Multi-core parallelism for ns-3

Guillaume Seguin

INRIA Sophia-Antipolis Méditerranée, PLANETE team

16/09/2009

Goals

- Being non-intrusive
- Speeding up simulations

About ns-3

- ns-3 is a discrete-time, event-driven network simulator
- ns-3 is split into a simulator part, which has no clue about the fact that it simulates networks, and a models part, which holds all the actual networking knowledge
- Network topologies and applications are described by C++ or Python scripts

Partitioning the network

- We do not want to ask the user to partition the network
- We cannot compute an efficient partitioning (we need more than the network topology)

→ We will use partitions with only one node

Synchronization algorithm

What does ensure simulation consistency ? An ordering constraint on events dates at each node.

- Conservative algorithms : enforce ordering constraint, simple but maybe slower
- Optimistic algorithms : does not enforce the constraint at processing time, requires extra coding (user-level, compiler-level, simulation status snapshots)

Our choice : conservative algorithms.

Synchronization barrier based algorithm

Lookahead

The lookahead for a given partition is the minimum delay before a neighboring partition may receive an event from this partition.

At each iteration :

- Threads gather at a meeting point (the barrier)
- A global maximum date up to which events can be processed is computed (using the date of the next event of each partition + the partition lookahead)
- Threads process events of all the partitions up to this date

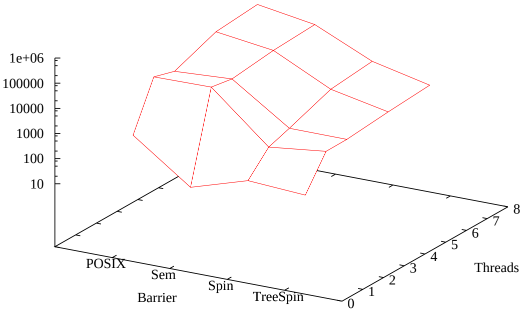
Thread safety

In ns-3, most structures are only written by the node they are related to, and are read in a thread-safe way.
Problems :

- Reference counting : disabling it in spots where problematic Ref/Unref could occur would be too intrusive $\rightarrow$ make it thread-safe
- Packet-related caches and buffers : simply disable them

Optimizing barriers

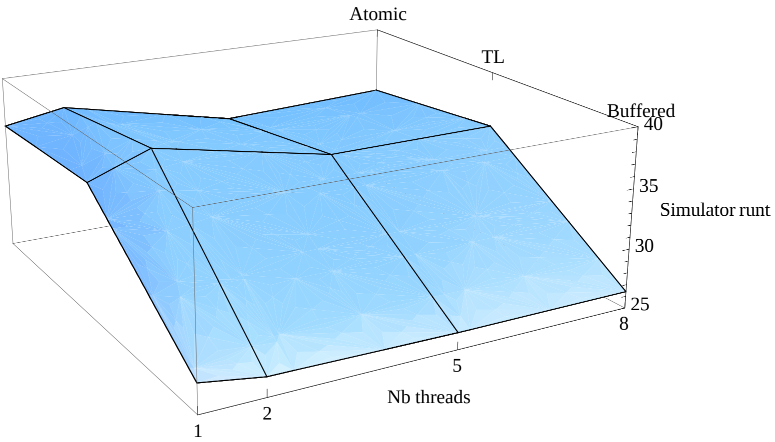
- POSIX barriers
- Semaphores barriers
- Spinlock barriers
- Tree-shaped spinlock barriers



Thread-safe reference counting

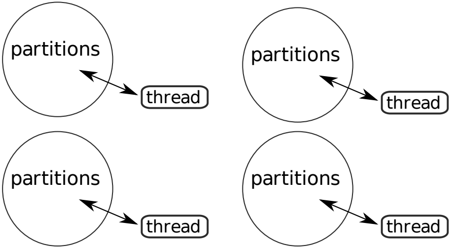
- Atomic operations implementation
- Thread-local reference counting
- Thread-local buffered reference counting : instead of directly applying operations, store them in a thread-local buffer and push the buffer when its full to a central processing location

Thread-safe reference counting performance



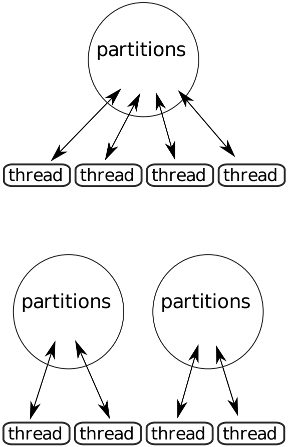
Solving memory contention

Avoid contention on a single shared ressource by dedicating each partition to a core

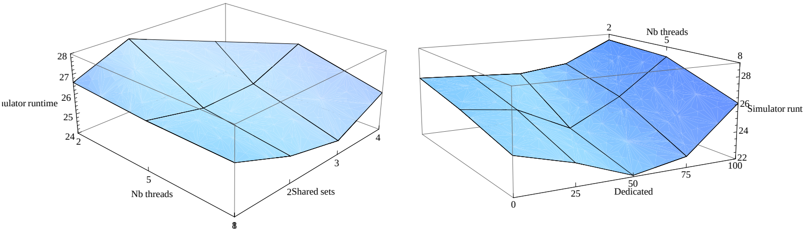


Balancing workload

- Dedicate a given ratio of the partitions to cores and keep the remaining in a shared set
- Use several shared sets to reduce contention even more



Results


28
27
26
25
24
ulator runtime
2
5
Nb threads
8
2Shared sets
3
4
2
Nb threads
5
8
28
26
24
22
Simulator runt
0
25
50
Dedicated
75
100

Further work

- Find better test cases
- Even smarter workload balancing
- Make packet-related caches thread-safe
- Computing lookahead for wireless networks

Conclusion

- Non-intrusivity leads to tough design choices
- Thread safety is quite expensive
- Linear gain looks really hard to reach
- Yet, we were able to achieve 20% speedup with 8 times more computation power.

Questions ?