Integrating Non-blocking Synchronisation in Parallel Applications: Performance Advantages and Methodologies

Philippas Tsigas Yi Zhang
Chalmers University of Technology

Outline

- Synchronisation in shared memory multiprocessor systems.
- Performance of synchronisation.
- Using non-blocking synchronisation in parallel applications.
- Conclusions.

Synchronisation in Shared Memory Systems

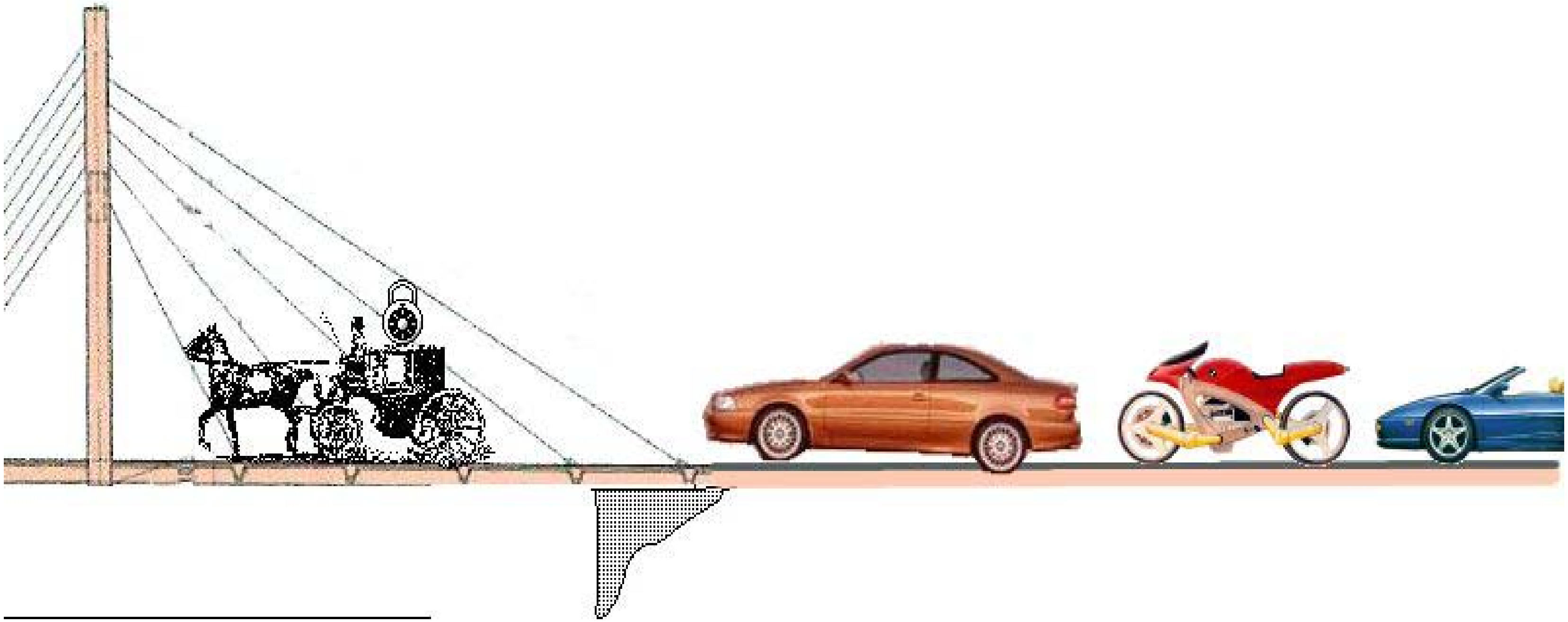
- Shared memory multiprocessor systems
  - UMA
  - NUMA
- Synchronisation
  - Mutual Exclusion
  - Non-blocking Synchronisation (lock-free, wait-free)

Performance and Synchronisation

- Synchronisation contributes a significant part in the computation time of parallel applications.
- Network contention
  - Access to shared memory
  - Spinning on shared memory
  - Cache coherent protocols
- Lock convoys

Lock Convoy

* Slowdown of one process may cause the whole system slowdown



Previous Work: Non-blocking Synchronisation in General

Synchronisation:

- An alternative approach for synchronisation.
- Protect shared objects without using mutual exclusion.

