Restart optimization using incremental context saving in hardware transactional memory

Zaharije Radivojević, Miloš Cvetanović
School of Electrical Engineering, Belgrade University

16th Workshop
“Software Engineering Education and Reverse Engineering”

Jahorina, Bosnia and Herzegovina
21-27 August 2016

Agenda

- Background
- Transactional Memory
- Restart Optimization
- Model and Simulation
- Results
- Conclusions

Background

Fields of interest:

- Concurrent programming
- Simulations
- Database systems
- Computer architecture

Is there a cross-section?

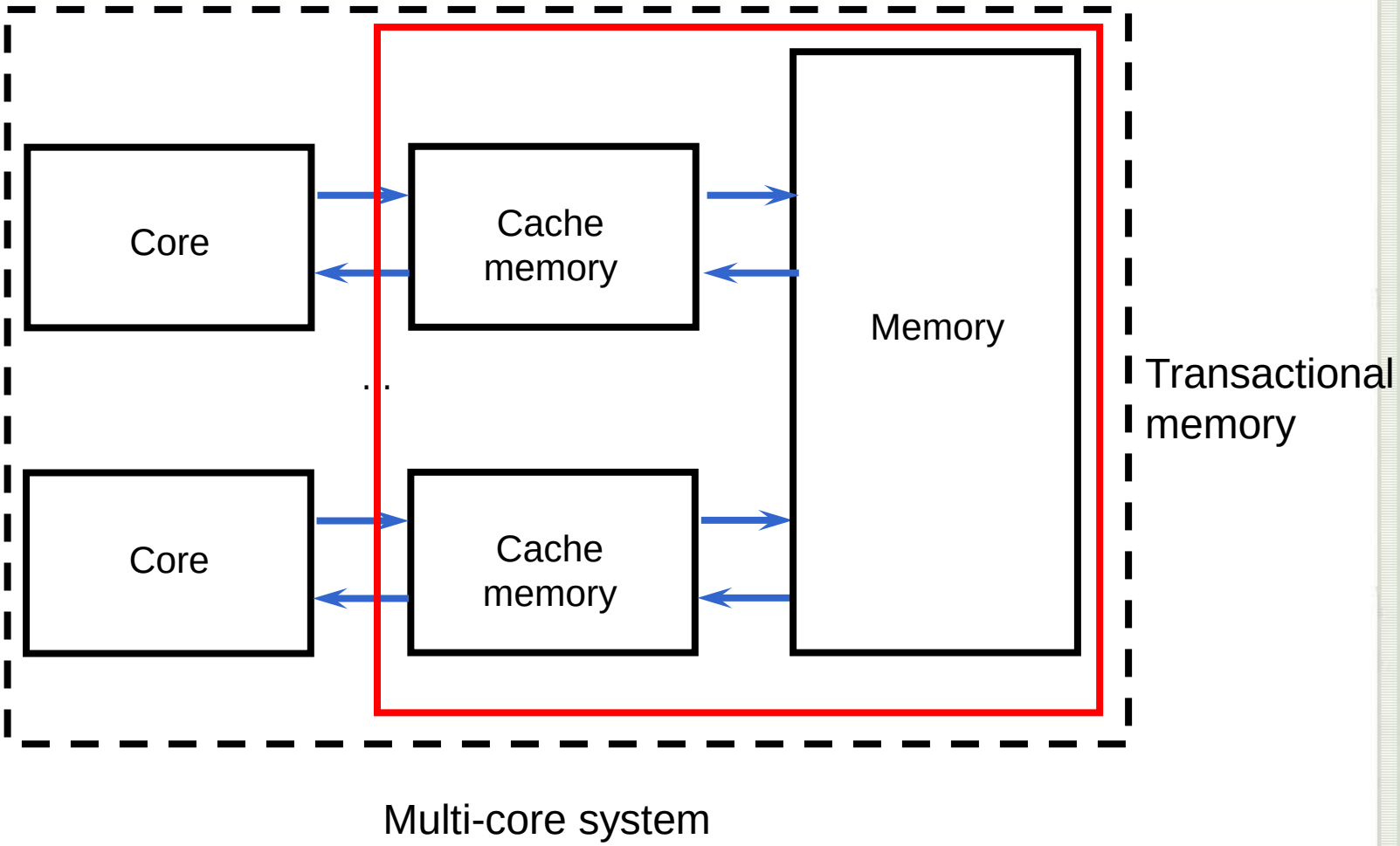
- Hardware transactional memory!

Hardware transactional memory

- The goal of (*hardware*) transactional memory is to simplify multi-core concurrent programming.

- Transactional memory provides run-time support, which includes
  - atomicity,
  - concurrency,
  - isolation.

Hardware transactional memory

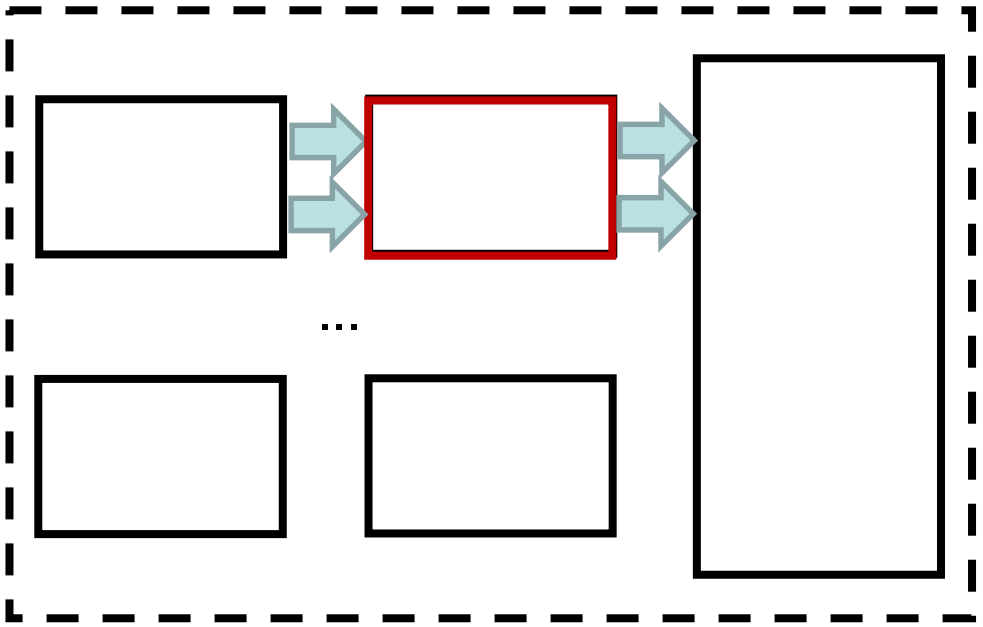


Hardware transactional memory

Version management defines whether modifications

- perform directly to the shared data (eager version management),
- are buffered as speculative writes (lazy version management).

