Per Stenstrom and M. M. Waliullah

Department of Computer Science & Engineering
Chalmers University of Technology
S-412 96 Goteborg, Sweden

Outline

- Background on transactional memory (TM)
- Research on TM at Chalmers
- Example: A TM protocol with subtransactions
- Conclusions

Background:
Transactional Memory

Example

Thread 1 | **Thread 2** | **Thread 3** | **Thread N**

Thread 1:

```
If cond1 then
    count1++

If cond2 then
    count2++

If condN then
    countN++
```

Thread 2:

```
count1++
```

Thread 3:

```
count2++
```

Thread N:

```
countN++
```

Problem:

Simultaneous incrementing of counters by multiple threads may violate the assumed atomicity

Example (cont'd)

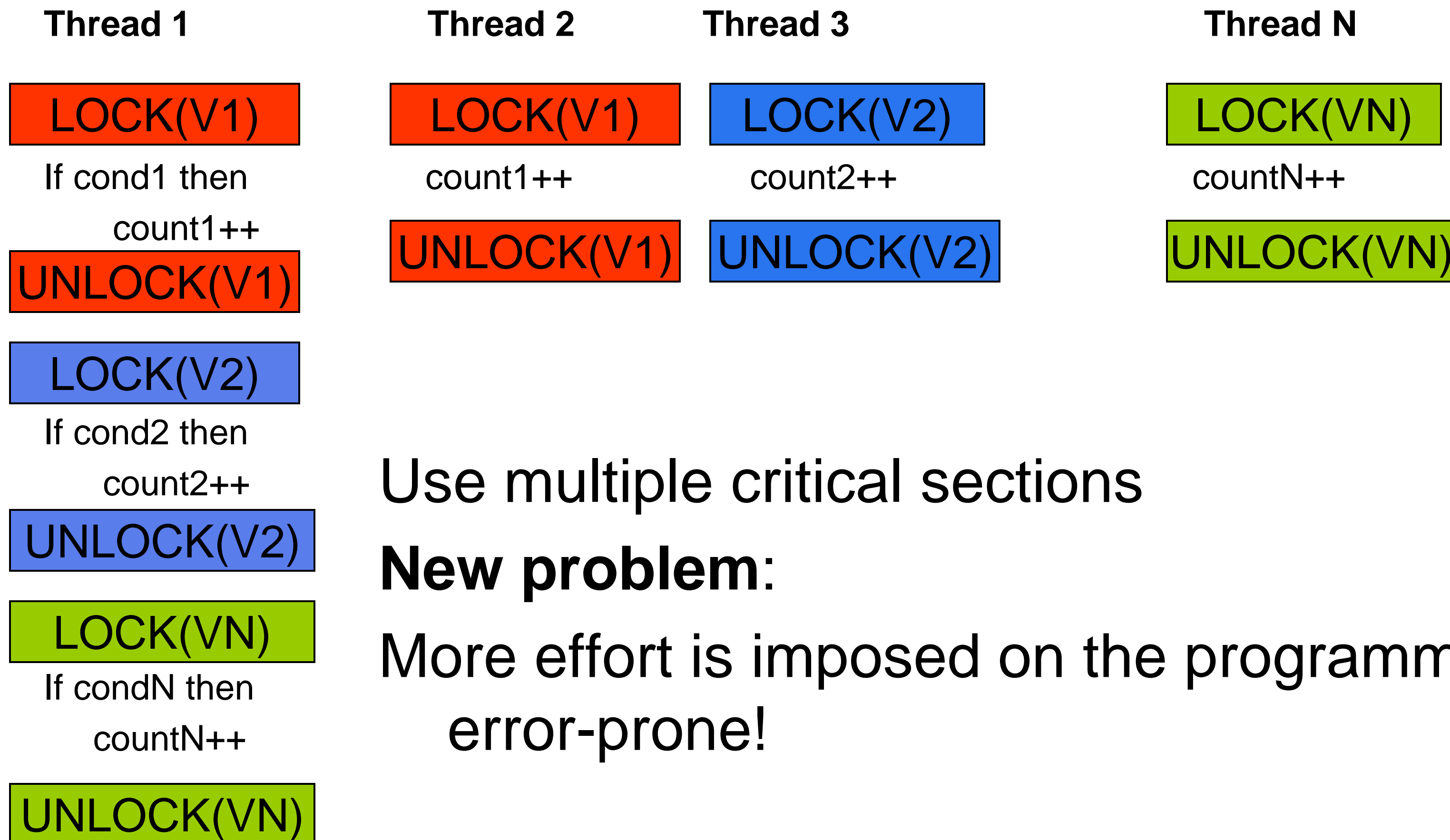
Thread 1	Thread 2	Thread 3	Thread N
LOCK(V1)	LOCK(V1)	LOCK(V1)	LOCK(V1)
If cond1 then count1++	count1++	count2++	countN++
	UNLOCK(V1)	UNLOCK(V1)	UNLOCK(V1)
If cond2 then count2++			
If condN then countN++			
UNLOCK(V1)			