Evaluation:

- Micro-benchmarks shows better performance than mutual exclusion in real or simulated multiprocessor systems.

Our Results

How performance of parallel applications is affected by the use of non-blocking synchronisation rather than lock-based one?

- The identification of the basic locking operations that parallel programmers use in their applications.
- The efficient non-blocking implementation of these synchronisation operations.
- The architectural implications on the design of non-blocking synchronisation.
- Comparison of the lock-based and lock-free versions of the respective applications

Applications

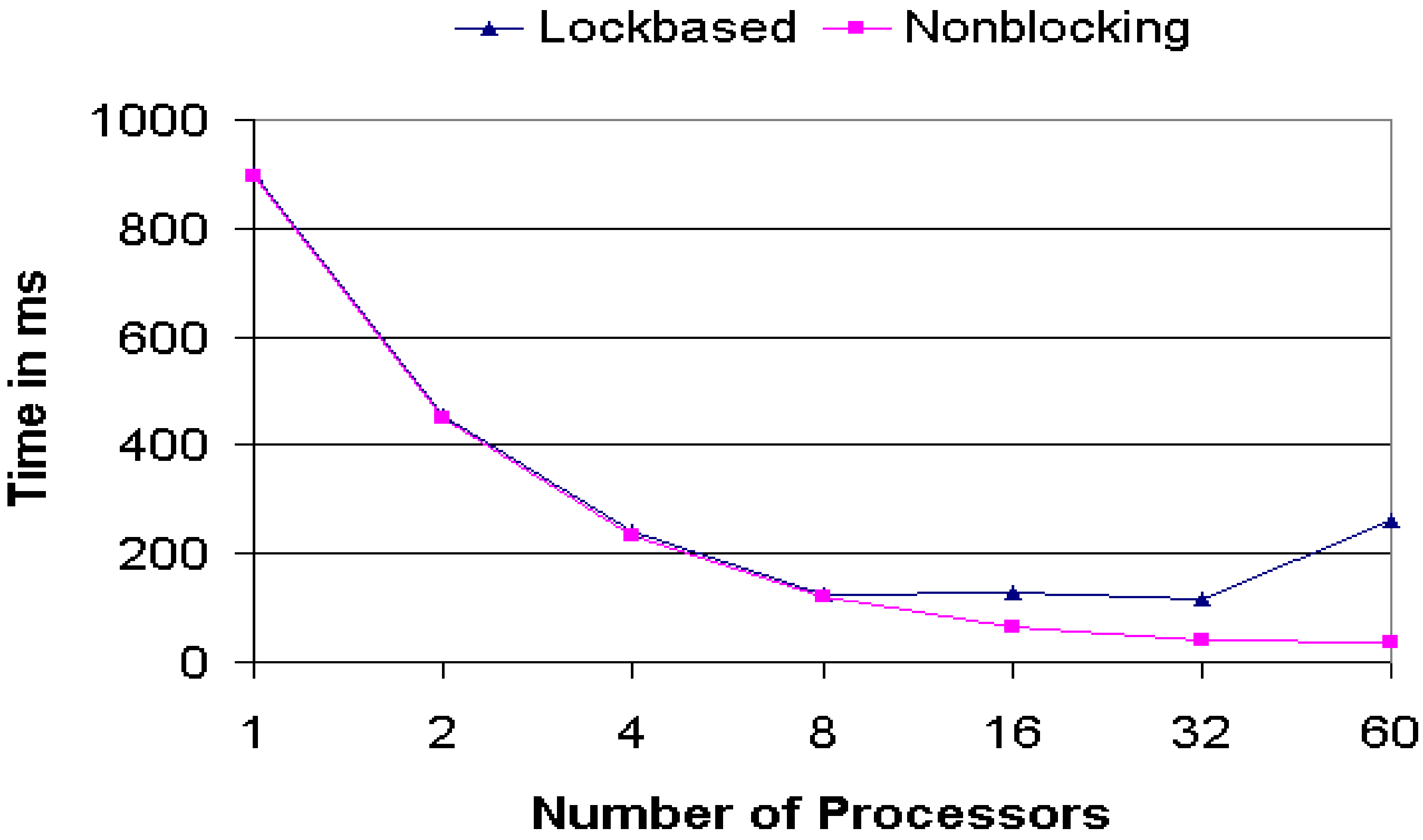
Ocean	simulates eddy currents in an ocean basin.
Radiosity	computes the equilibrium distribution of light in a scene using the radiosity method.
Volrend	renders 3D volume data into an image using a ray-casting method.
Water	Evaluates forces and potentials that occur over time between water molecules.
Spark98	a collection of sparse matrix kernels.

