Transparent and Efficient Shared-State Mangement
for Optimistic Simulation on Multi-Core Machines

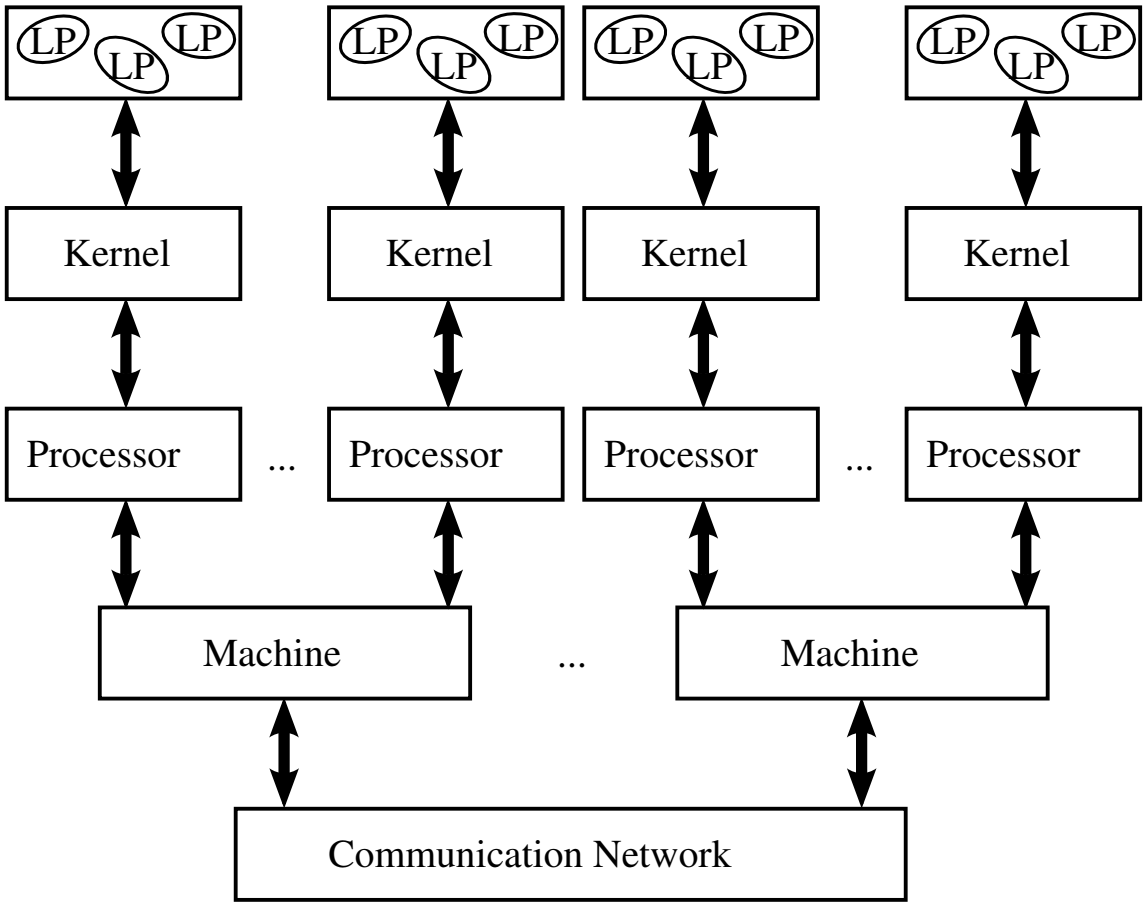
SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Pellegrini
Roberto Vitali
Sebastiano Peluso
Francesco Quaglia

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

MASCOTS 2012

Rationale (1)



Rationale (2)

$$\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset \qquad S = \bigcup_{i=1}^{numLP} S_i$$

Rationale (2)

$$\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset \qquad S = \bigcup_{i=1}^{numLP} S_i$$

- Disjoint States: Message Passing to represent interactions

Rationale (2)

$$\cancel{\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset} \qquad S = \bigcup_{i=1}^{numLP} S_i$$

- Disjoint States: Message Passing to represent interactions

Rationale (2)

$$\cancel{\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset} \qquad S = \bigcup_{i=1}^{numLP} S_i$$