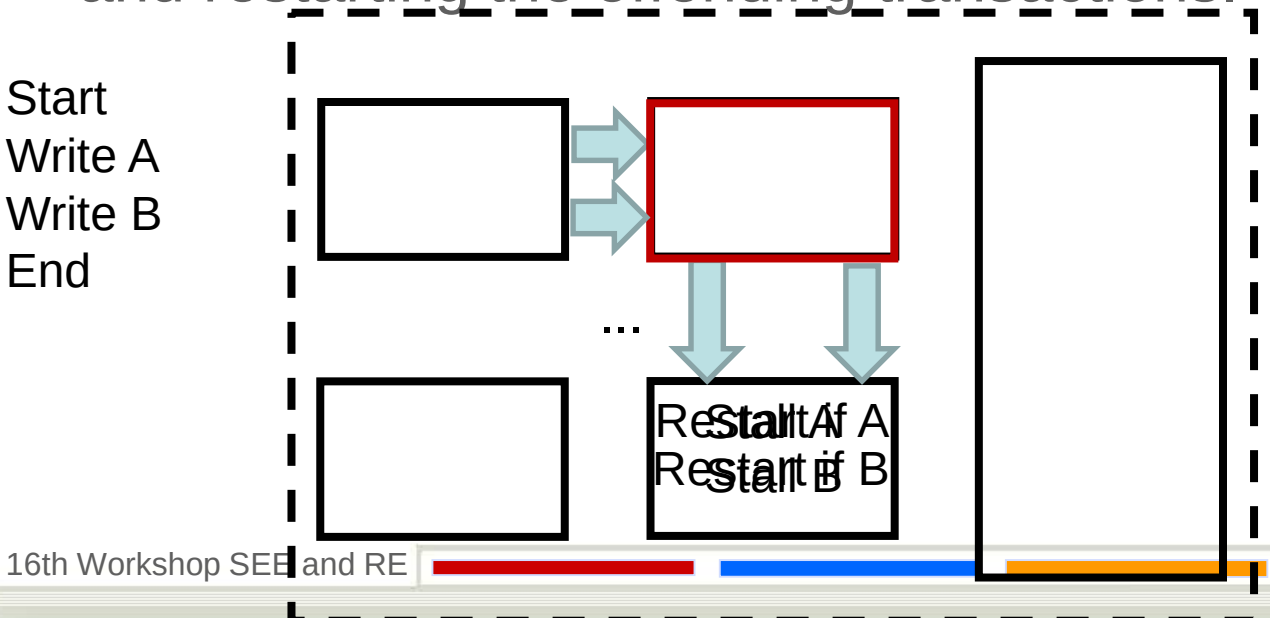
Start
Write A
Write B
End



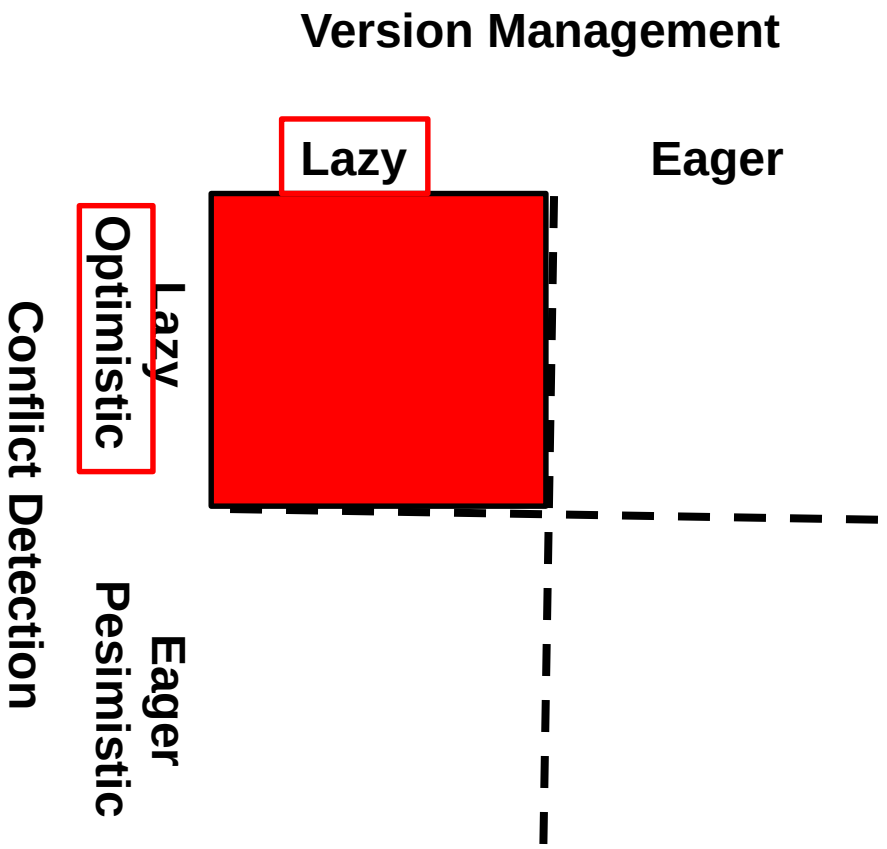
Hardware transactional memory

Conflict detection

- Eager conflict detection tends to prevent possible conflicts by introducing synchronization mechanisms that stall the offending transactions.
- Lazy conflict detection tends to react to consequences of the conflicts afterwards by aborting and restarting the offending transactions.

