University of Karlsruhe (TH)

Research University · founded 1825

Technical Briefing Session „Multicore Software Engineering“ @ ICSE 2009

Transactional Memory versus Locks - A Comparative Case Study

Victor Pankratius

Faculty of **Computer Science**
Head of Young Investigator Group
www.ipd.uni-karlsruhe.de/~pankratius

updated Sep 2, 2009

Traditional Parallel Programming

- Synchronize critical sections using locks
  - Explicit lock / unlock
- Claimed to to be advantageous for performance
- Problems
  - Very low-level
  - Error-prone
  - Burden on developers

Transactional Memory

- Instead of explicit locks, use **atomic transactions**

  ```
  atomic { /*critical section code*/ }
  ```

- A run-time system allows threads to execute atomic blocks concurrently, while making it appear that only one thread at a time executes within an atomic block.
  - Intention: relieve developer from locking details, esp. in situations with many locks and complex locking protocols
  - A transaction is aborted and re-executed if it conflicts with another transaction (operations must be reversible)
  - Run-time system ensures atomicity, consistency, isolation
- **Sounds promising in theory, but…**

Predjudices Against TM from the Literature

- Transactional Memory is **slow**
- Transactional Memory is only a **research toy**
- Transactional Memory **may not be applicable** to more complex, non-numerical programs
- Transactional Memory **does not offer any real benefits** for parallel software development

Based on the empirical results of this study, I want to show you that this is not generally true.

Traditional Approaches of Evaluating TM

- Small (numerical) programs
- Micro-benchmarks
- Translating lock-based programs into TM
- Mostly worst-case analyses

...is this enough?

About the Case Study

- **12 students, semester** project in C
  parallel desktop search engine
  from scratch (indexing and search)

- **Competition:** best performance, given target features
  - 3 teams randomly assigned to use Pthreads (i.e., locks)
  - 3 teams Phreads + TM (i.e., transactions)
  - Realistic scenario: just end product matters. Students were allowed to re-use any code from Web, employ different strategies and data structures

- Questions
  - What happens? E.g., performance, code, effort, difficulties

About the Case Study

- Cooperation with Intel
  - Provided Software Transactional Memory compiler
  - Most advanced STM compiler so far
  - Based on Intel's widely-used C compiler

- Initially, 3 weeks teaching for all students
  - Parallel programming, search engine technology
  - Pthreads library, Transactional Memory (using Intel‘s STM compiler)

- 6 teams randomly created (2 students each)

Performance Summary

- **Indexing time**
  - **Locks winners** (team 5) : 605 s with 4 threads
  - **TM winners** (team 6) : 178 s with 7 threads
- **18 types of queries**
  - TM teams faster than locks teams on 9 out of 18 queries

- Determining winners:
  - Equally weighted score for indexing and query time of correct queries
  - Benchmark: 51,000 text files, 742MB, 8 core machine

- TM winners combined TM with a few semaphores for producer-consumer synchronization and outperformed team 5 on indexing and search.
- TM team 2 (one of the most inexperienced teams) was excluded; program crashed on benchmark

Program Sizes

Total LOC are about the same....

	Pthreads Teams			TM Teams		
	Team 1	**Team 4**	**Team 5**	**Team 2**	**Team 3**	**Team 6**
Total LOC	2014	2285	2182	1501	2131	3052
	avg: 2160, stddev: 137			**avg: 2228, stddev: 780**		
Total LOC with Parallel Constructs	157 8%	261 11%	120 5%	53 4%	45 2%	151 5%
	avg: 179, stddev: 73			**avg: 83, stddev: 59**		

..but TM teams have fewer LOC with parallel constructs

Remarks: Team2: incomplete program,
Team 6: implemented many low-level library functions
(to ensure that they worked with TM)

Effort

- Collected data on a day-by-day basis
- Students kept track of hours

1-reading doc
2-search for libs
3-conceptual design
4-implementation
5-experiments
6-testing
7-debugging
8-other