- Disjoint States: Message Passing to represent interactions
- Relaxing this constraint can result in a more flexible paradigm

Rationale (2)

$$\cancel{\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset} \qquad S = \bigcup_{i=1}^{numLP} S_i$$

- Disjoint States: Message Passing to represent interactions
- Relaxing this constraint can result in a more flexible paradigm

Goal:

- Enable the application programmer to access both the LP's private state and the global portion
- Introduce a specifically-targeted Shared State Management Subsystem(SSMS)

Targets and Technical Supports

- Time Warp Synchronization protocol
- Shared-Memory Architectures

- Implement shared state as multi-versioned variables
- Propose an extended rollback scheme
- Rely on non-blocking algorithms for data synchronization
- Use software instrumentation for transparency

Read/Write Detection (1)

- Variables' accesses must be explicitly intercepted
- Actual machine-code instructions are modified at linking time
  - A specifically-targeted Instrumentation Tool is used
  - i386/x86-64 instructions are parsed
  - ELF executables can be handled

Read/Write Detection (1)

- Variables' accesses must be explicitly intercepted
- Actual machine-code instructions are modified at linking time
  - A specifically-targeted Instrumentation Tool is used
  - i386/x86-64 instructions are parsed
  - ELF executables can be handled

- Two main APIs are exposed by SSMS:
  - `write_glob_var(void *orig_addr, time_type lvt, ...)`
  - `void *read_glob_var(void *orig_addr, time_type my_lvt)`

Read/Write Detection (2)

Different code blocks can be found during the intrumentation phase:

- Load/Store operations (namely, `mov` instructions):
  - The matching SSMS' API is put in place of the instruction

Read/Write Detection (2)

Different code blocks can be found during the intrumentation phase:

- Load/Store operations (namely, `mov` instructions):
  - The matching SSMS' API is put in place of the instruction
- Operations with memory operands as destination (e.g., `inc m32`), or string instructions (e.g., `movs`):
  - The instruction is replaced by a block of operations, mimicking the same logic

Read/Write Detection (2)

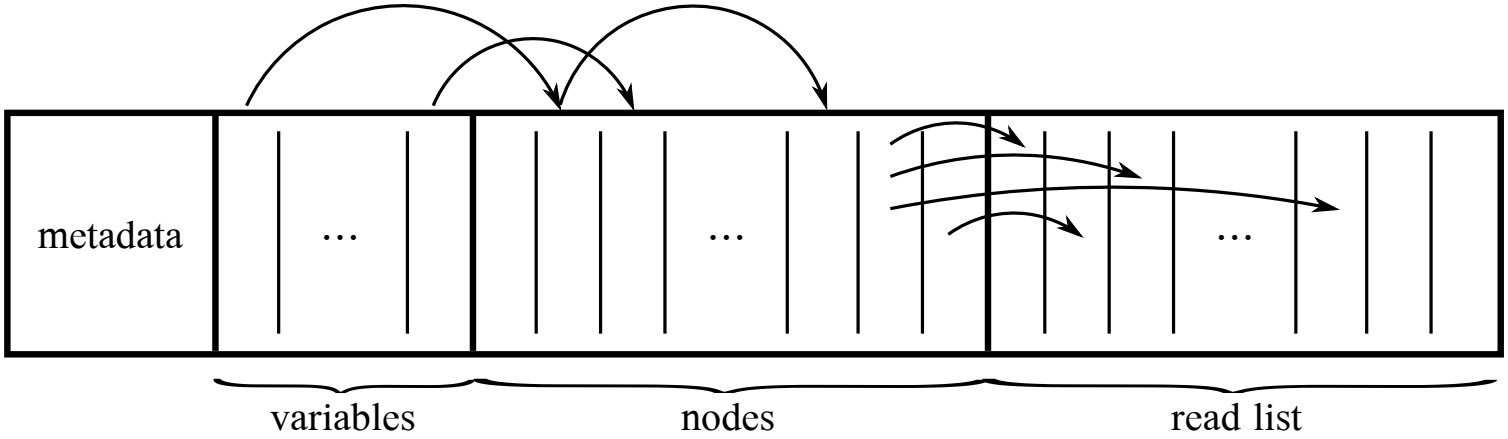
Different code blocks can be found during the intrumentation phase:

