered as *interrupts*
- Upon the receival of an interrupt, the task is not immediately finalized
- A light (constant time) top-half module is executed, for registering the operation to be performed

- Three situations can explicitly produce an interrupt:
  1. A worker thread is running a locally hosted LP_j in **forward** mode, which produces a new event to be scheduled for the locally hosted LP_i
  2. A worker thread is currently running a locally hosted LP_j in **rollback** mode, giving rise to the production of an antimessage destined to LP_i
  3. The message passing layer notifies the worker thread about a new message/antimessage incoming from some remote kernel instance.

Top/Bottom Halves (2)



Computational Power Reallocation Policy

- The symmetric multi-threaded kernel allows scaling up/down the amount of per-kernel worker threads.
- This feature allows for dynamically reallocating the computational power wrt the workload variations

C_{tot}	available CPU cores
$K_{tot}\,(\leq C_{tot})$	active symmetric kernel instances

Goal: determine C_i $(1 \leq C_i < K_{tot})$ $\forall K_i$ for a given wall-clock-time window, so to improve resource exploitation.

Computational Power Reallocation Policy (2)

Idea: Dynamically assign an amount of CPU-cores to kernel K_i which is proportional to the actual computation requirements of K_i for the achievement of its relative event rate, compared to the one by the other kernels.

$$wevr_i = \underbrace{evr_i}_{\text{current event rate}} \cdot \underbrace{\Delta_i}_{\text{average CPU time for processing events}}$$

The Power Reallocation follows the following steps:

$$\alpha_i = \frac{wevr_i}{\sum_{j=1}^{K_{tot}} wevr_j} \quad (1)$$

$$C_i = \lfloor \alpha_i \cdot C_{tot} \rfloor \quad (2)$$

Computational Power Reallocation Policy (3)

$$\forall K_i \text{ s.t. } C_i \geq numLP_i, \quad C_i = numLP_i \tag{3}$$

At this point, some CPU-cores might be unassigned yet, which we do on the basis of the request for allocation remainder:

$$r_i = [(\alpha_i \cdot C_{tot}) - C_i] \tag{4}$$

- We order the kernels for which the finalization of C_i values still needs to be performed according to decreasing values of the product $r_i \cdot wcta_i$
- We assign the remaining CPU-cores according to a round-robin rule following the priority defined by such an ordering

Binding LPs to Worker Threads

- A given set of LPs hosted by K_i is temporarily bind to a specific worker thread
- Once the new value for C_i is computed, the policy to determine which LPs are bind to a specific worker thread is:
  - For LP_j hosted by K_i we compute cpu_i^j, i.e. the *total amount of CPU-time* needed for committing its events during the last observation period
  - $\max_{i,j}\{cpu_i^j\}$ is considered as a reference knapsack
  - A modified greedy-approximation algorithm by George Dantzig for knapsack solution is executed

Implementation within ROOT-Sim

- ROOT-Sim is an open-source general-purpose C-based optimistic simulation platform
- The end-user can transparently rely on the ANSI-C set of programming facilities

http://www.dis.uniroma1.it/∼hpdcs/ROOT-Sim/

Implementation within ROOT-Sim

- ROOT-Sim is an open-source general-purpose C-based optimistic simulation platform
- The end-user can transparently rely on the ANSI-C set of programming facilities

http://www.dis.uniroma1.it/∼hpdcs/ROOT-Sim/

- Worker threads are implemented using `pthread` tehcnology
- Per-LP data structures are reshuffled:
  - per-thread private data (avoid synchronization efforts)
  - cache-aligned data structures, via `posix_memalign` and proper padding (avoid false cache-sharing)
- Accesses to MPI layer are synchronized via wrappers
- GVT reduction is carried out by one single thread
- Fossil collection is performed by all worker threads in parallel

Experimental Results

Hardware Setting

- 64-bit NUMA machine
- 32 2-GHz cores
- 64 GB of RAM

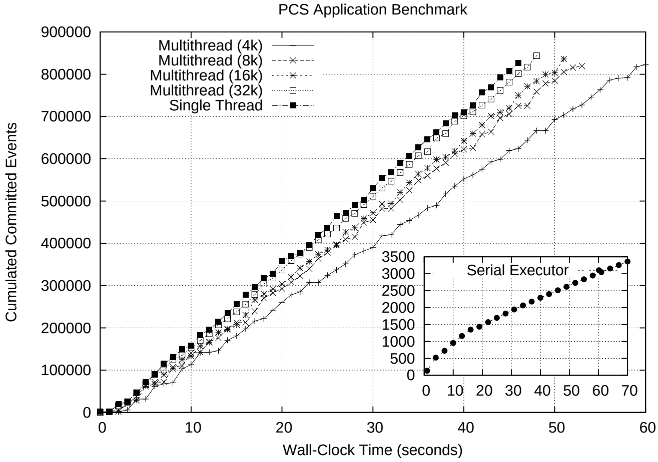
Personal Communication Service

- It implements a simulation model of GSM communication systems
- Channels are modeled in a high fidelity fashion

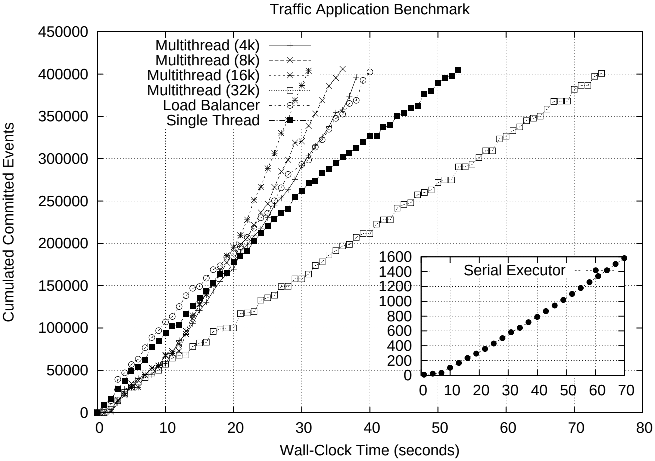
Traffic

- It simulates a complex highway system (at a single car granularity)
- The topology is a generic graph

Experimental Results (2)



Experimental Results (3)



Thanks for your attention

Questions?