Use a single critical section

New problem:

May cause serialization even when there are no simultaneous accesses

Example 1 (cont'd)



Use multiple critical sections

New problem:

More effort is imposed on the programmer – error-prone!

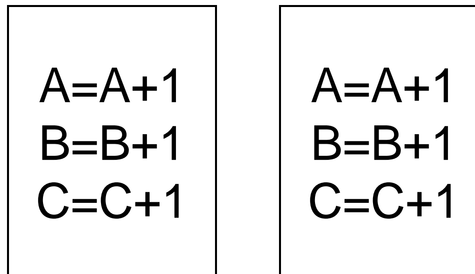
A transactional code block

```
A=A+1
B=B+1
C=C+1
```

The code block executes atomically and in isolation

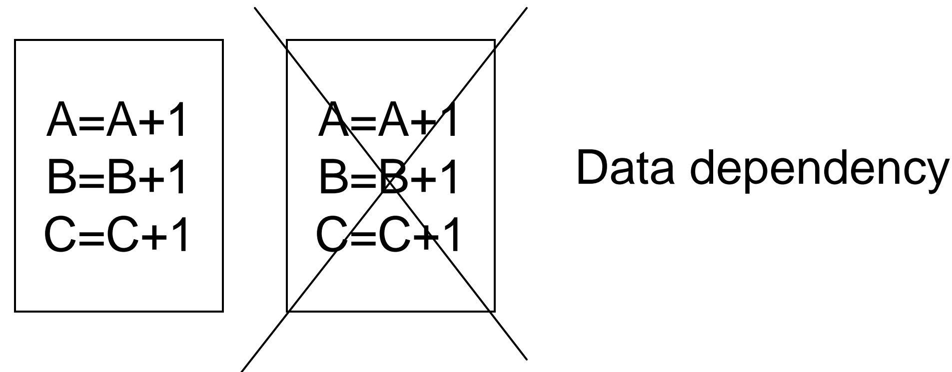
- A, B, C are all updated atomically with respect to other threads

Two transactional code blocks



Assume two transactions are executed in parallel

Two transactional code blocks



The TM system detects dependence violations and selects one conflicting transaction for re-execution