Team 1		1	2	3	4	4.1	4.2	4.3	4.4	5	5.1	5.2	5.3	5.4	6	6.1	6.2	6.3	6.4	6.5	7	7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	8
Sum of person hours	149,5	5,5	3	9	79,5	46,5	30	1	2	10	5	2	3	0	13,5	6	5,5	0	2	0	29	16	4	2	0	0	3	0	4	0
	100%	4%	2%	6%	53%					7%					9%						19%									0%
Mo, 20.10.08	0				0					0					0						0									
Di, 21.10.08	0				0					0					0						0									
Mi, 22.10.08	0				0					0					0						0									
Do, 23.10.08	0				0					0					0						0									
Fr, 24.10.08	0				0					0					0						0									
Sa, 25.10.08	0				0					0					0						0									
So, 26.10.08	0				0					0					0						0									
Mo, 27.10.08	0				0					0					0						0									
Di, 28.10.08	0				0					0					0						0									
Mi, 29.10.08	0				0					0					0						0									
Do, 30.10.08	0				0					0					0						0									
Fr, 31.10.08	0				0					0					0						0									

Effort in Person Hours

1-reading doc
2-search for libs
3-conceptual design
4-implementation
5-experiments
6-testing
7-debugging
8-other

Less for TM

	1	2	3	4	5	6	7	8	total
Team 1 (L)	6	3	9	80	10	14	29	0	151
Team 5 (L)	24	1	17	72	7	52	16	19	208
Team 4 (L)	29	12	14	196	12	21	48	2	334
Team 6 (TM)	18	4	12	55	6	18	19	9	141
Team 2 (TM)	7	6	33	74	18	38	22	10	208
Team 3 (TM)	6	6	21	139	12	39	38	0	261
sum all	**90**	**32**	**106**	**616**	**65**	**182**	**172**	**40**	**1303**
	7%	2%	8%	47%	5%	14%	13%	3%	100%
sum L	**59**	**16**	**40**	**348**	**29**	**87**	**93**	**21**	**693**
	9%	2%	6%	50%	4%	13%	13%	3%	100%
sum TM	**31**	**16**	**66**	**268**	**36**	**95**	**79**	**19**	**610**
	5%	3%	11%	44%	6%	16%	13%	3%	100%
sum L - sumTM	*28*	*0*	*-26*	*80*	*-7*	*-8*	*14*	*2*	*83*

Winners
(delta: 67h)

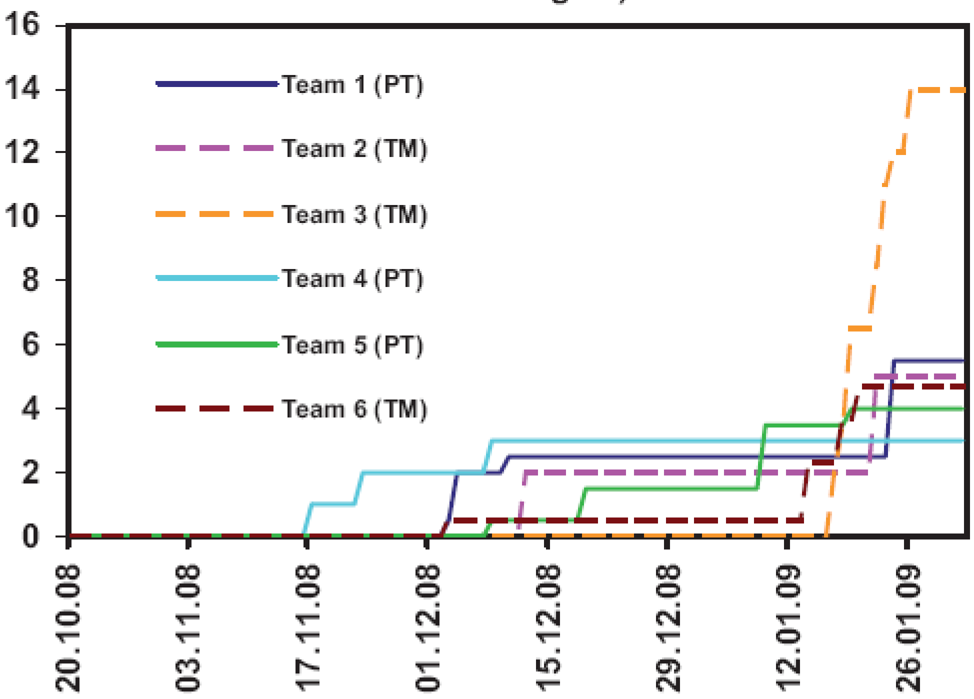
Implementation Effort

	Sequential Code	Parallel Code	Refactoring	Other	
Pthreads teams	**168**	**121**	**50**	**9**	**348**
	48%	35%	14%	3%	100%
TM teams	**173**	**65**	**17**	**13**	**268**
	65%	24%	6%	5%	100%

TM teams spent more time on sequential code, less on parallel code.