Removing Locks in Applications

- Most locks are SimpleLock.
- Many critical sections contain shared floating-point variables.
- Large critical sections.

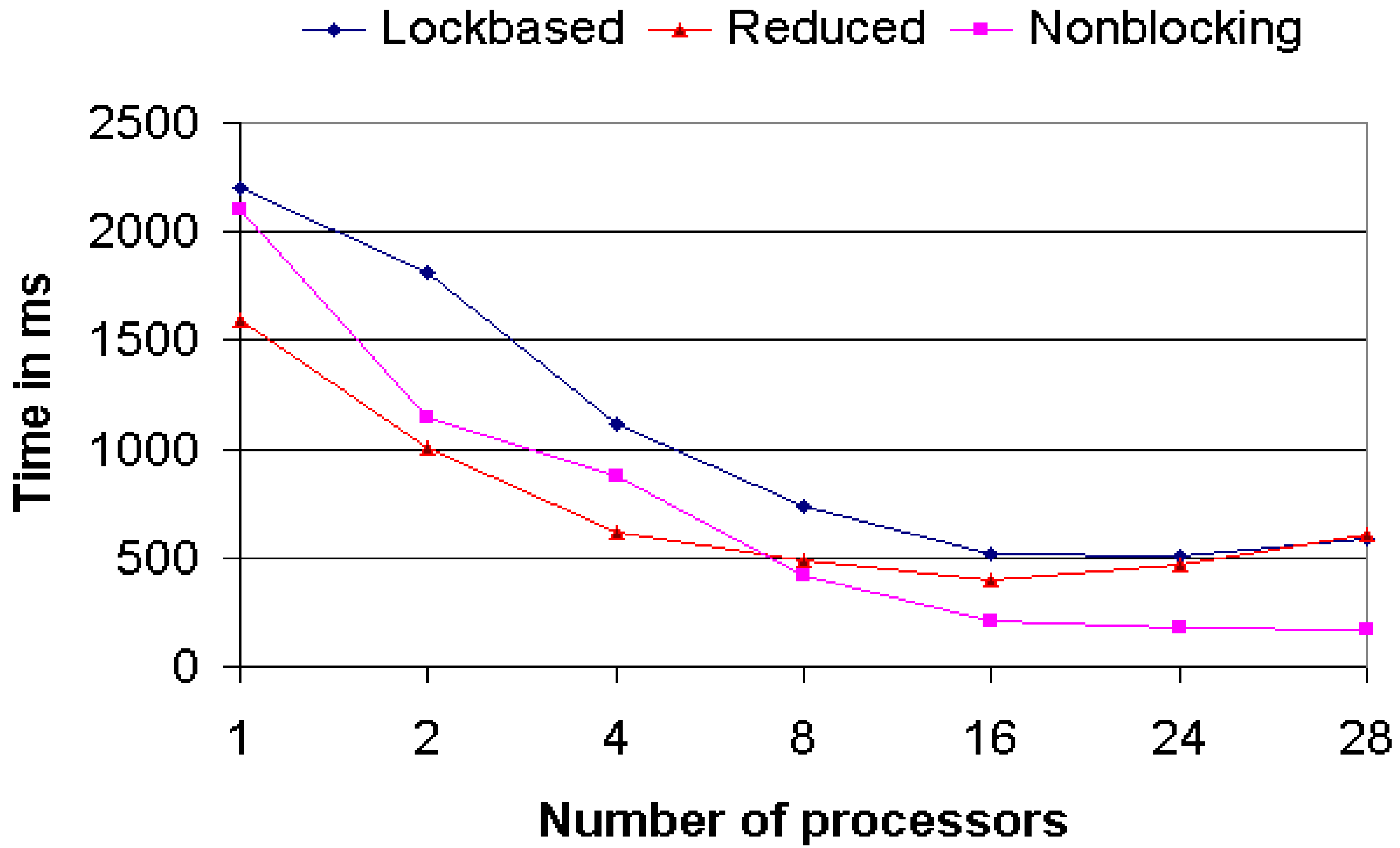
- CAS and LL/SC can be used to implement non-blocking version.
- Floating-point primitives are needed. A Double-Fetch-and-Add implementation is proposed here.
- Efficient Non-blocking bsp_tree and queue implementations are used.