Two transactional code blocks

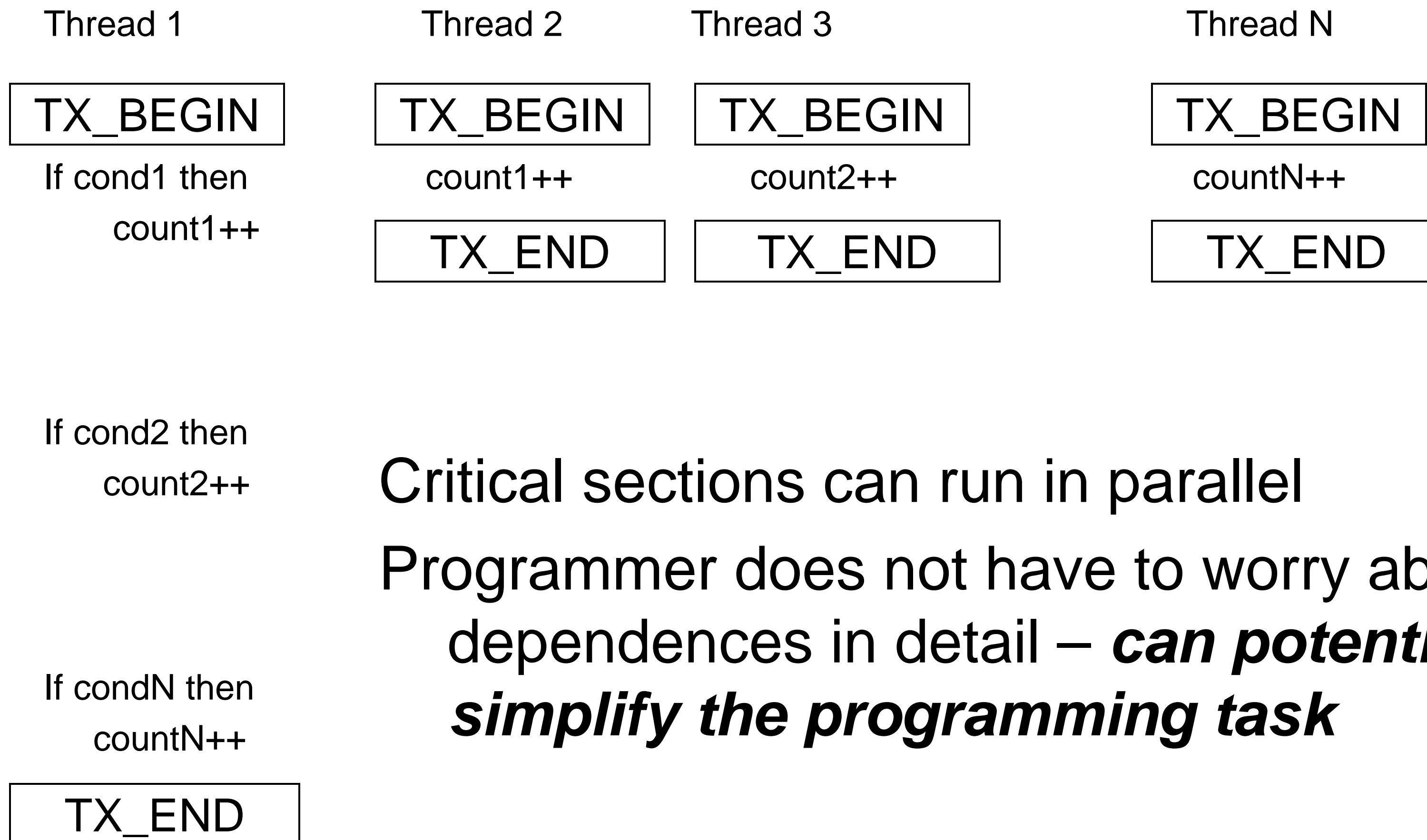
A=A+1
B=B+1
C=C+1

A=A+1
B=B+1
C=C+1

Re-execute to maintain transactional semantics

How can transactions reduce programming effort?