- Load/Store operations (namely, `mov` instructions):
  - The matching SSMS' API is put in place of the instruction
- Operations with memory operands as destination (e.g., `inc m32`), or string instructions (e.g., `movs`):
  - The instruction is replaced by a block of operations, mimicking the same logic
- Memory access via pointers:
  - A call to a `monitor` routine is placed before these instructions,
  - The destination address is fastly computed
  - A custom linker script is used to place boundaries on global variables
  - If the pointer falls within this area, SSMS is triggered

Read/Write Detection (3)

- To efficiently support runtime execution, an exact number of multi-versioned global variables must be installed
- At linking time the `.symtab` section is explored, to find global variables in the executable
- A table of $\langle name, address, size \rangle$ tuples is built
- At simulation startup, the correct number of multi-versioned variables is installed

Shared Memory-Map Organization



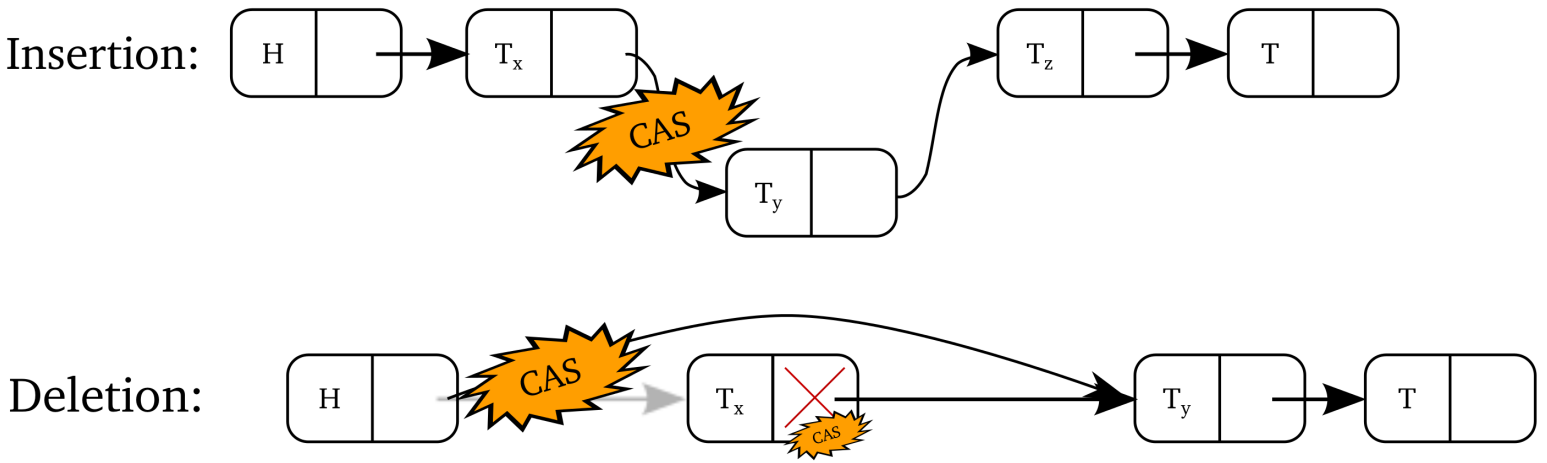

```
typedef struct _globvar_node {
  volatile int alloc;
  time_type lvt;
  unsigned char value[MAX_BUFF];
  spinlock_t read_list_spinlock;
  long long next;
} globvar_node;
```

Concurrent Allocator

```
 1: procedure ALLOCATE
 2:     m ← generate_mark()
 3:     slot ← first_node_free
 4:     while true do
 5:         alloc ← vers[slot].alloc;
 6:         if alloc ∨ ¬ CAS(vers[slot].alloc, alloc, m) then
 7:             slot ← next slot in circular policy
 8:         else
 9:             break
10:         end if
11:     end while
12:     atomically update first_node_free
13:     return slot
14: end procedure
```

Version Lists

- Multi-versioned variables are implemented as version lists
- Each node represents one variable's value at a certain lvt
- Insert/Delete operations are implemented as non-blocking operations by relying on the `CAS` primitive