Volrend

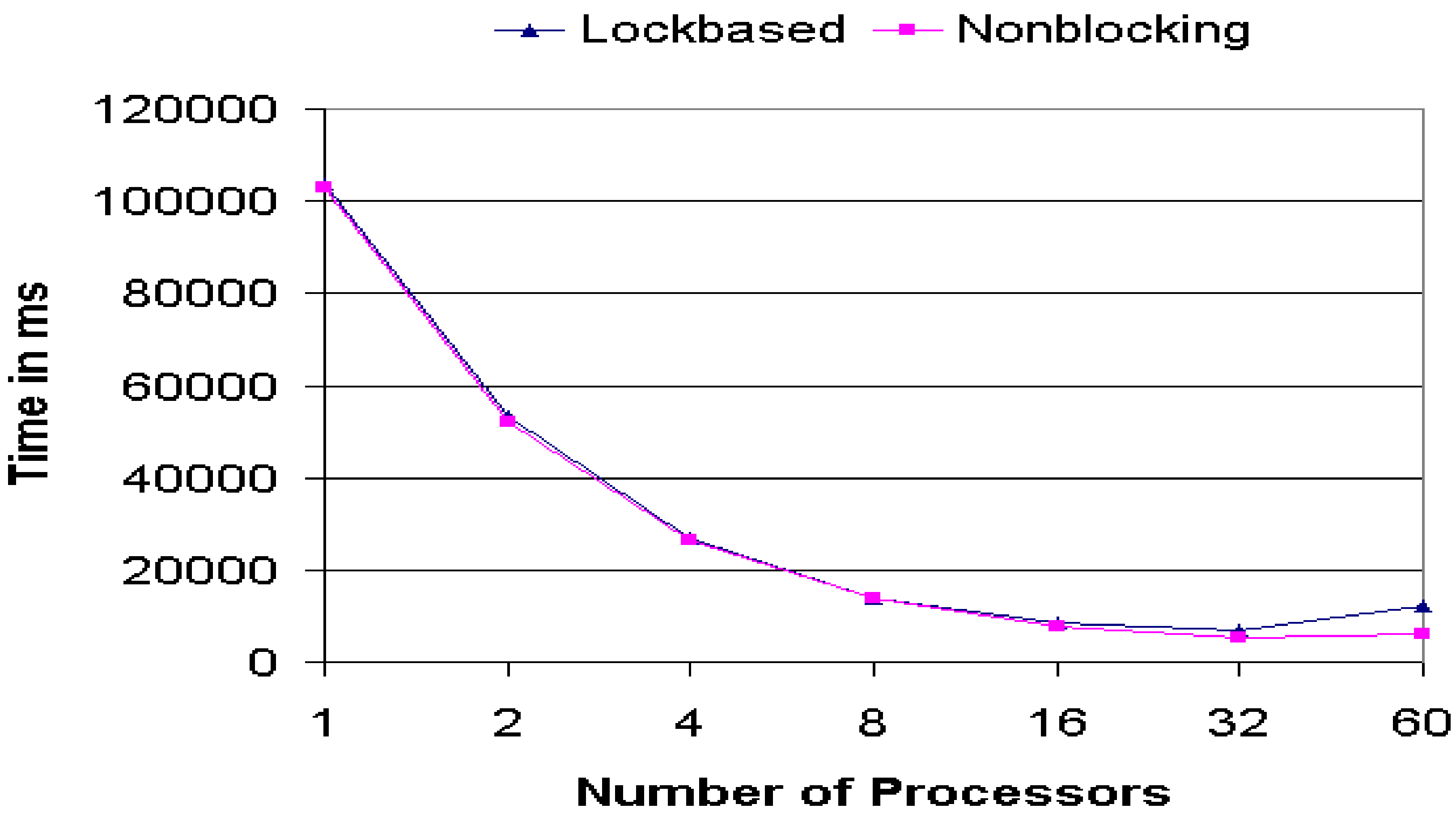


Lockbased
Nonblocking
1000
800
600
400
200
0
Time in ms
1
2
4
8
16
32
60
Number of Processors