”Transactified” example



Critical sections can run in parallel

Programmer does not have to worry about dependences in detail – ***can potentially simplify the programming task***

Outline

- Background on transactional memory (TM)
- Research on TM at Chalmers
- Example: A TM protocol with subtransactions
- Conclusions

TM challenges

- TM looks promising, but:
  - Very little experience with it
  - Not known yet how to implement it efficiently
- Programmers have to be experts in computer architecture

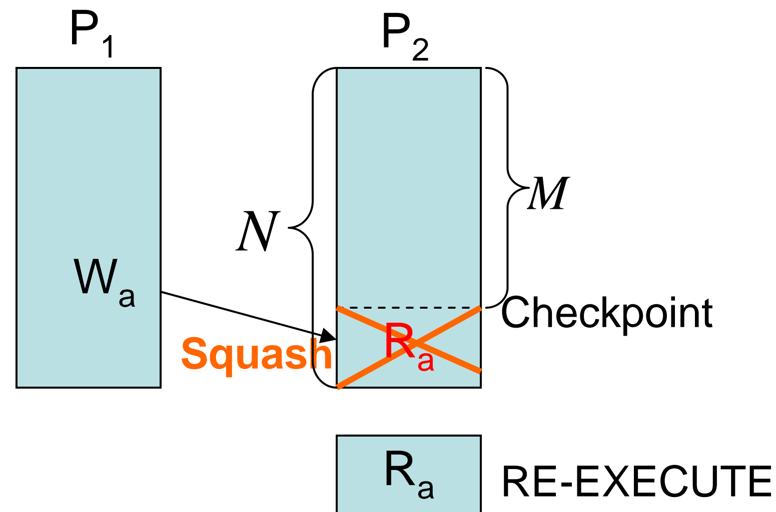
What is needed is a fresh new HW/SW interface!

That's what my group wants to contribute to

Outline

- Background on transactional memory (TM)
- Research on TM at Chalmers
- Example: A TM protocol with subtransactions
- Conclusions

Problem: Late Conflicts



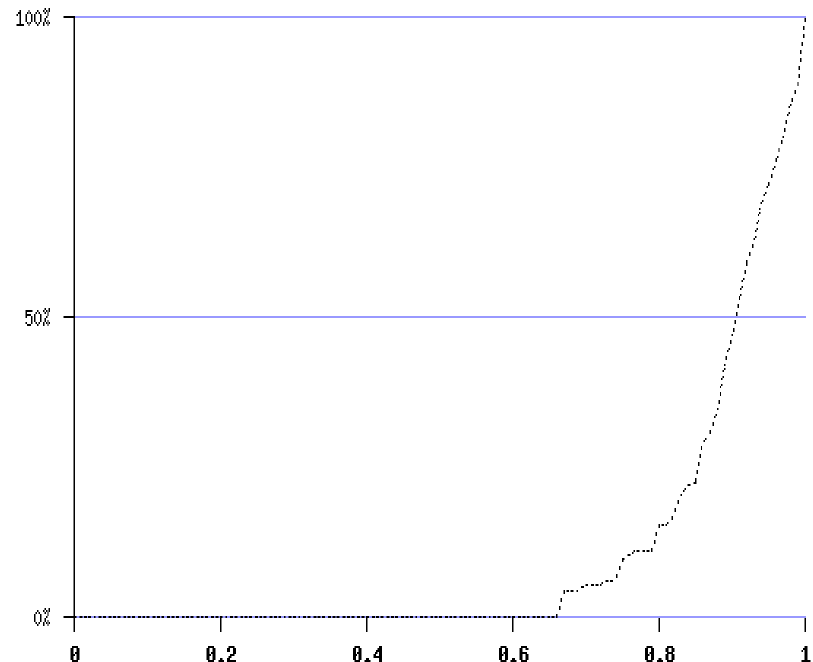
- N – Total transaction execution time
- M – Execution to salvage
- Trading off re-execution overhead of M for checkpointing overhead

Our approach:

- Subtransactions "under the hood" to minimize losses
- Yields higher gain if conflicting access occurs late – esp. for long-running transactions