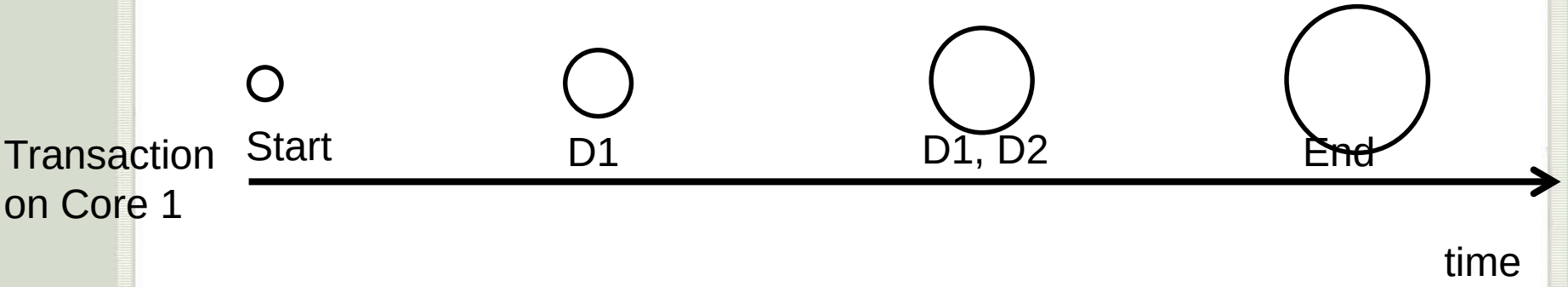


Hardware transactional memory

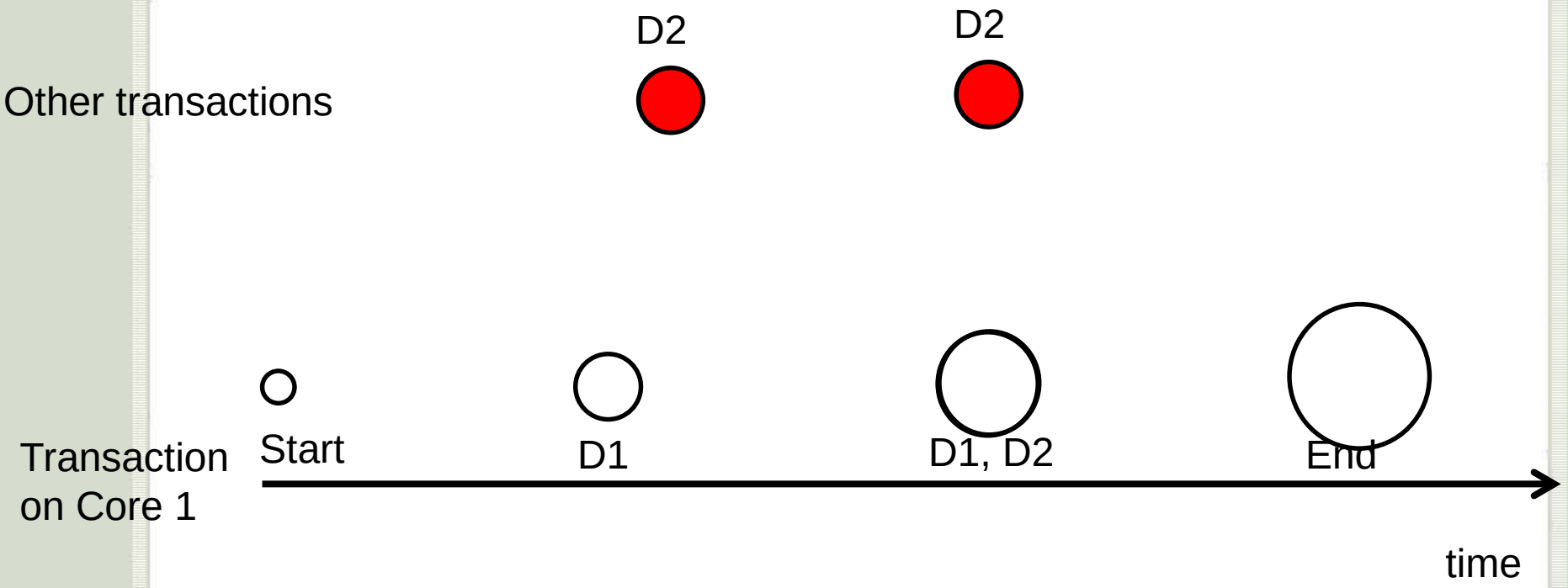


Transactions

No transactions on other cores



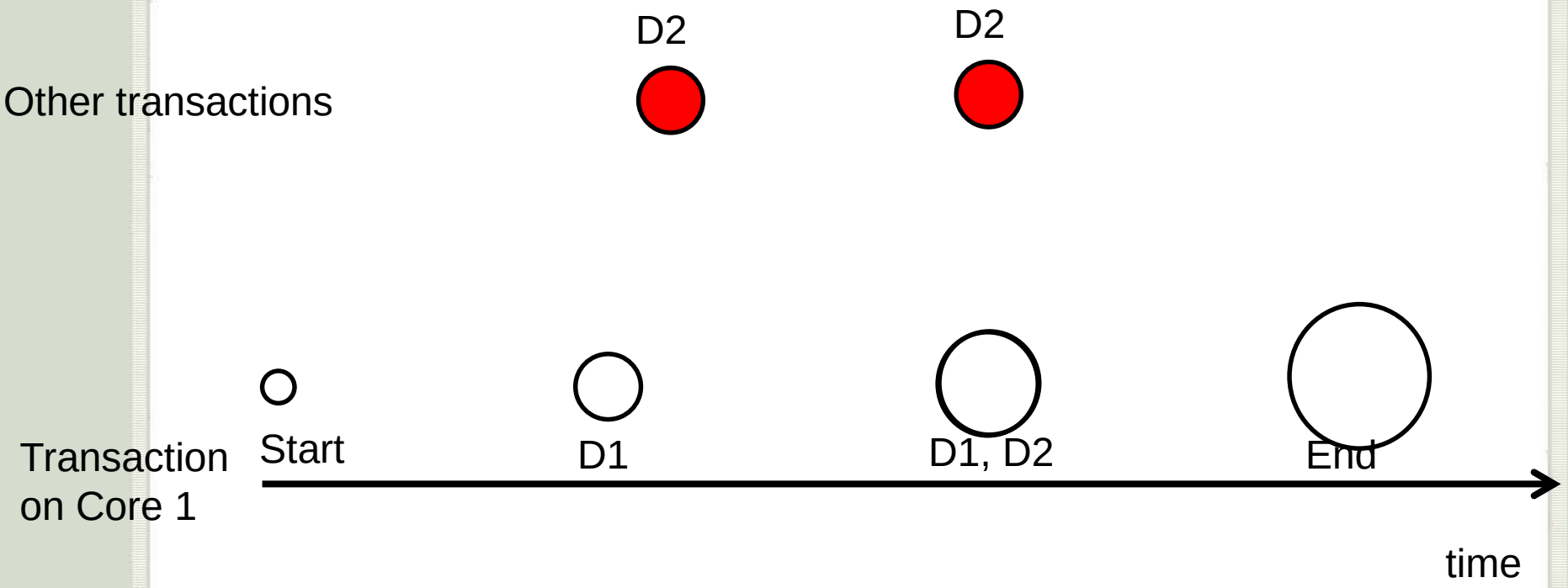
Transactions



Restart Optimization

- Is it possible not to restart to the beginning of a transaction?
- If possible what is a place in the transaction where can be safely restarted to?
- Instead of restarting from the start, the transaction restarts from the last valid state. The last valid state corresponds to the context of the transaction just before the first access to the shared data that caused the restart.