TM Performance Is Not Well-Understood

6.2) Accumulated hours for performance tests (of search engine)



„Late restructuring problem“

- TM teams run into performance problems at the end of the project
- TM performance hard to predict
- Atomic sections too long, needed restructuring (team 3)

Code Inspections

- Done by myself and Ali Adl-Tabatabai's STM compiler team at Intel
- Parallel code of TM teams was easier to understand
  - TM teams had comparable functionality, but fewer critical sections than the locks teams;
    about 12 - 24 atomic blocks
  - Locks teams: up to thousands of locks (!)
- Both winning teams had races detected by inspection

Questionnaire – Psychological Issues

- TM teams apparently had less fear that adding parallel constructs would break their program.
- Yet TM winners tried to postpone parallelization work more often than the locks winners.
  - Personal observations and interviews show this for all TM teams.
- Compared to locks teams, TM teams thought that their programs were more difficult to understand
  - Not true according to code inspections.
- TM winners thought they were not advancing fast enough due to TM.
  - Not true: they had first parallel program and lowest total effort.

Other Results

- First demo of working parallel program presented by TM winners
  - At the beginning of the fifth project week
  - Four weeks earlier than locks winners

- Combination of TM with locks or semaphores worked (team 3 and 6)
  - Usage of TM and locks can be complementary
  - We don‘t have to think of TM or locks as alternatives

Summary

- TM alone is no silver bullet, but combined with lower-level parallel constructs, it can
  - improve quality of parallel code,
  - reduce implementation and debugging effort, and
  - provide acceptable performance.
- If programmers are inexperienced (team 2), then TM does not help them either. Parallel programming remains difficult.
- Better software engineering for TM needed
  - Patterns, library support, program understanding, performance monitoring, debugging
  - TM community needs to address these issues

Thank you for your attention!

- Full technical report is available:

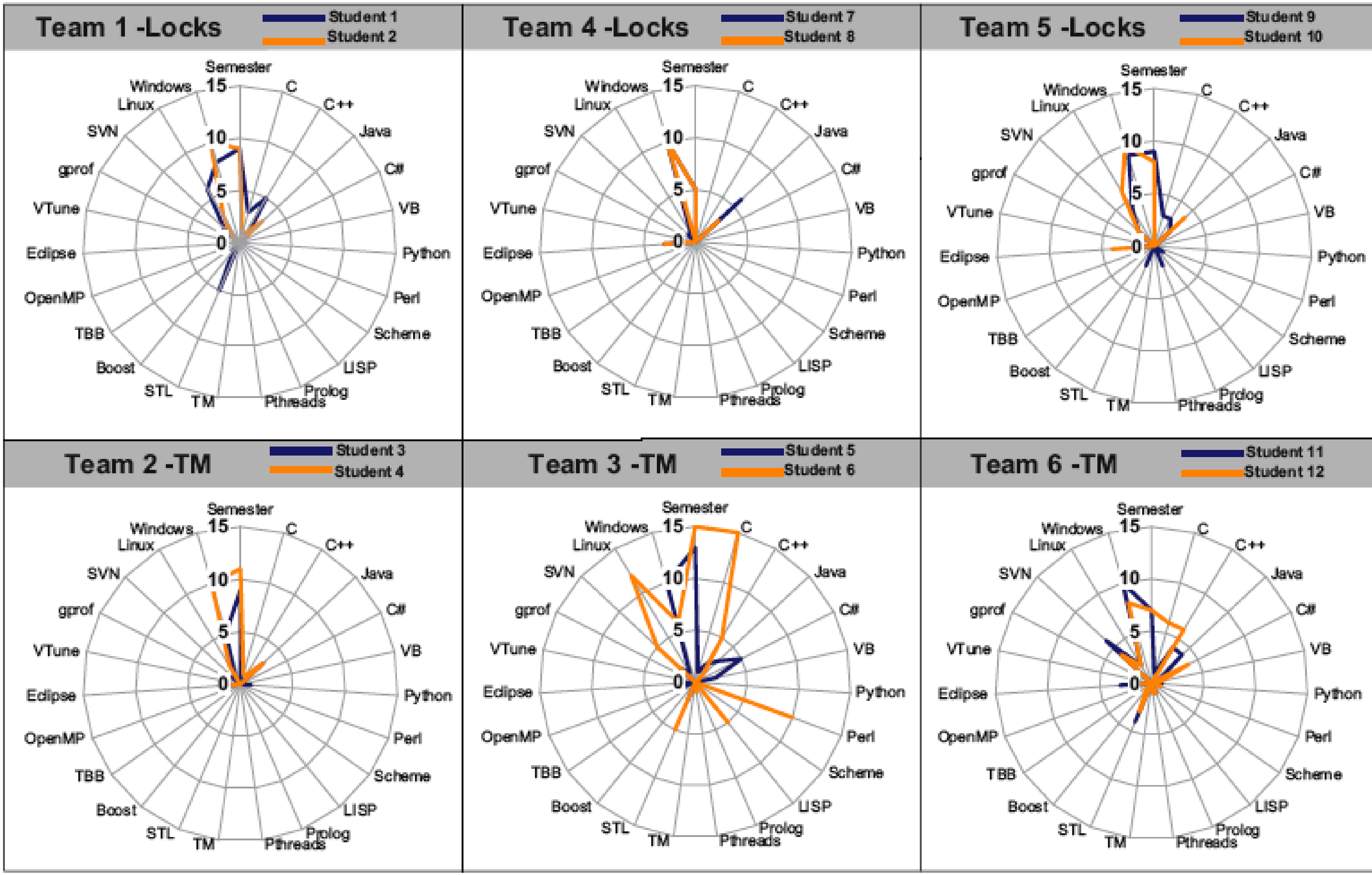
  Victor Pankratius, Ali-Reza Adl-Tabatabai, Frank Otto.
  ``**Does Transactional Memory Keep Its Promises? Results from an Empirical Study.**'', Technical Report 2009-12, September 2009, University of Karlsruhe, Germany

