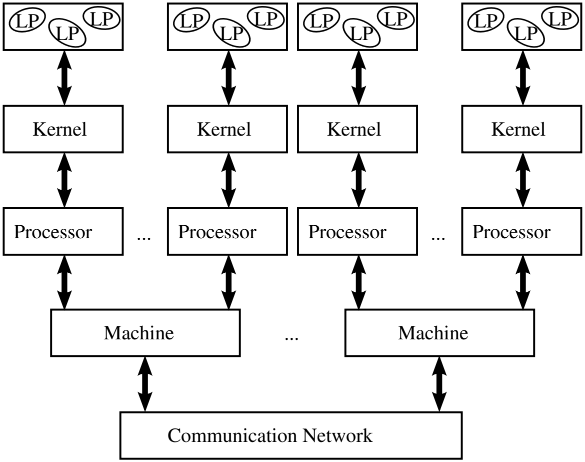
A Load-sharing Architecture for High Performance Optimistic Simulations on Multi-core Machines

Roberto Vitali
Alessandro Pellegrini
Francesco Quaglia

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

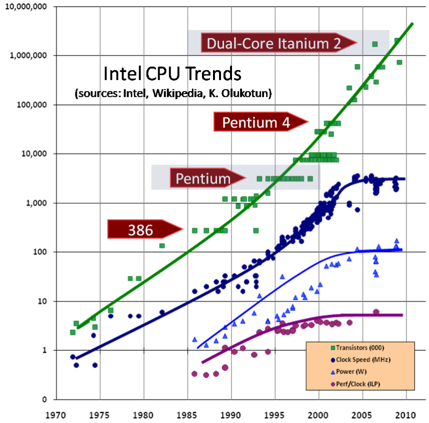
HiPC 2012 – Pune, India

PDES Logical Architecture



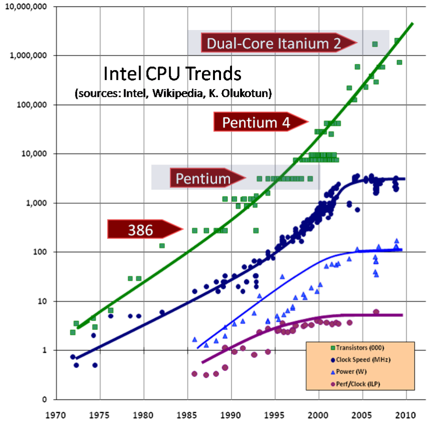
Rationale

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



Rationale

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



Goals

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

Goals

- Paradigm Shift towards Symmetric Multi-threaded Optimistic Simulation Kernels
  - Reshuffle 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
  - Computing 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

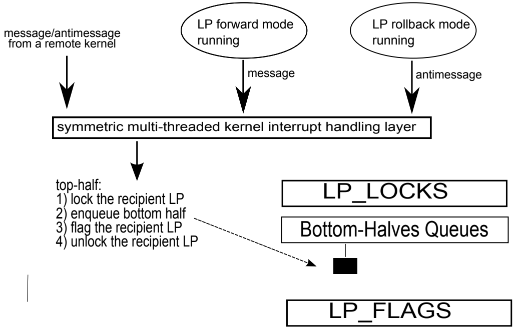
- Avoid *lock-everything effects* in kernel mode
  - Reduced set of operations/data structures
  - Inherent strict coupling among the LPs
- *Input/output queues* are frequently updated
  - Core of the cross-LP dependencies
  - Updates by the worker thread currently running the LP
  - Additionally, by 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
- *Input/output queues* are frequently updated
  - Core of the cross-LP dependencies
  - Updates by the worker thread currently running the 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

- Data-structures updates are logically considered as *interrupts*
- Interrupt tasks are not immediately finalized
- A light (constant time) top-half module is executed, for registering the operation to be performed



Computing 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 computing 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.