Read Operation

```
1:  procedure READ(addr, lvt)
2:      slot ← hash table's entry associated with addr
3:      if slot ∈ AccessSet then
4:          version ← AccessSet[slot]
5:      else
6:          version ← FIND-NODE(slot, lvt)
7:          AccessSet[slot] ← version
8:      end if
9:      return vers[version].value;
10: end procedure
```

Write Operation

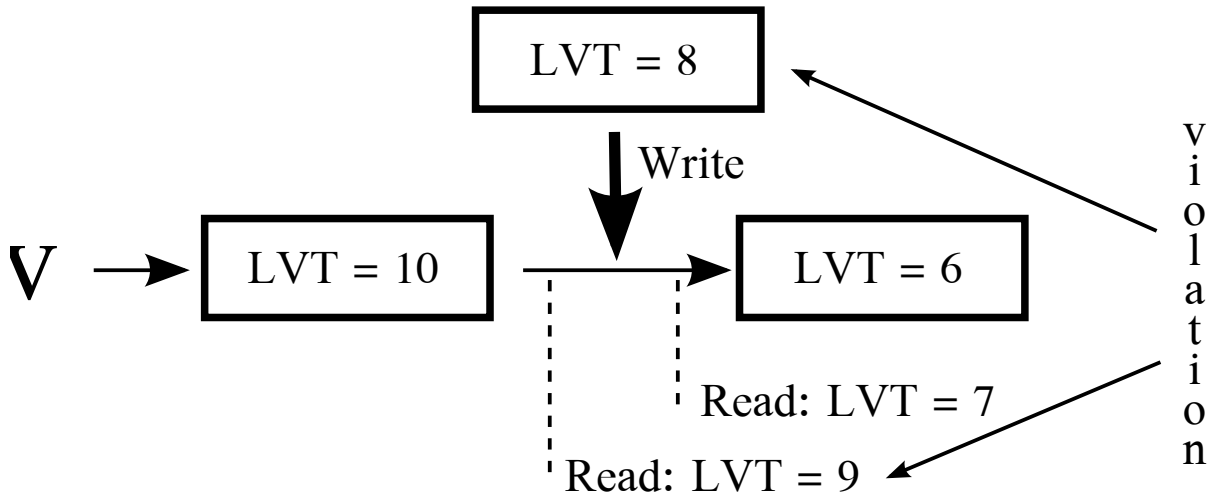
```
1:  procedure WRITE(addr, lvt, val)
2:      slot ← hash table's entry associated with addr
3:      if slot ∈ AccessSet then
4:          version ← AccessSet[slot]
5:          vers[version].value ← val
6:      else
7:          version ← INSERT-VERSION(slot, lvt, val)
8:          AccessSet[slot] ← version
9:      end if
10: end procedure
```

Synchronization and Rollback (1)

- To strengthen the optimism, we allow interleaved reads and writes on a version list
- We explicitly avoid a freshly installed version to invalidate any version related to a greater lvt

Synchronization and Rollback (1)

- To strengthen the optimism, we allow interleaved reads and writes on a version list
- We explicitly avoid a freshly installed version to invalidate any version related to a greater lvt



Synchronization and Rollback (2)

- Processes which reads a version node must leave a mark, i.e., visible reads are enforced.
- Classical *rollback*'s notion is augmented:
  - In case of inconsistent read, a special anti-message is sent to the related LP

Synchronization and Rollback (2)

- Processes which reads a version node must leave a mark, i.e., visible reads are enforced.
- Classical *rollback*'s notion is augmented:
  - In case of inconsistent read, a special anti-message is sent to the related LP

- A *ReadList* is maintained, to keep track of versions reads
- After each *Write* operation, the *ReadList* of the previous node is checked to see if an anti-message must be scheduled to some LPs

Synchronization and Rollback (2)

- Processes which reads a version node must leave a mark, i.e., visible reads are enforced.
- Classical *rollback*'s notion is augmented:
  - In case of inconsistent read, a special anti-message is sent to the related LP

- A *ReadList* is maintained, to keep track of versions reads
- After each *Write* operation, the *ReadList* of the previous node is checked to see if an anti-message must be scheduled to some LPs
- When an antimessage is received because of an inconsistent read, version nodes related to that particular event must be removed
  - This is done by connecting every node in the *message queue* with version nodes installed during an event execution