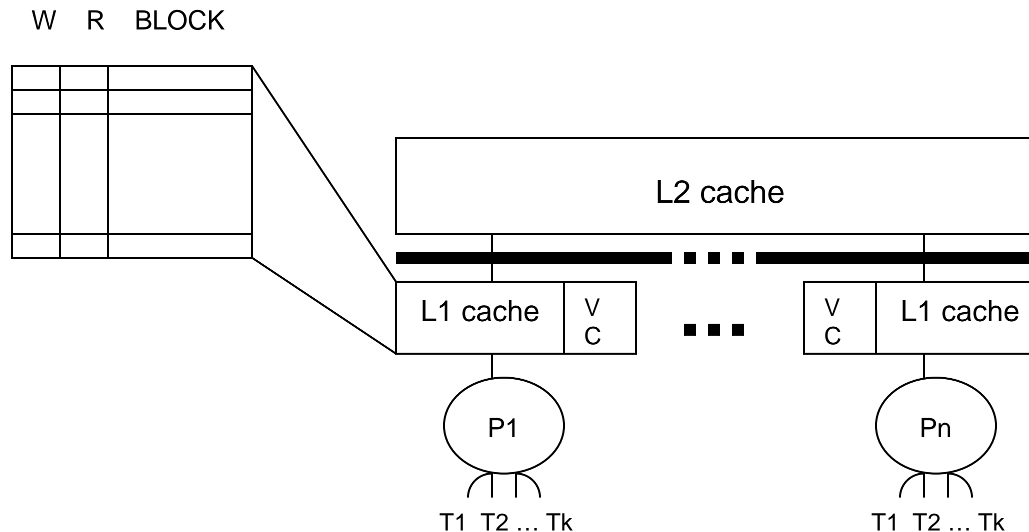
Motivation: Conflicts in a Micro-Benchmark

```
worker(low, high)
{
    for (i=low; i<high; ++i){
        Trans_Begin();
        for(j=0; j<n_vowels; ++j){
            if(str[i] == vowels[j]){
                ++count;
                break;
            }
        }
        Trans_End();
    }
}
```



- The only shared variable '*count*' is incremented at the end of each transaction
- The conflicting access happens when at least 65% of the transaction has been executed.

Baseline TM System



- **Maintains *isolation* by keeping modifications locally**
- **Commits involve notifying others which blocks are modified**
- **Conflicting transactions are *squashed* and re-executed – <u>may lead to execution *losses.*</u>**