  http://www.rz.uni-karlsruhe.de/~kb95/papers/pankratius-TMStudy.pdf

- I'm happy to receive feedback:

pankratius@ipd.uka.de

Student Experience Prior to Study



Performance

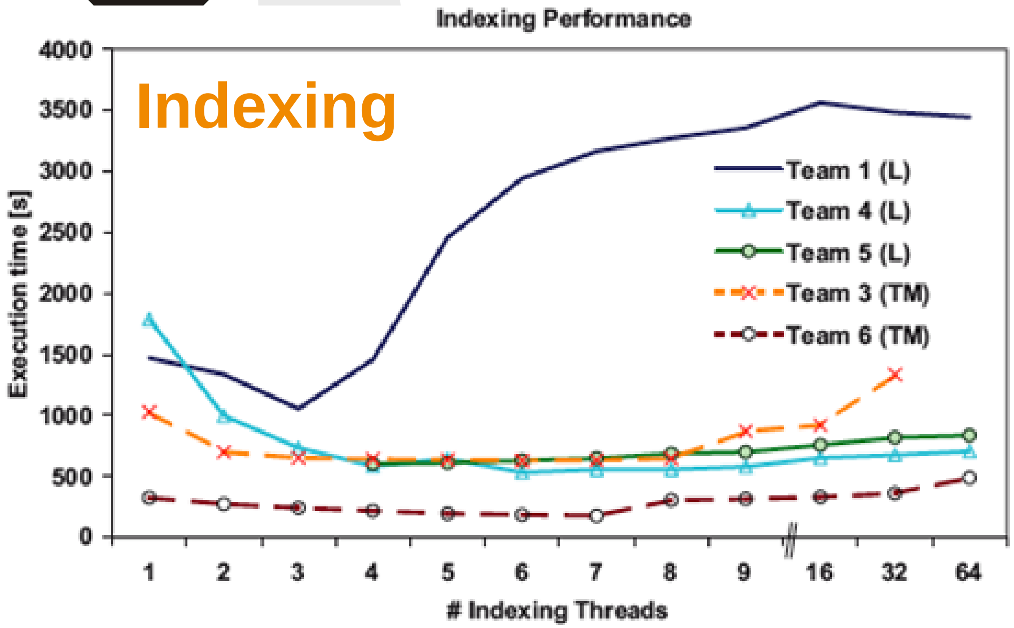
Determining winners: weighted score
- *50% for indexing time*
- *50% for query time of correct queries*
- *Benchmark: 50,887 text files, 742MB*

