Adaptive Concurrency Control for Transactional Memory

Using 40% fewer cores without degrading performance

Mohammad Ansari, Christos Kotselidis, Kim Jarvis, Mikel Lujan, Chris Kirkham, Ian Watson
Advanced Processor Technologies Group

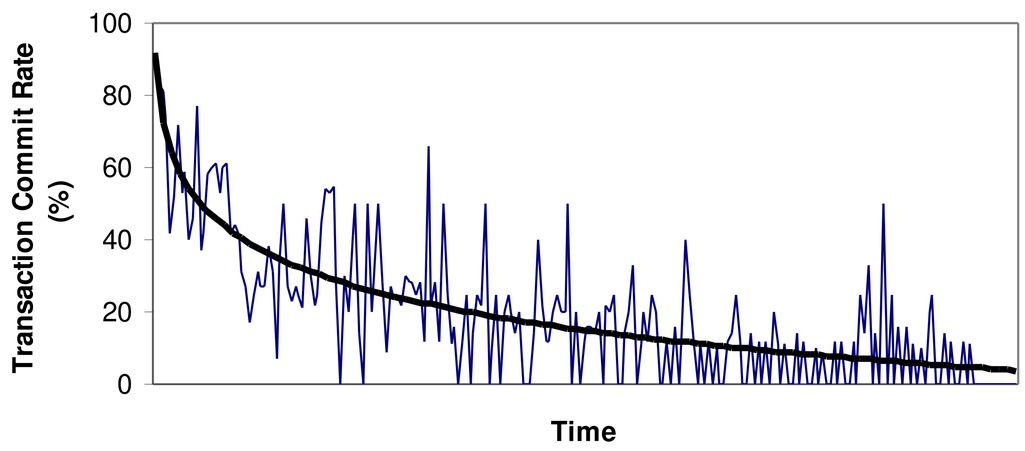
1. What is Transactional Memory?

- **Multi-cores add a new requirement to mainstream programming: parallel and scalable software.**
- **Currently, such software is built using fine-grain locks, but they are challenging to use in building robust and correct software.**
- **Transactional memory is an alternative to fine-grain locks that aims to be easier, while maintaining performance.**

2. Motivating Adaptive Concurrency Control

- **Figure 1: Lee's routing algorithm [2], for some datasets, showed decaying *available parallelism*, i.e. the number of transactions that could be executed concurrently without any aborts.**
- **Executing more, or fewer, transactions than the available parallelism is inefficient: executing too many leads to aborts and wasted work, executing too few degrades performance.**
- **Concurrency control attempts to dynamically change the number of transactions executed concurrently at runtime in response to the available parallelism.**

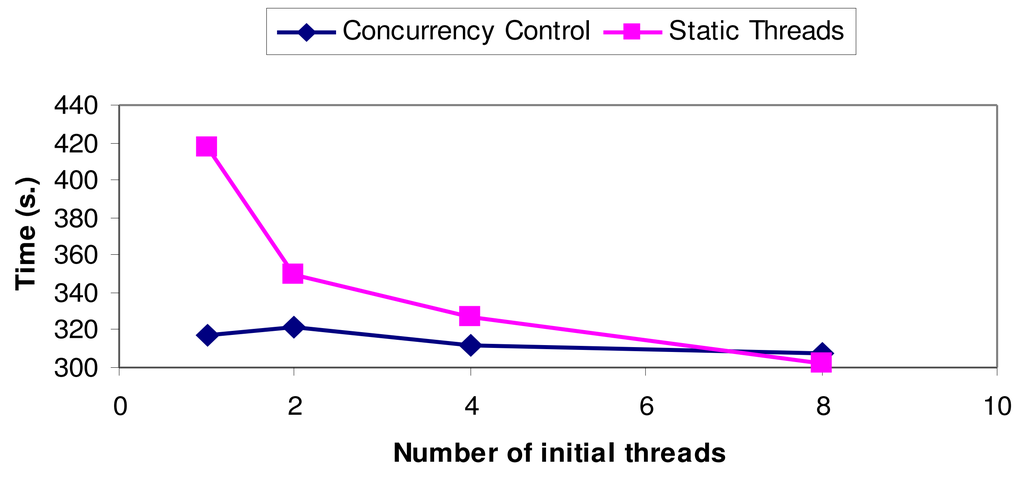
Figure 1. Decaying available parallelism
in a Lee's routing algorithm input dataset