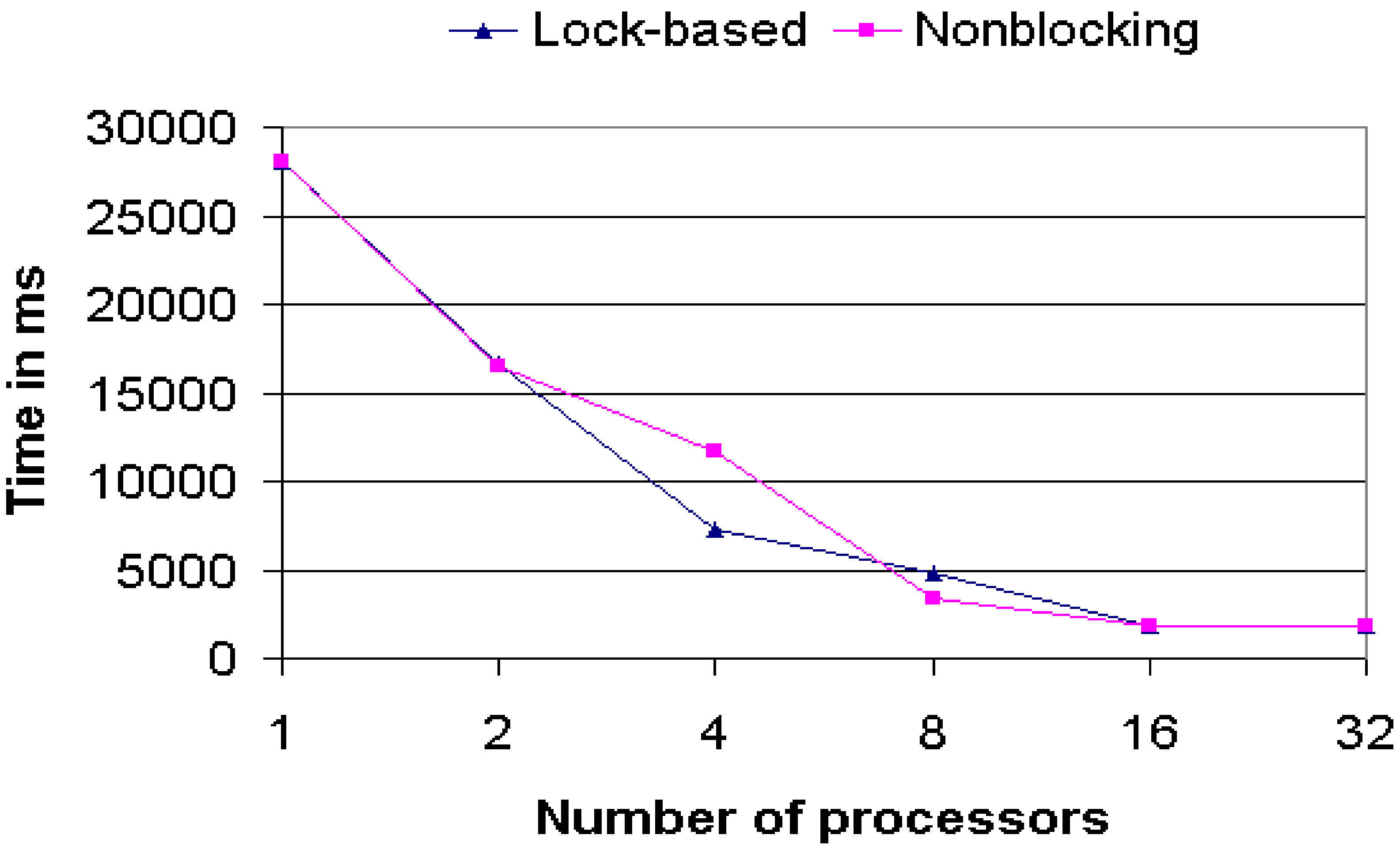
SPARK98


Lockbased
Reduced
Nonblocking
Time in ms
2500
2000
1500
1000
500
0
1
2
4
8
16
24
28
Number of processors