-Winner for locks: team 5

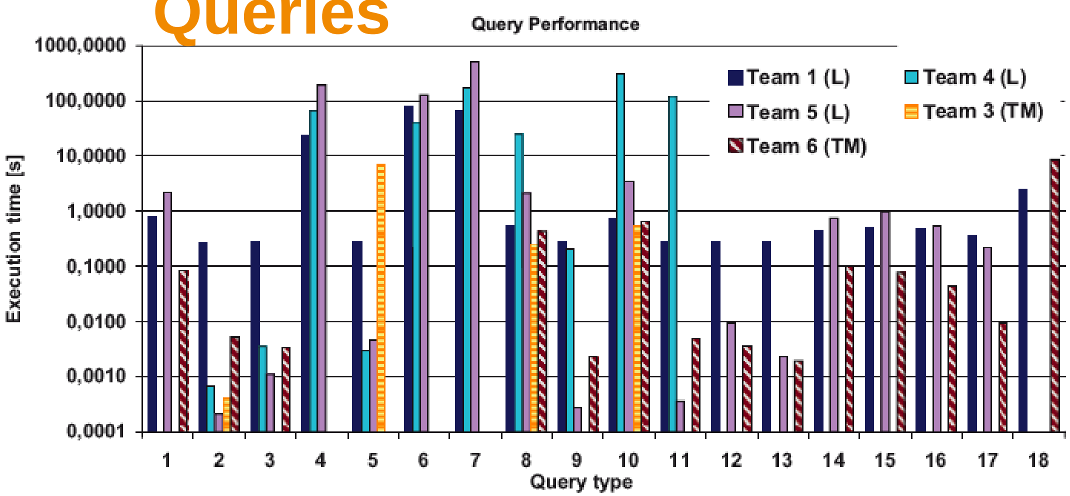
-Winner for TM: team 6
- combined TM with semaphores for producer-consumer synchronization
- outperformed team 5 on indexing and search

Inexperienced team 2: program crashed
→ excluded

Indexing



Queries



	Query type
1	one frequent word
2	one rare word
3	one random word
4	frequent text passage (2 words)
5	rare text passage (2 words)
6	frequent text passage (3 words)
7	rare text passage (3 words)
8	wildcard frequent (word*)
9	wildcard rare (word*)
10	wildcard frequent (*word)
11	wildcard rare (*word)
12	AND frequent (2 words)
13	AND rare (2 words)
14	AND frequent (3 words)
15	AND rare (3 words)
16	exclusion (1 frequent word)
17	exclusion (1 rare word)
18	AND four characters with wildcards

Implementation Effort

	4.1	4.2	4.3	4.4	
Team 5 (L)	35	5	27	5	72
Team 1 (L)	47	30	1	2	80
Team 4 (L)	86	86	22	2	196
Team 6 (TM)	42	7	3	3	55
Team 2 (TM)	25	33	6	10	74
Team 3 (TM)	106	25	8	0	139
sum all	**341**	**186**	**67**	**22**	**616**
	55%	30%	11%	4%	100%
sum L	**168**	**121**	**50**	**9**	**348**
	48%	35%	14%	3%	100%
sum TM	**173**	**65**	**17**	**13**	**268**
	65%	24%	6%	5%	100%

4.1 Implementation of mostly sequential code

4.2 Implementation of mostly parallel code (using Pthreads of TM constructs)

4.3 Refactoring

4.4 Other implementation tasks

TM teams spent more time on sequential code, less on parallel code

Program sizes

Total LOC are about the same....

	Pthreads Teams			TM Teams		
	Team 1	Team 4	Team 5	Team 2	Team 3	Team 6
Total LOC	2014	2285	2182	1501	2131	3052
	avg: 2160, stddev: 137			**avg: 2228, stddev: 780**		
LOC pthread*	157	261	120	17	23	12
LOC tm_*	0	0	0	36	22	139
Total LOC with Parallel Constructs	157 8%	261 11%	120 5%	53 4%	45 2%	151 5%
	avg: 179, stddev: 73			**avg: 83, stddev: 59**		

..but TM teams have fewer LOC with parallel constructs