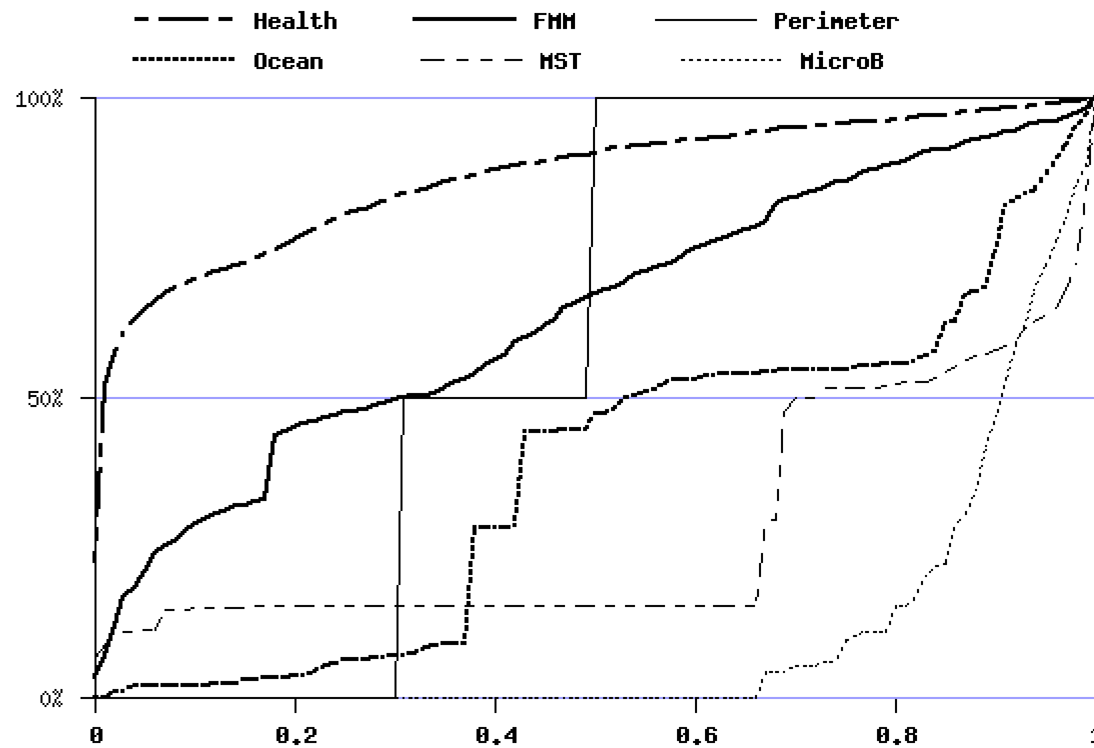
Basic idea

1. Each node collects *conflicting addresses* from previous transactions
2. Every memory access is compared with the set of conflicting addresses
3. On a match, an intermediate checkpoint is taken and a new subtransaction is started
4. On a conflict, the transaction rollbacks to the checkpoint that precedes the earliest conflicting access

Experimental Methodology

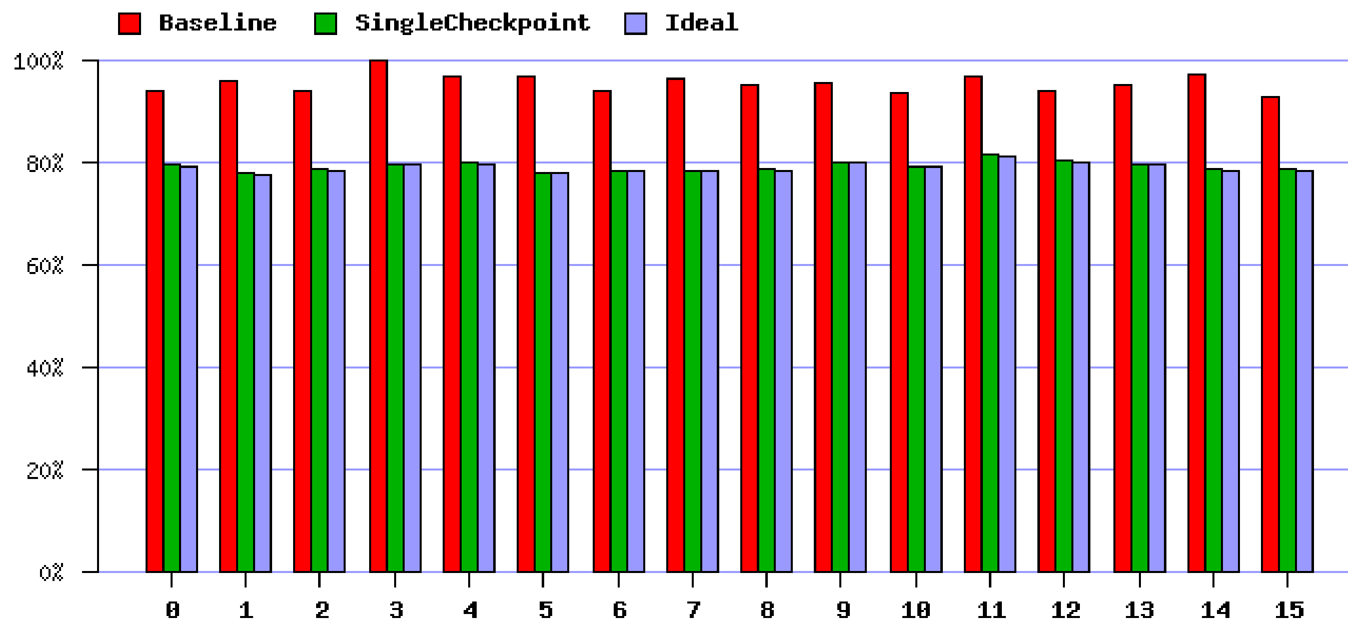
- Three systems are modeled:
  1. Baseline system that emulates TCC
  2. An Ideal scheme that checkpoints every memory access
  3. Intermediate check-pointing scheme with a maximum of four checkpoints
- Trace-driven approach based on three Olden and two SPLASH-2 applications
- Transactification methodology:
  - Replaces critical sections and epochs with transactions