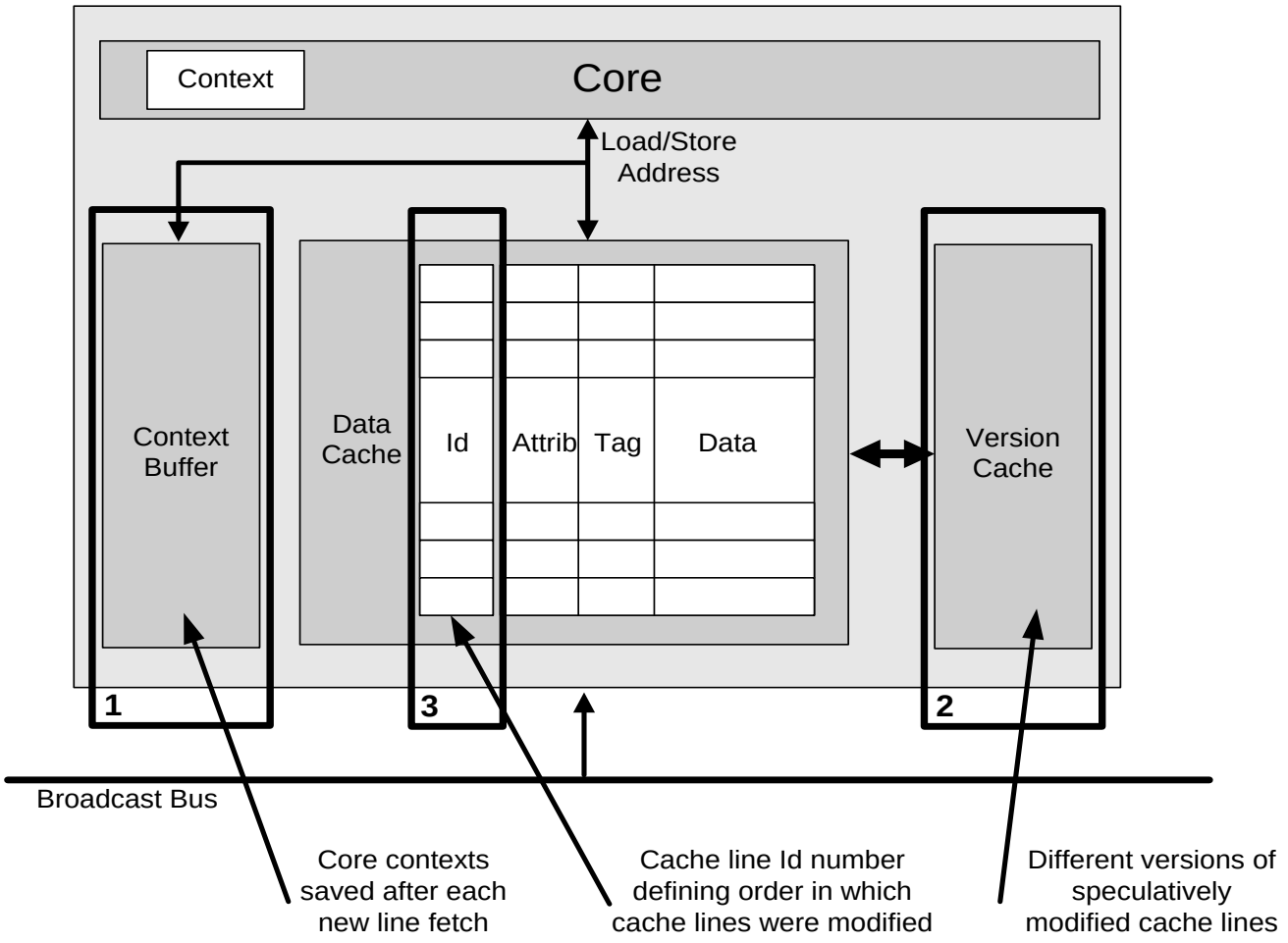
Transactions



Transactional Memory

- *What is a context to be saved?*
  - *Core context*
  - *Cache context*
- *When to save context?*
  - *Core context – when accessing date for the first time*
  - *Cache context – when changing data*
- *Does saving context slows down transactions?*
  - *Core context – first time access = Cache Miss!*
  - *Cache context – Parallel write*