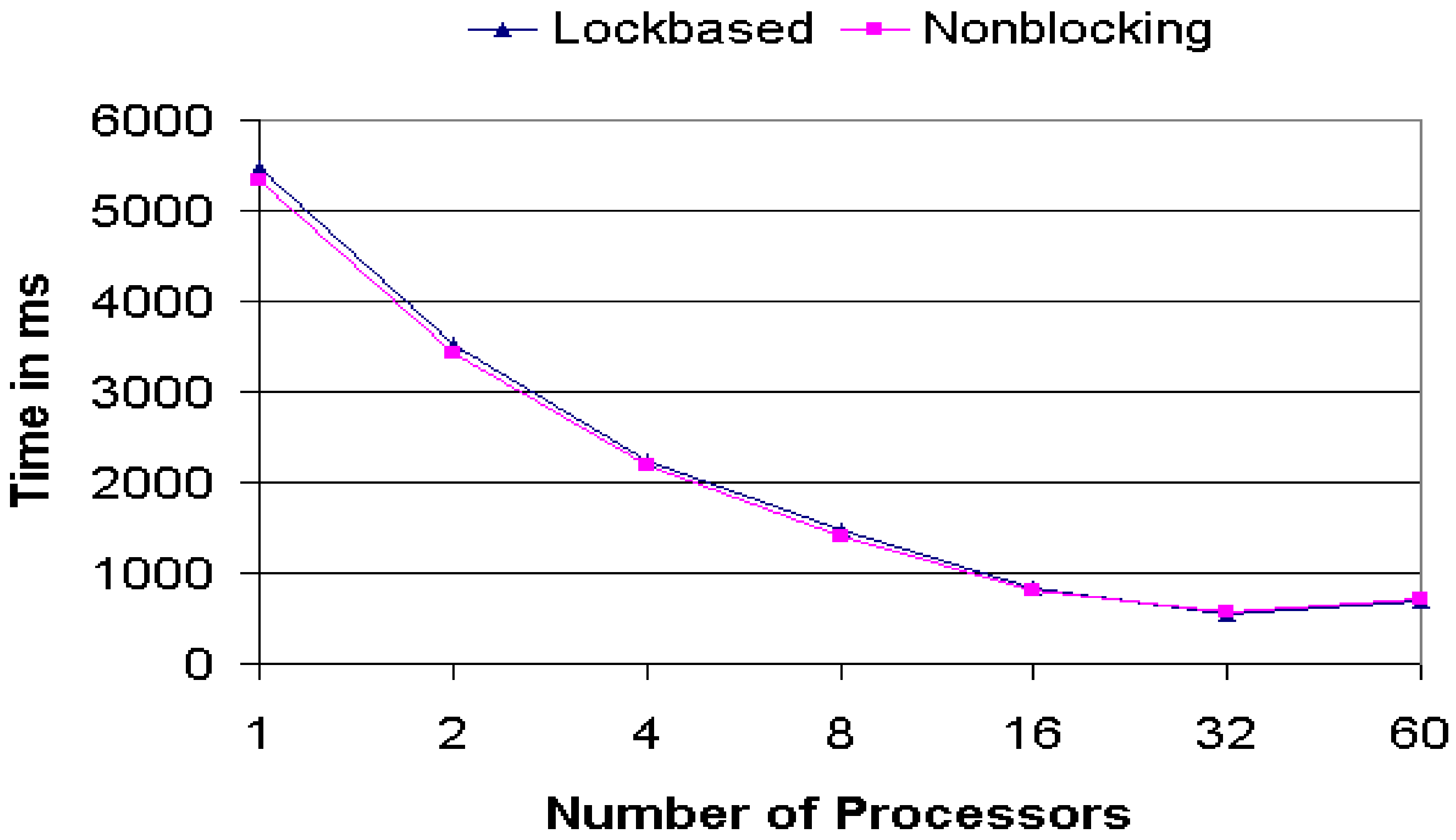
Radiosity


Lockbased
Nonblocking
120000
100000
80000
60000
40000
20000
0
1
2
4
8
16
32
60
Time in ms
Number of Processors