Execution Loss Distribution



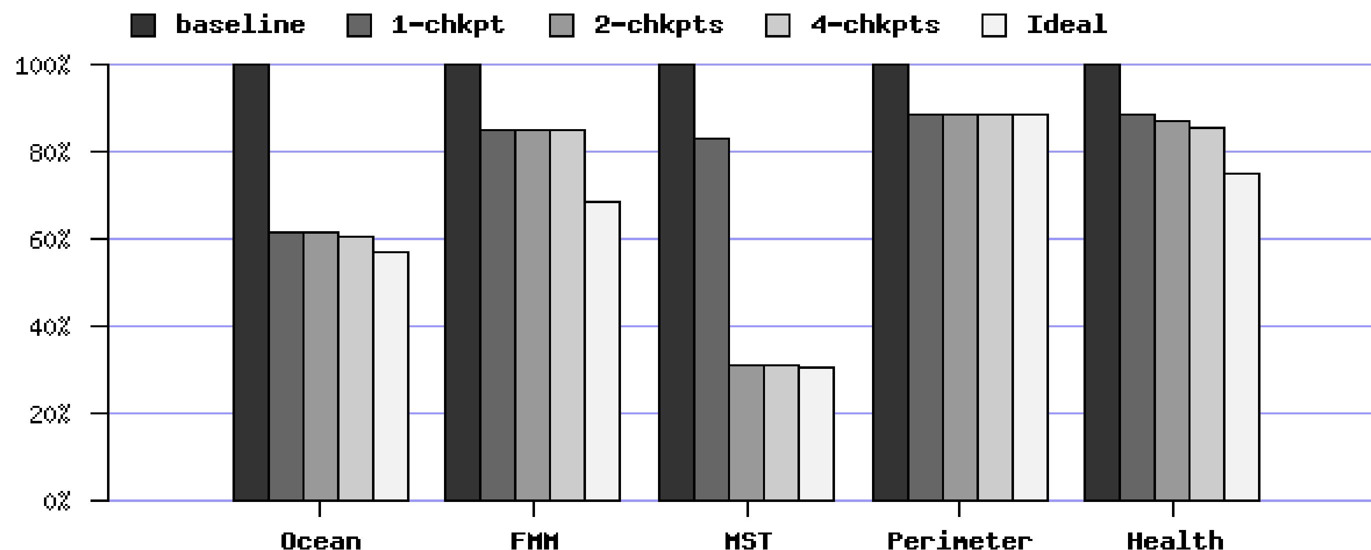
Cumulative distribution of the fraction $\frac{M_i}{N_i}$

Reduction of Losses for the Micro Benchmark



- **Using a single check-point is as efficient as in the ideal scheme**

Performance of the other Applications



For the 2-checkpoint scheme:

- **It performs as well as the Ideal scheme for Ocean, MST and Perimeter**
- **Half of the gains are reaped in FMM and Health**
- **Between 16% and 70% of execution time is reduced for all applications**

Concluding Remarks

- Transactional memory may yield execution losses, especially for long-running transactions
- We have contributed with a *new* TM protocol:
  - Prediction-based insertion of subtransactions to reduce execution losses on roll-backs
- Transactional memory is a bit hyped; future research will show whether it lives up to its promises.