Transactional Memory



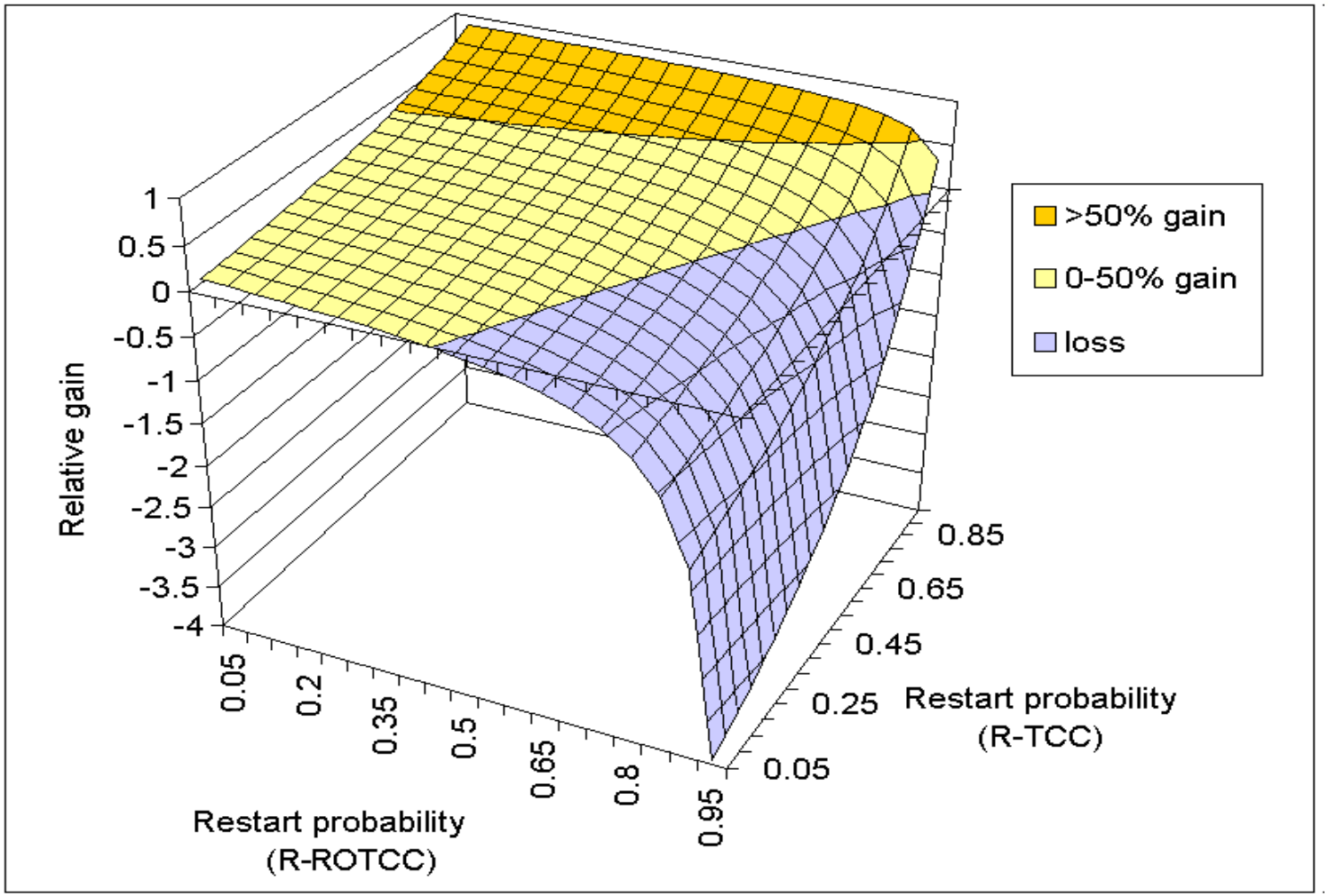
Model

$$F_{nr}(x) = \int_0^x \sum_{n=0}^{+\infty} e^{-\bar{n}} \cdot \frac{(t\frac{K}{L})^n}{n!} \cdot \frac{(U-n)!\cdot(U-K_w)!}{U!\cdot(U-n-K_w)!} dt$$

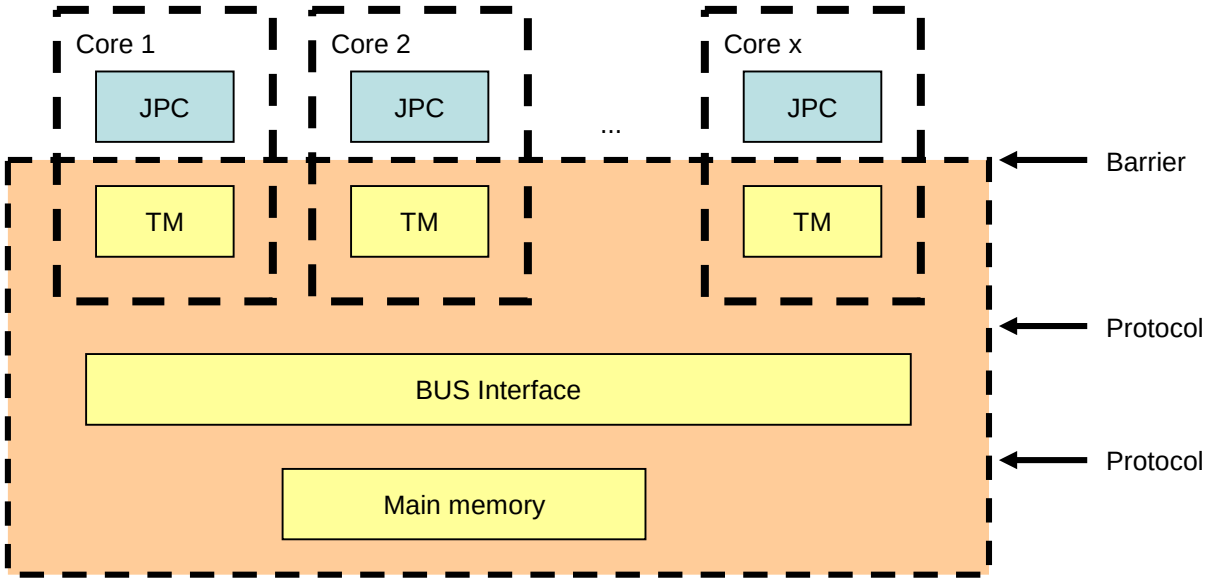
$$R = 1 - e^{-\frac{(N-1)}{\frac{E(T)}{L}+\frac{V}{L}}\cdot\left(1-\frac{F_{nr}(L)}{L}\right)}$$

$$E(T) = \frac{(2-R)^2}{(1-R)\cdot(4-R)} \cdot L$$

Model


Relative gain
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
0.05
0.2
0.35
0.5
0.65
0.8
0.95
Restart probability
(R-ROTCC)
0.05
0.25
0.45
0.65
0.85
Restart probability
(R-TCC)
>50% gain
0-50% gain
loss