3. Implementation

- **Available parallelism is approximated by calculating Transaction Commit Rate (TCR) as numCommittedTx/numTotalTx*100.**
- **TCR calculated every 20 seconds, then control algorithm invoked:**
- **Control algorithm: If (TCR < 50%) decrease number of threads by one, if (TCR > 80%) increase by one, else leave unchanged.**
- **Implemented and evaluated in DSTM2 [3] using Lee's routing algorithm. Parameters chosen through tuning experiments.**

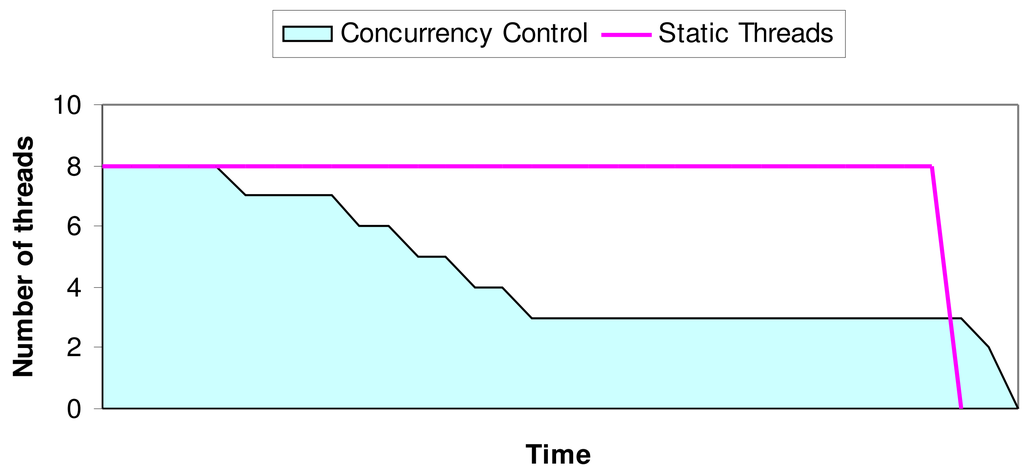
Figure 2. Execution time comparison
in a Lee's routing algorithm input dataset



4. Evaluation

- **Eight core hardware platform used.**
- **Priority [4] contention manager results presented as it gave best overall execution time with static number of threads, thus these results show the *minimum benefit* of concurrency control.**
- **Figure 2: Concurrency controlled execution time improves when a static number of threads under-exploit the available parallelism.**
- **Figure 2: For all numbers of initial threads, execution time variance reduces and is near best-case static number of threads execution time, reducing need to select the number of threads.**
- **Figure 3: At 8 threads, concurrency control reduces resource usage by 41%, yet only increases execution time by 2%.**

Figure 3. Resource utilization comparison
in a Lee's routing algorithm input dataset



5. Summary

- **Adaptive concurrency control automatically improves performance when available parallelism is under-exploited.**
- **Adaptive concurrency control automatically reduces resource usage when available parallelism is over-exploited, with minimal performance degradation.**
- **Concurrency reduces the need to specify a number of threads as it gives performance near to the best-case performance seen with a static number of threads, regardless of the number of initial threads specified.**

References

[1] Ansari *et al.*, Adaptive concurrency control for transactional memory, *MULTIPROG '08*
[2] Watson *et al.*, A study of a transactional parallel routing algorithm, *PACT '07*
[3] Herlihy *et al.*, A flexible framework for implementing software transactional memory, *OOPSLA '06*
[4] Scherer III *et al.*, Contention management in dynamic software transactional memory, *CSJP '04*

Email: ansari@cs.manchester.ac.uk **Further info:** http://www.cs.manchester.ac.uk/apt/people/ansarim/