Computing 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.

step 1: each simulation kernel k_i, for each hosted Logical Process l:

$$L_l = \frac{q_l \cdot \delta_l}{LVT_l^{q_l} - LVT_l^1}$$

step 2: each simulation kernel k_i:

$$L^{k_i} = \sum_{l=1}^{numLP^{k_i}} L_l$$

Computing Power Reallocation Policy (3)

step 3: k_i determines the max degree of parallelism it can accomplish:

- Hosted LPs' workload factors are ordered non-increasingly:

$$L_{l_1}, L_{l_2}, \ldots, L_{l_H}$$

- The highest L_{l_1} factor is taken as reference value for a knapsack formed by LP_{l_1}
- Other j 0-1 one-dimensional knapsacks are built using *Dantzig*'s greedy approximation algorithm
- The optimal number of worker threads required by kernel k_i is $W^{K_i} = j$.

Computing Power Reallocation Policy (4)

step 4: k_i notifies the tuple $\langle W^{k_i}, L^{k_i} \rangle$ to the *master kernel*

step 5: the master kernel computes the system's workload:

$$L^{tot} = \sum_{i=1}^{K} L^{k_i}$$

step 6: a preliminary core-assignment estimation is computed:

$$T_{k_i} = \left\lfloor \frac{L^{k_i}}{L^{tot}} \cdot C \right\rfloor$$

enforcing at least $T_{k_i} = 1$.

Computing Power Reallocation Policy (5)

step 7: A refined estimation is then calculated:

$$T'_{k_i} = \begin{cases} T_{k_i} & \text{if } T_{k_i} \leq W^{k_i} \\ W^{K_i} & \text{if } T_{k_i} > W^{k_i} \end{cases}$$

step 8: if $\sum_{i=1}^{K} T_{k_i} T'_{k_i} < C$, then some cores must still be assigned. Then, kernels are non-increasingly ordered by allocation reminder $R^{k_i} = (W^{k_i} - T'_{k_i})$ and cores are round-robin assigned.

step 9: The master kernel notifies the tuple $\langle j, T'_{k_j} \rangle \forall j$.

Implementation within ROOT-Sim

- ROOT-Sim is an open-source general-purpose C-based optimistic simulation platform
- Programming model is ANSI-C

`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)

Experimental Results

Hardware Setting

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

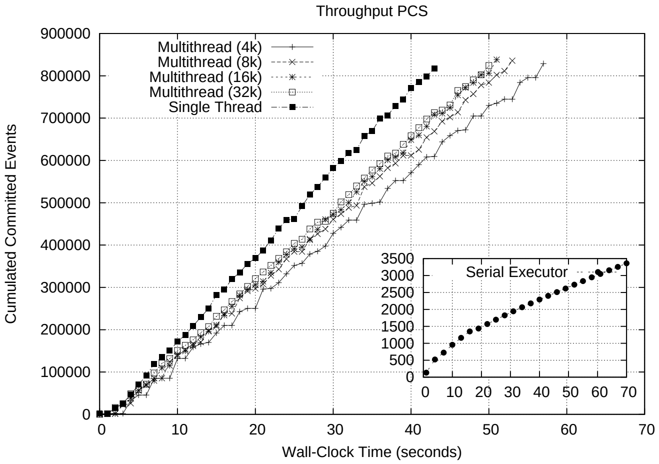
Personal Communication Service

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

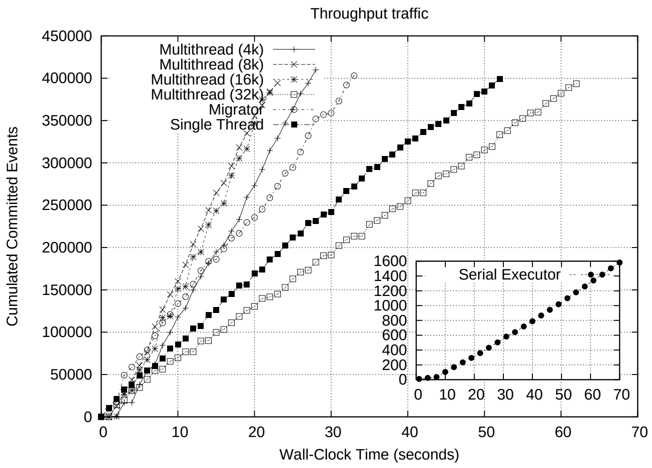
Traffic

- 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?

http://www.dis.uniroma1.it/∼pellegrini/

pellegrini@dis.uniroma1.it

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