Simulation



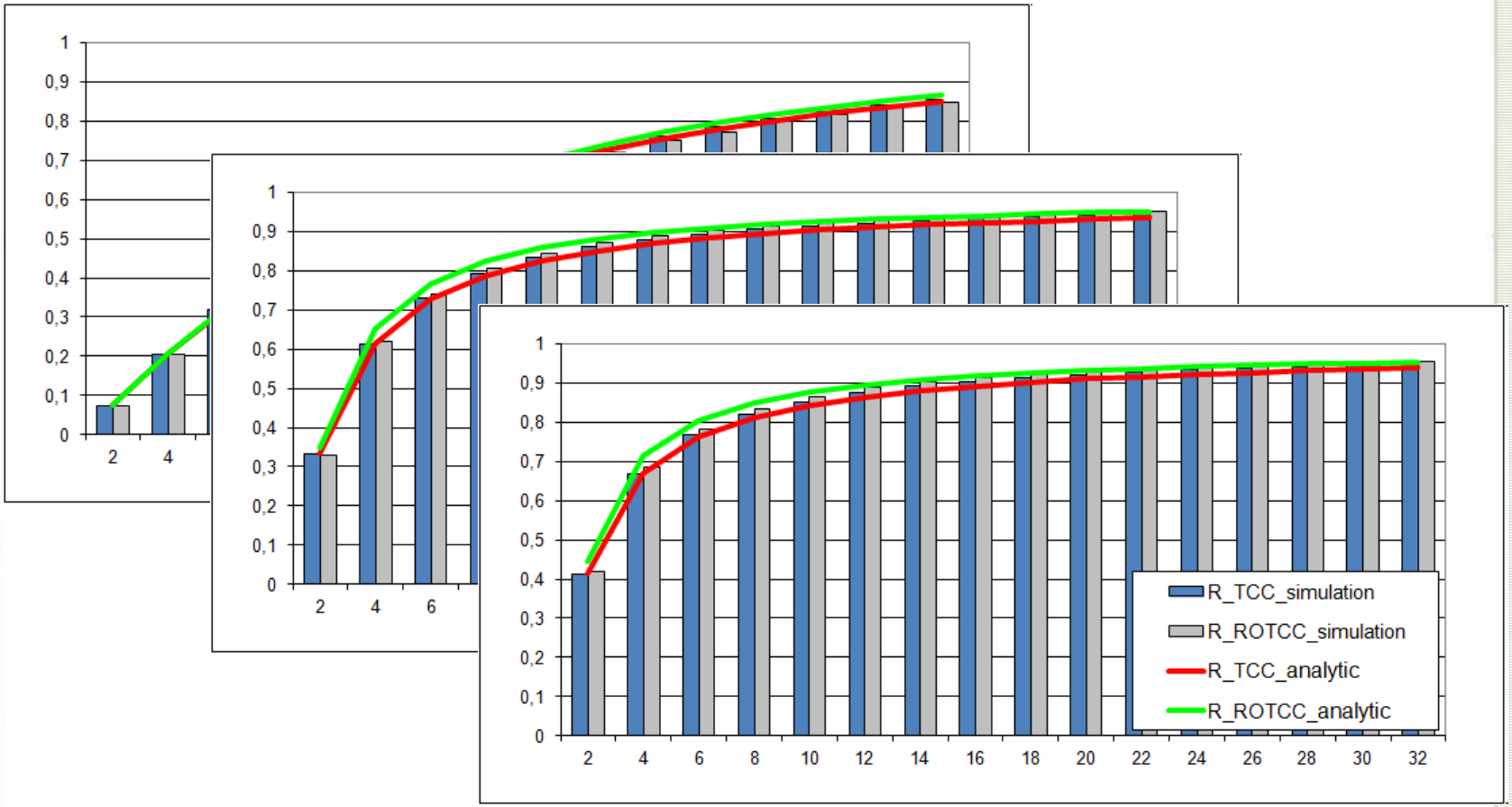
RTL simulation

Model and Simulation

Parameter	Name	Typical range *	Default value **
Time outside of a transaction	V/L	0.03-32.3	0.1
Access set	K	20-800	600
Read set	K_r	10-800	400
Write set	K_w	10-800	300
Number of memory accesses	B	15-220000	6000
Working set	U	10000-3000000	40000
Write probability	P_w	0.05-0.49	0.3
Number of cores	N	2-32	4