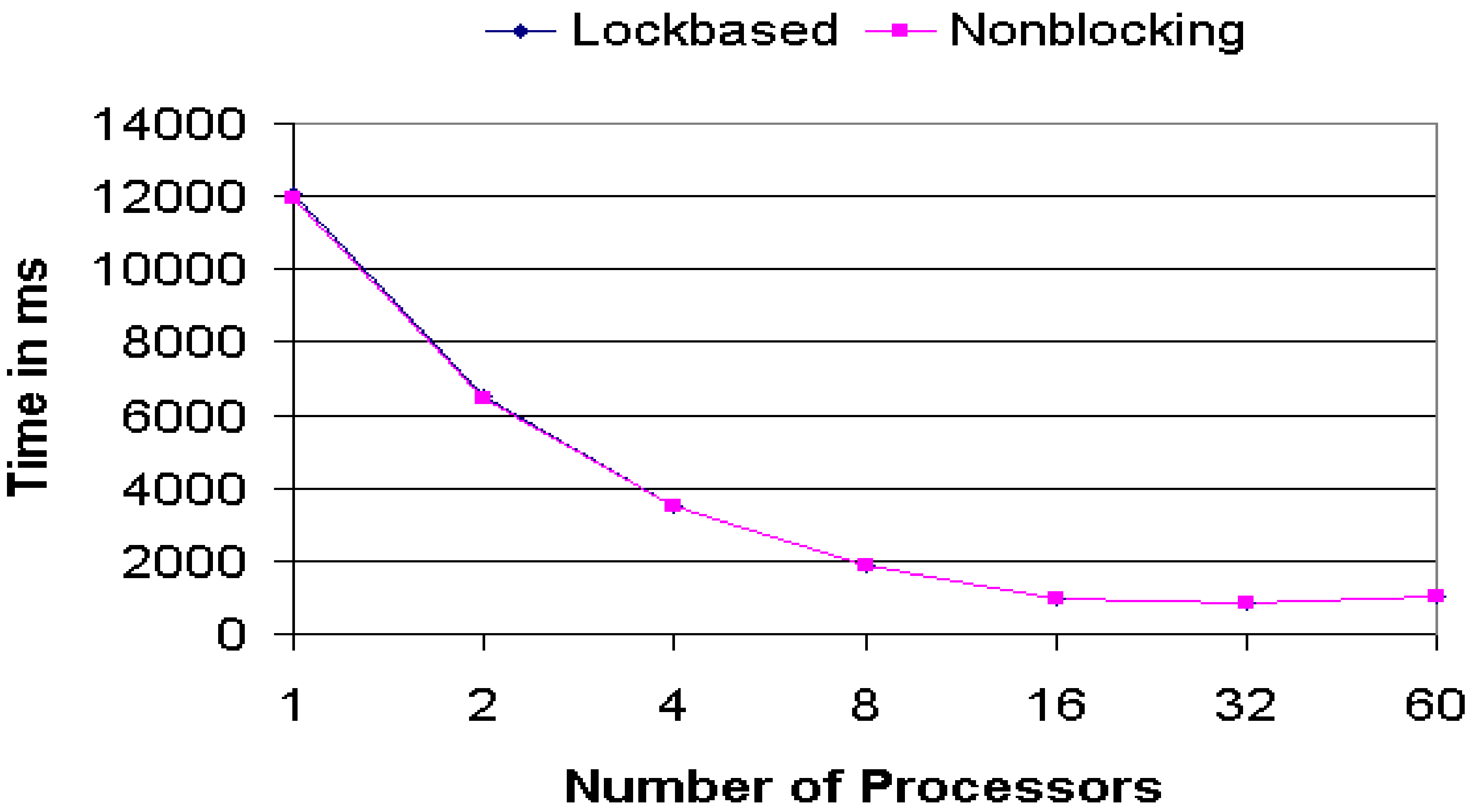
Ocean


Lock-based
Nonblocking
Time in ms
30000
25000
20000
15000
10000
5000
0
1
2
4
8
16
32
Number of processors