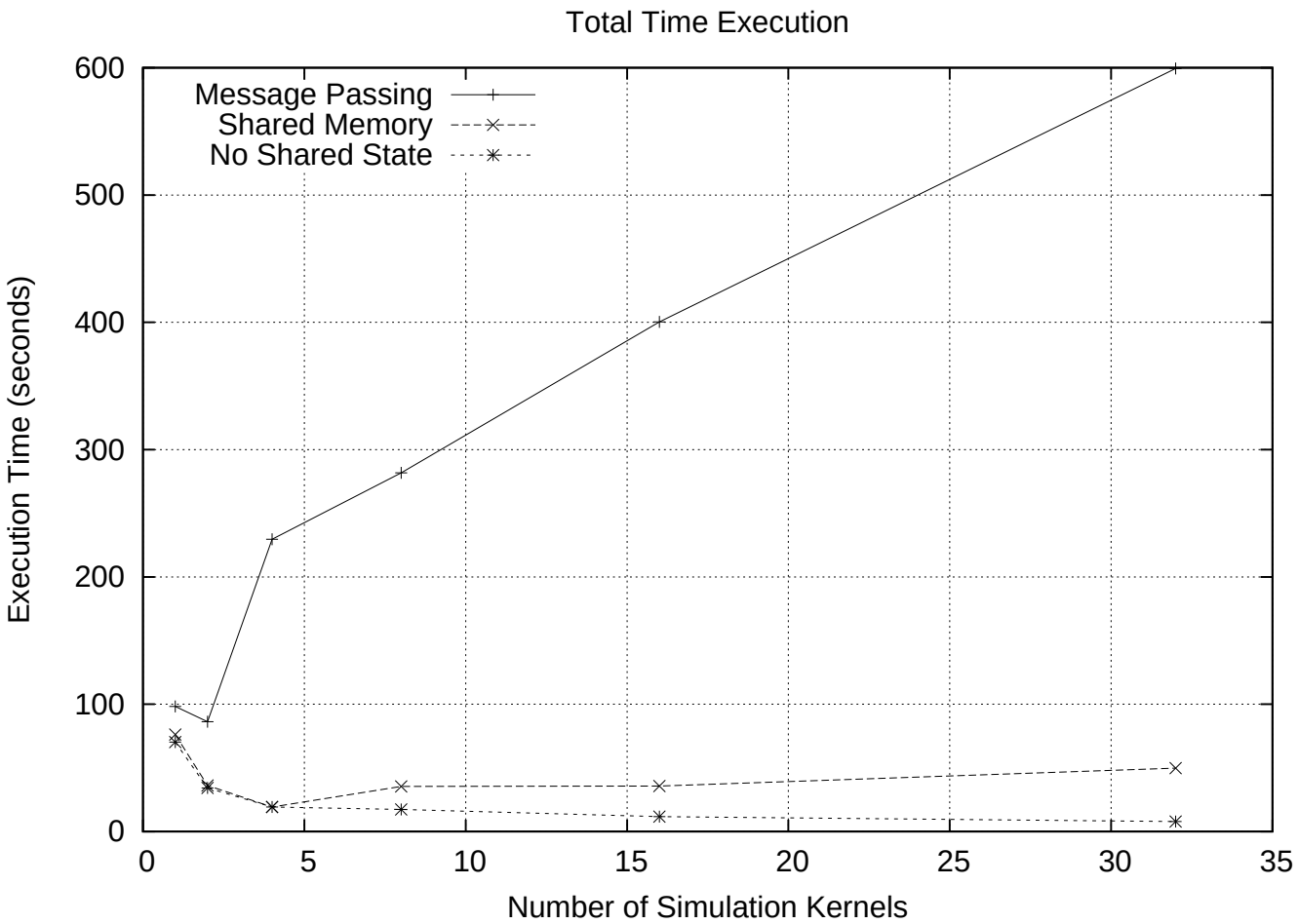
Experimental Results (1)

- We have run our model on top of the ROme OpTimistic Simulator (ROOT-Sim), an open-source, general-purpose simulation platform developed using C/POSIX technology