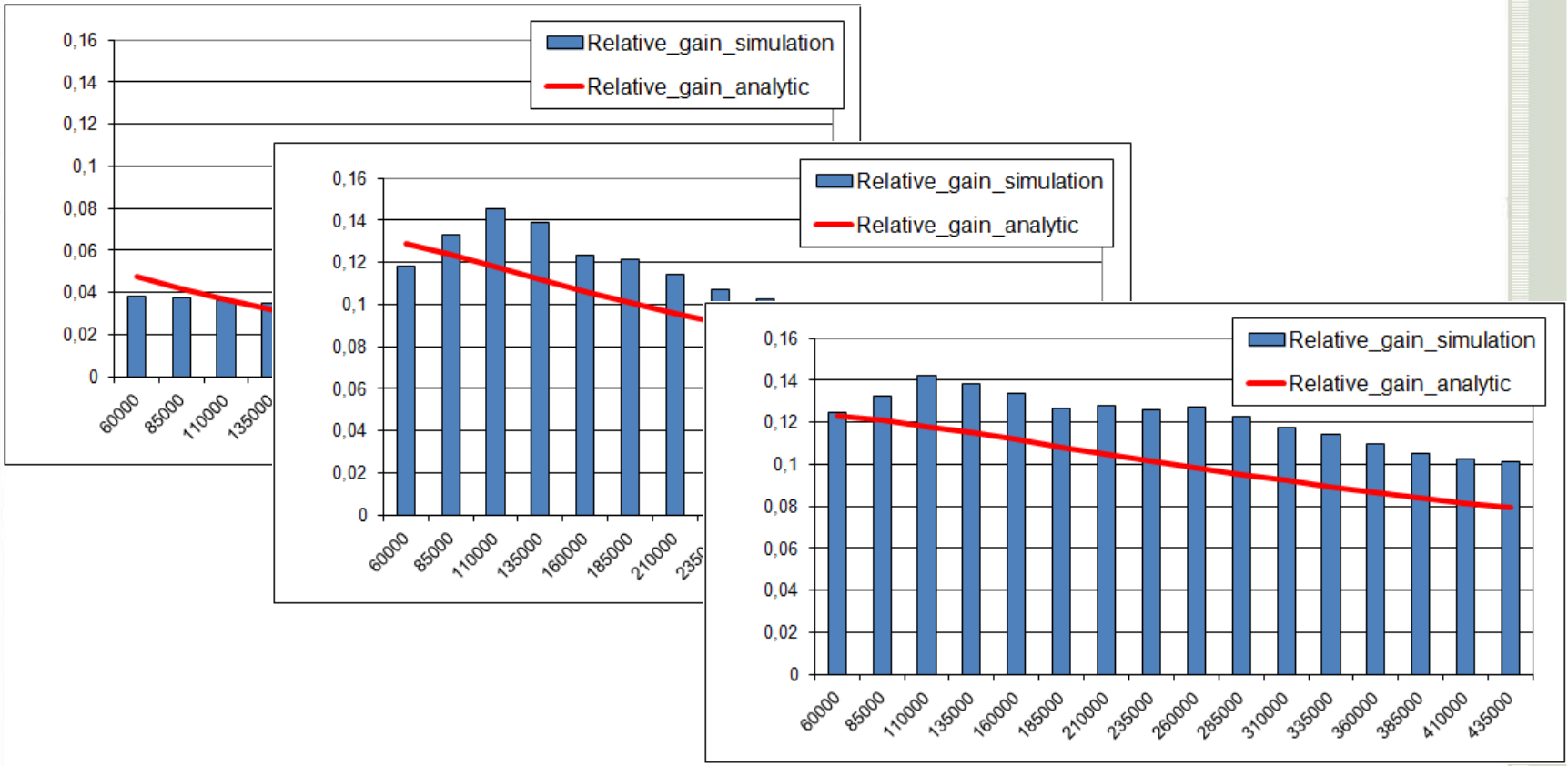
Simulation parameters

Model and Simulation



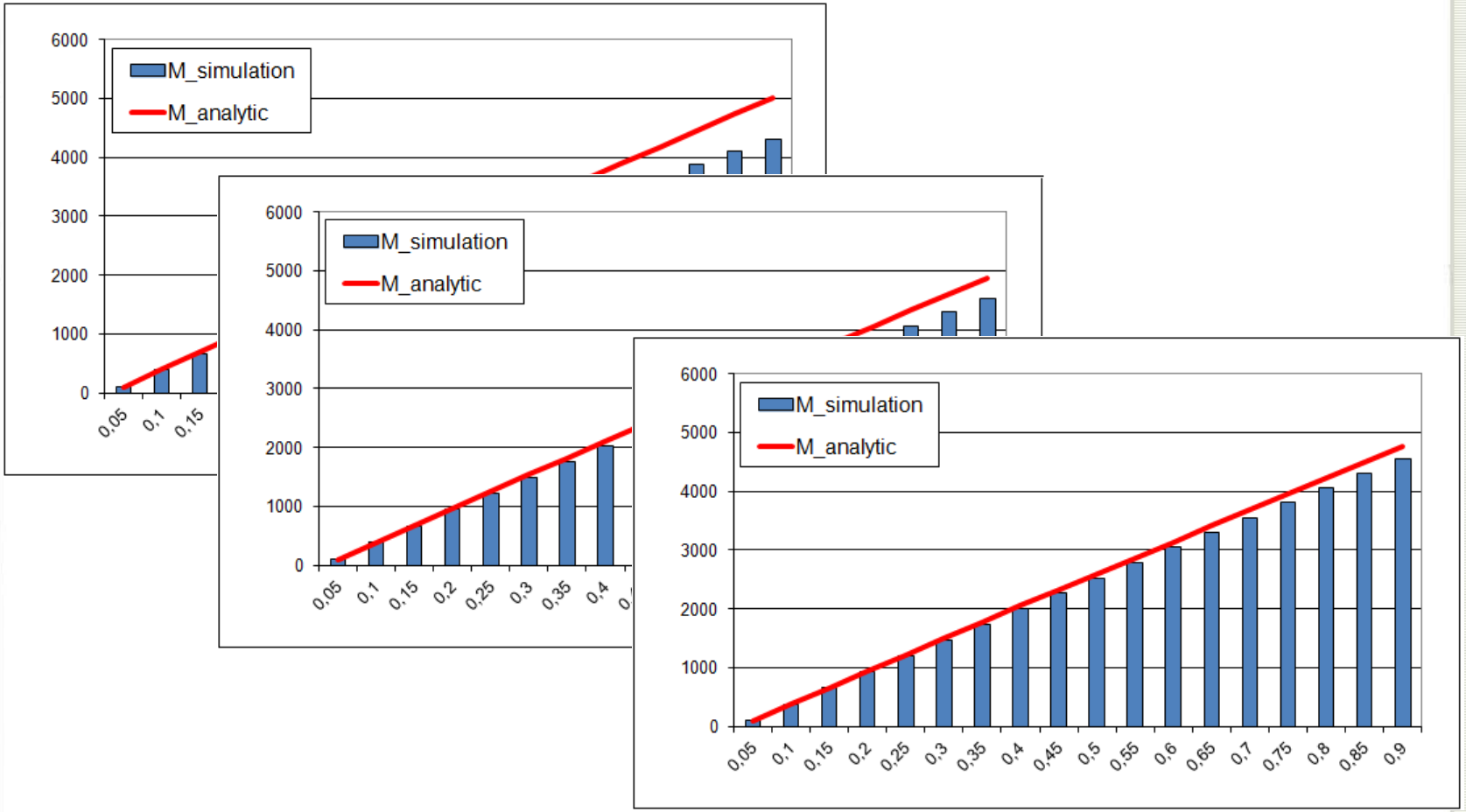
Results for restart probability depending on the number of cores