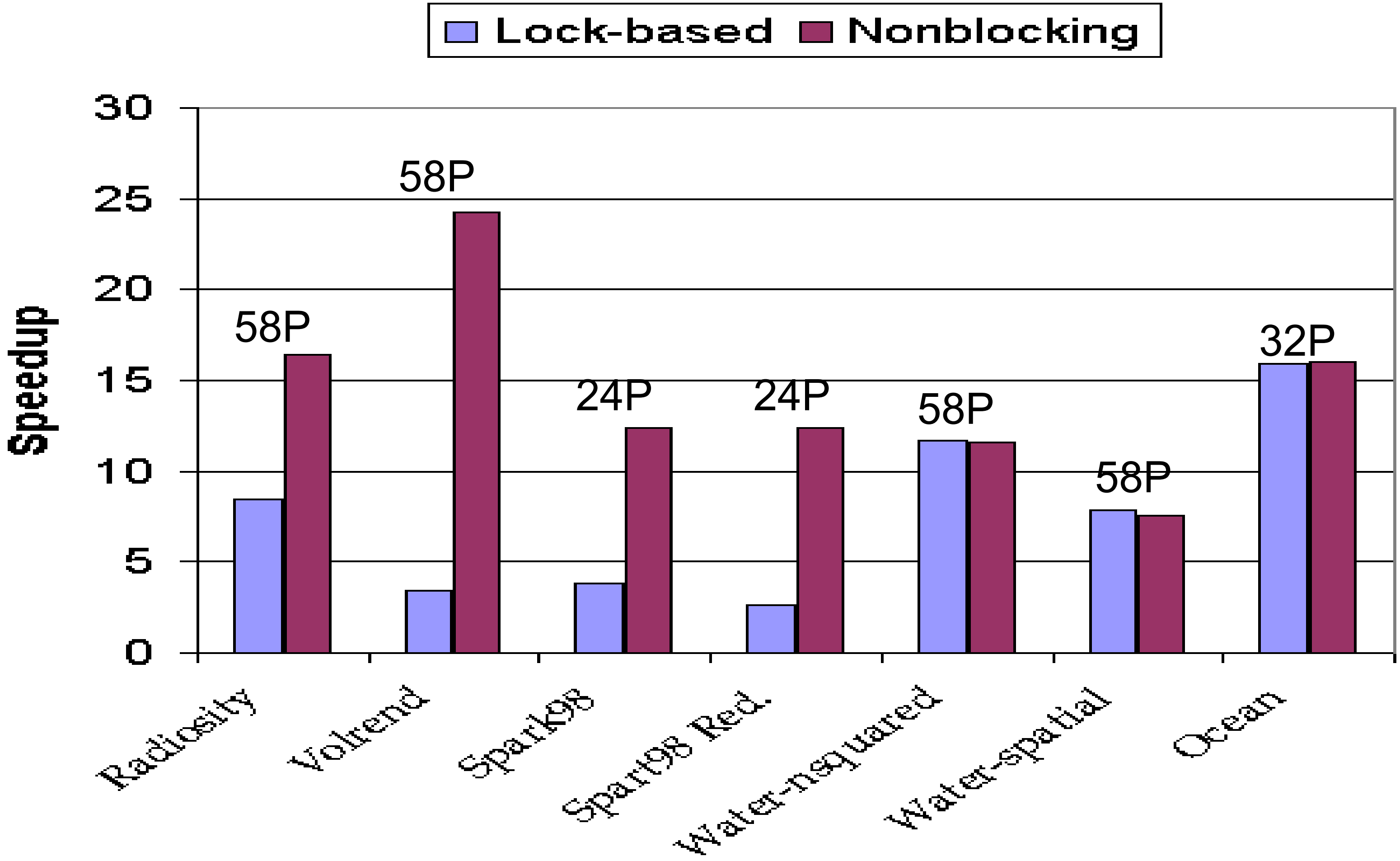
Water-spatial



Water-nsquared



Experimental Results: Speedup



Conclusions

- Non-blocking synchronisation performs as well, and often better than the respective blocking synchronisation.
- For certain applications, the use of non-blocking synchronisation yields great performance improvement.
- Irregular applications benefit the most from non-blocking synchronisation.
- Efficient methods for removing locks in parallel application are presented.

Future Work

- Experiments with more applications.
- Understanding in more detail how non-blocking synchronisation benefits applications.
- Deriving more efficient and general methods to transfer mutual exclusion to non-blocking.

Non-blocking Synchronisation Lock-free

- Definition:
  - If several processes concurrently invoke operations on the same object, although some of them might halt or fail, ***some*** processes is guaranteed to completes their operation in a finite number of their own steps
- Allows individual processes to starve
- Usually implemented as Read-Modify-Write retry loop

Non-blocking Synchronisation

- Wait-free synchronisation
  - All concurrent operations can proceed independently of the others.
  - Every process always finishes the protocol in a bounded number of steps, regardless of interleaving
  - No starvation