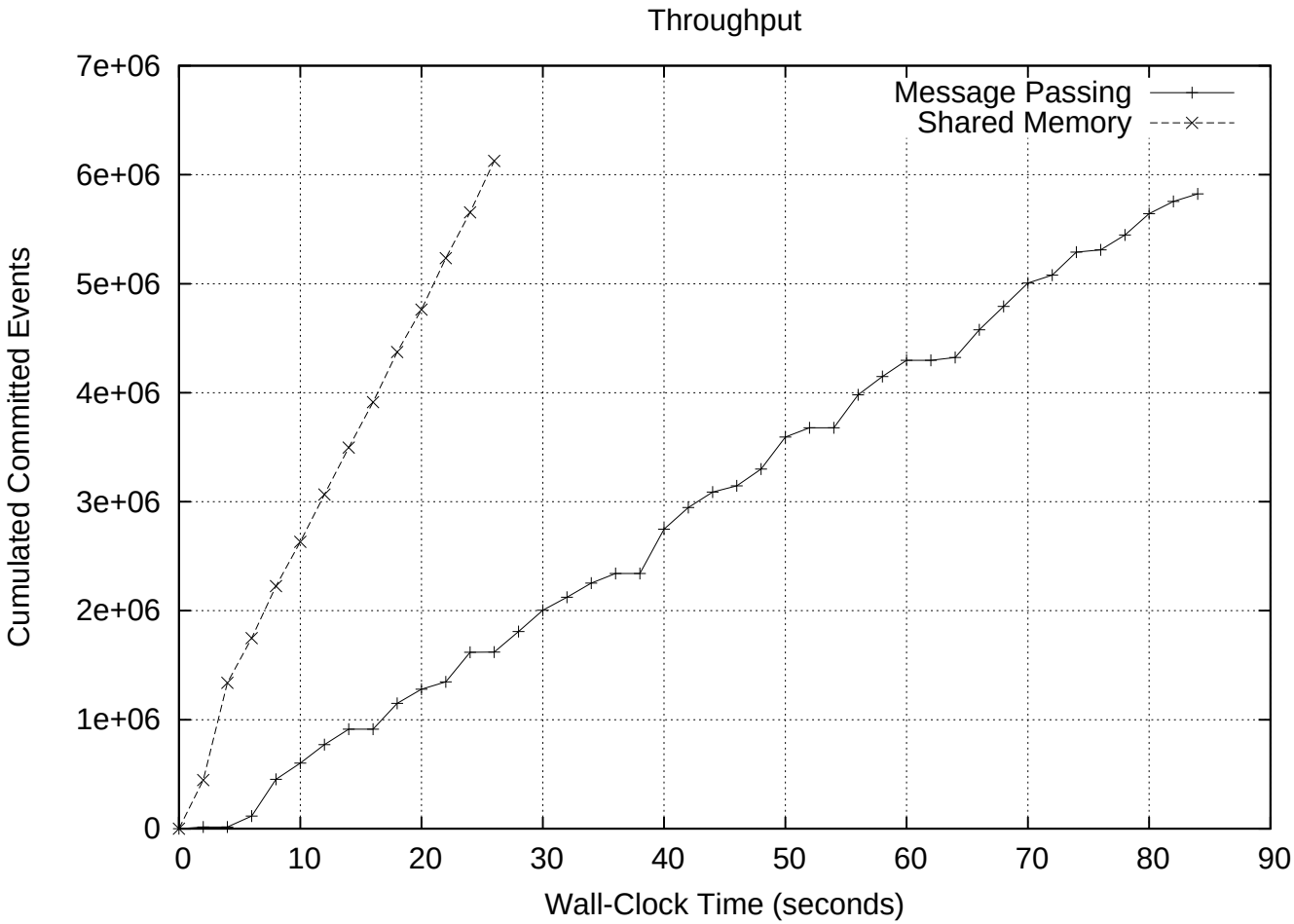
`http://www.dis.uniroma1.it/~hpdcs/ROOT-Sim/`

- As a test-bed, we have used Personal Communications Service (PCS), a suite of differently parameterized simulation models of wireless communication systems adhering to GSM technology
- Global variables handle global statistics, i.e. the total number of calls, the total number of handoffs, and the global cumulated power

Experimental Results



Experimental Results



Thanks for your attention

Questions?

http://www.dis.uniroma1.it/~hpdcs/ROOT-Sim
http://www.dis.uniroma1.it/~pellegrini
pellegrini@dis.uniroma1.it