NUMA Time Warp

Alessandro Pellegrini
Francesco Quaglia

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

PADS 2015
The NUMA Architecture

- Memory divided into different banks
- The same core sees some banks closer, other farther
- This has an effect on access latency

**AMD Opteron 6128**
The NUMA Architecture

- Memory divided into different banks
- The same core sees some banks closer, other farther
- This has an effect on access latency
- Time-Warp systems are highly demanding for memory
Reference Time Warp Architectural Context

- Optimistic PDES systems based on the multi-thread paradigm
  - highly suited for shared memory platforms
  - data exchange can be optimized
  - computing power can be well balanced

- Temporarily binding of simulation objects to worker threads
  - no concurrent access on recoverability data, and input/output queues of a simulation object

- Permanent binding of worker-threads to CPU cores

- Dual-mode execution scheme: application versus platform modes

- Worker threads schedule only one simulation object at a time (the current simulation object)
Goal: Optimizing Latency on NUMA Architectures

- NUMA-oriented memory manager
  - per-simulation-object management of memory segments made up by disjoint sets of pages
  - both static and dynamic binding of memory pages to specific NUMA nodes
- Page migration
  - to cope with worker-thread binding of simulation objects
  - based on Linux services
- Manage at the same time:
  - simulation states’ memory pages
  - recoverability data
  - event buffers
- Fully transparent to the application-level code
• malloc library calls are intercepted
• The simulation platform transforms anonymous allocations into non-anonymous allocations
Non-Anonymous Memory Allocator

- Mid-level memory manager: DyMeLoR (any other can do the job)
  - traditional version to serve model requests
  - we have a new version with no recoverability data for platform usage

- Low-level NUMA memory manager:
  - memory is pre-reserved for the mid-level memory manager
  - pre-reserving done using `mmap`
  - for each simulation object, the following meta-data are kept:
    
    ```
    void   *base;
    size_t size;
    int    active;
    ```
Managing the Memory Map

mem_map

one instance for each simulation object

size
active base

<address,numpages>

actual contiguous virtual memory pages for the segment

segment table
Allocations from Application- and Platform-level

By using `set_mempolicy` we force the Linux kernel to materialize the pages on the NUMA node closest to the worker thread.
Data Exchange Management

- Not all data accesses are “private”: what about event exchange?
Data Exchange Management

- Not all data accesses are “private”: what about event exchange?
- NUMA-oriented implementation of the bottom half-based message-exchange scheme, using additional meta-data:

  ![Diagram](image)

  - `mem_map`
  - `live_bh` `expired_bh`
  - `pages for bottom half queues`
  - `pointers switched upon a new era`
  - One instance for each simulation object
Data Exchange Management

- Not all data accesses are “private”: what about event exchange?

- The worker thread managing the destination simulation object accesses it more frequently ⇒ keep pages close to it

- Yet the pages are not guaranteed to be located on the node closest to the CPU running this worker thread!
  - remember set_mempolicy?
Page Migration: the pagemigd daemon

- `<address,numpages>`
- `map_move`
- `spinlock`
- `target_node`
- `need_move`
- `addresses[]`
- `target_nodes[]`
- `move_pages`
- `Linux kernel`

flag triggering the move of all the valid segments by the daemon

One instance for each simulation object
Migrating Bottom Halves

- `mem_map`: one instance for each simulation object
- `expired_bh`: `bh_addresses[]`
- `live_bh`: pages for bottom half queues
- `stable pointers`:
  - Pointers switched upon a new era
- One instance for each simulation object

Diagram:

```
  +-------------------+         +--------------------+
  | mem_map           |         | stable pointers     |
  +-------------------+         +--------------------+
  | bh_addresses[]    |         | one instance for    |
  +-------------------+         | each simulation    |
  | live_bh expired_bh|         | object              |
  +-------------------+         +--------------------+
  | pages for bottom  |         |
  +-------------------+         | half queues        |
  |                   |         +--------------------+
```

11 of 18 - NUMA Time Warp
Experimental Evaluation: Test-bed Platform

- **Hardware configuration:**
  - HP ProLiant server equipped with 64GB of RAM
  - 4 8-cores CPU (32 cores total)
  - 8 NUMA nodes, close to 4 cores, distant to all the others

- **Software configuration:**
  - ROOT-Sim Optimistic Simulation Kernel, using 32 symmetric worker threads
  - Debian 6
  - 2.6.32-5-amd64 Linux kernel
Benchmark Application: Traffic

- Balanced Scenario:
  - 137 simulation objects
  - Accident probability close to zero
  - Even workload (no rebinding)
  - When active, pagemigd daemons are very aggressive

- Unbalanced Scenario:
  - 1024 simulation objects
  - Number of active daemons in [4, 32]
  - Activation interval in [2, 10]
Balanced Configuration: Execution Time

![Bar chart showing execution time vs. car interarrival time for different configurations: Sequential, glibc-based, NUMA no daemons, and NUMA with daemons. The chart indicates performance metrics under varying conditions.](image)
Unbalanced Configuration: vs sequential
Unbalanced Configuration: vs glibc

![Graph showing speedup vs daemon activation interval and number of parallel daemons]
Unbalanced Configuration: with or without daemons
Thanks for your attention

Questions?

pellegrini@dis.uniroma1.it
http://www.dis.uniroma1.it/~pellegrini
http://www.github.com/HPDCS/ROOT-Sim