Model and Simulation



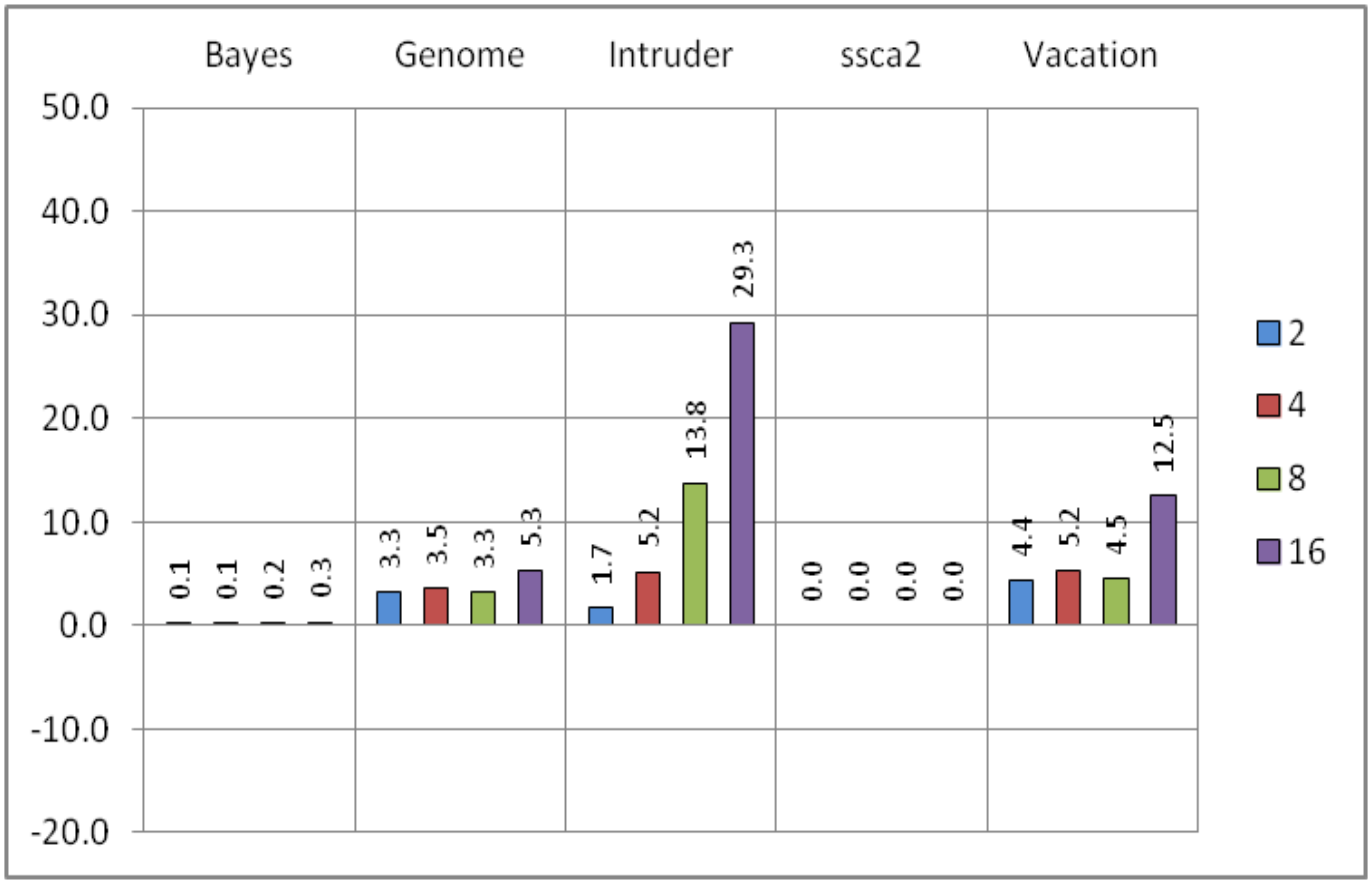
Results for relative gain depending on the working set size

Model and Simulation



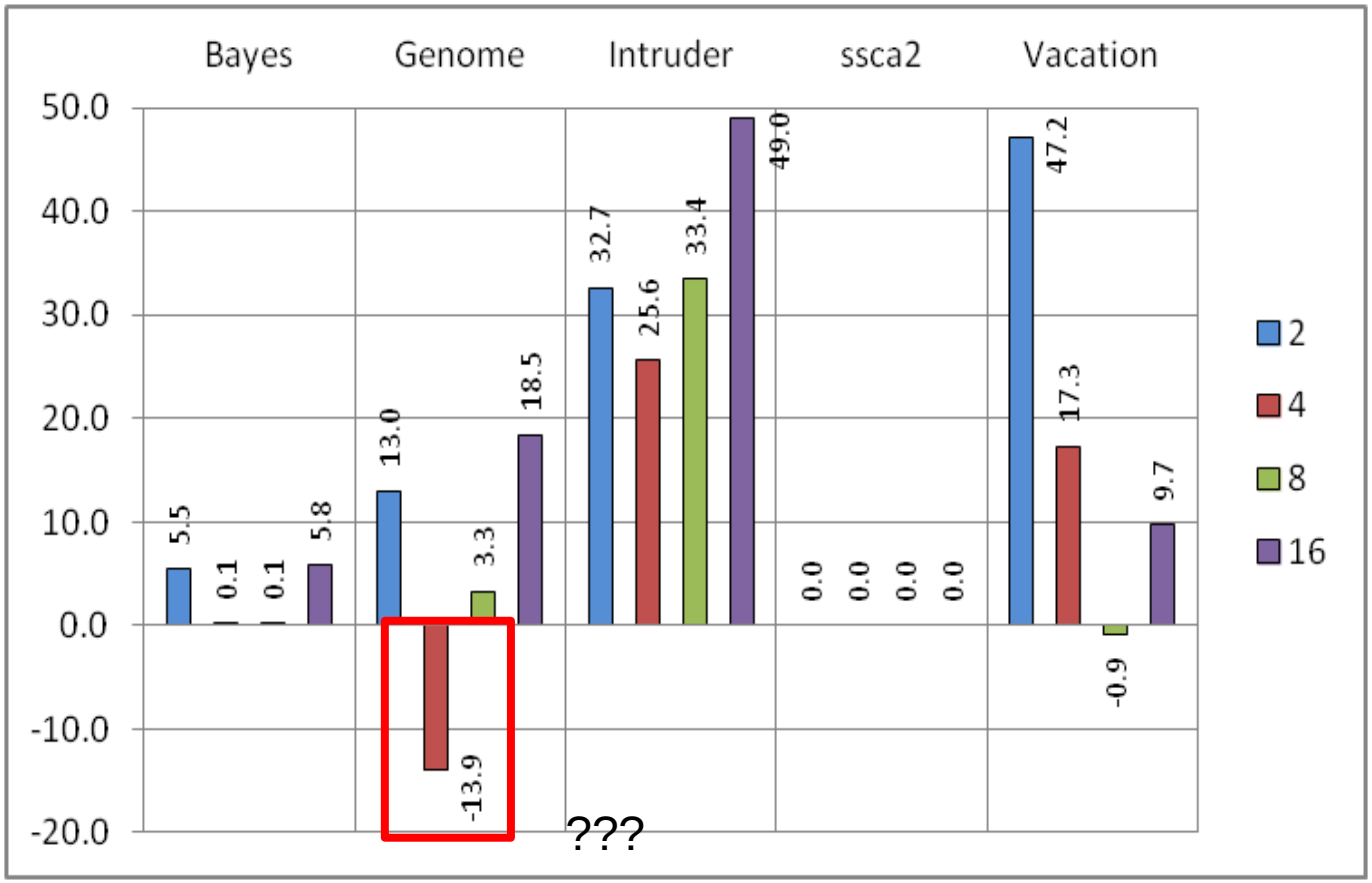
Results for results for the required space depending on the write probability

Model and Simulation



execution time spent in transactions

Model and Simulation



execution time spent in transactions that restarted at least once

Conclusion

- The simulation results indicate that the optimization does not have a significant influence on the value of the restart probability.
- The expected relative gain for transactions that have been restarted at least once is a monotonically increasing function with values ranging from 0.17 to 0.33.
- The generality of the presented approach allows its applicability to other types of transactional memory with lazy version management regardless of the conflict detection strategy.

Thank you!

Radivojevic Zaharije