Experiences using Adaptive Concurrency in Transactional Memory with Lee's Routing Algorithm

Mohammad Ansari
The University of Manchester
ansari@cs.manchester.ac.uk

Christos Kotselidis
The University of Manchester
kotselidis@cs.manchester.ac.uk

Kim Jarvis
The University of Manchester
jarvisk@cs.manchester.ac.uk

Mikel Luján
The University of Manchester
mikel@cs.manchester.ac.uk

Chris Kirkham
The University of Manchester
chris@cs.manchester.ac.uk

Ian Watson
The University of Manchester
watson@cs.manchester.ac.uk

Abstract

Experience in profiling Lee's routing algorithm, a new complex TM application, showed that transactional applications may exhibit dynamic exploitable parallelism, i.e. the amount of useful parallelism available at any point in time varies during the execution of the application. Obviously, executing too many transactions at times when the available parallelism is low will lead to high contention and wasted computation in aborted transactions, and vice versa. Current Transactional Memory (TM) implementations do not account for this behavior.

This work employs *adaptive concurrency* to dynamically adjust the number of threads executing transactions concurrently. Our preliminary evaluation is performed in DSTM2 using Lee's routing algorithm, both of which were simple to modify to enable adaptive concurrency, and shows significant reduction in resource usage, and modest performance gains.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel Programming

General Terms Algorithms, Experimentation, Performance

Keywords Adaptive Concurrency, DSTM2, Transactional Memory

1. Introduction

Chip Multiprocessors (CMPs) have become mainstream, and it is well-known that a challenge for future software is to be highly scalable. The difficulty with achieving high scalability is the lack of easy-to-use concurrent programming paradigms. Locks and barriers have been used for decades to program parallel applications, but are known to be difficult to use to build large, correctly functioning software, and this has typically been a domain for experts. Transactional Memory (TM) is a new concurrent programming paradigm that promises to ease programmer effort significantly.

PPoPP'08, February 20–23, 2008, Salt Lake City, Utah, USA.
ACM 978-1-59593-960-9/08/0002.

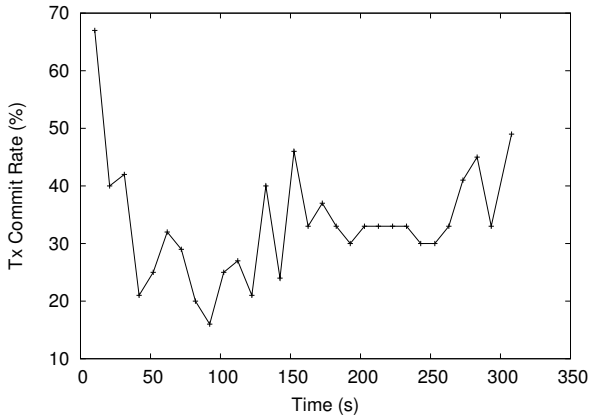

Figure 1. Dynamic amounts of exploitable parallelism in Lee's routing algorithm (8 threads).

TM makes concurrent programming easier by requiring programmers to label code blocks that access shared variables as transactions. A runtime layer monitors execution of code within transactions and automatically resolves conflicts between multiple accesses to shared variables from different threads. The difficulty is moved from the application developer to the TM implementation developer.

Due to the relative infancy of TM, recent research has focused on many areas of TM implementation; work has been done on improving performance, strengthening isolation, TM-specific complex applications, extending TM theory, and more. This work explores a new area by searching the combined space of performance and complex applications.

Our investigations into the execution profile of Lee's routing algorithm [3], a new complex application for TM, showed that it exhibited dynamic exploitable parallelism during execution. By this we mean the number of transactions that can be usefully executed in parallel (without conflicts) varies over time. Figure 1 shows the percentage of transactions committed at sampled points during the execution of Lee's routing algorithm, Attempting to execute more, or fewer, transactions than the available parallelism is inefficient: executing more leads to conflicts and aborts, whilst executing fewer leads to poor exploitation of the available parallelism.

Threads	Non-adaptive (s)	Adaptive (s)	Speedup w/adaptive
1	417.27	317.15	1.32
2	349.28	321.89	1.09
4	327.24	311.33	1.05
8	301.93	307.99	0.98

Table 1. Execution times and speedup of adaptive-enabled execution and non-adaptive execution.

Threads	Resource usage ratio adaptive/non-adaptive
1	1.97
2	1.08
4	0.85
8	0.59

Table 2. Amount of resources used by adaptive-enabled execution over non-adaptive execution.

This paper presents *adaptive concurrency*, the first attempt to harness dynamic amounts of exploitable parallelism in TM applications. Adaptive concurrency controls the number of transactions executed in parallel by dynamically changing the number of threads used based on the sampled transaction commit percentage.

2. Adaptive Concurrency Implementation

This section describes the basic adaptive algorithm, and the changes made to the TM implementation (DSTM2 [2]). The adaptive algorithm is a basic first attempt, and remains subject to improvement, but discussion of adaptive algorithms is beyond the scope of this paper and the topic of TM. DSTM2 collects per-thread statistics of total transactions executed, and the number committed. The adaptive algorithm averages these results over all executing threads to produce an average transaction commit percentage.

If the transaction commit percentage is above an upper threshold, the number of threads executing is increased by one, and vice versa. If the percentage is within the threshold range, no change is made. For the results submitted in this paper the upper threshold is 80%, the lower threshold is 30%, and the adaptive algorithm fires every 20 seconds, i.e. the sampling period is 20 seconds.

Modifications made to DSTM2 converted the static array of threads into a variable thread pool, with flags added to allow the adaptive algorithm to enable/disable threads. Lee's routing algorithm generates all transactions at the start and places them in a shared queue. The small change needed to Lee's routing algorithm was to ensure a thread that has been disabled, and aborts its final transaction, returns the transaction to the shared queue for another active thread to execute and commit.

3. Evaluation with Lee's Routing Algorithm

Evaluation is performed on an 4 x dual-core Opteron-based machine with OpenSUSE 10.1, Sun Java 1.6 64-bit using the standard dataset supplied with Lee's routing algorithm that generates a microcode microprocessor PCB. All contention managers were used [4], but only results from the Priority manager [1] are presented here as it gave the best execution time results.

The benchmark was executed using 1, 2, 4, and 8 threads. The non-adaptive execution uses the same number of threads throughout, and the adaptive-enabled execution starts with the number of threads specified, but changes it according to the transaction commit percentage.

Table 1 shows the speedup of the adaptive-enabled execution over the non-adaptive execution. Adaptive execution is significantly faster than non-adaptive single-threaded execution, but in all other cases the performance difference is marginal. One clear benefit of the adaptive execution, however, is that the difference in execution time between the runs is significantly reduced. This offers a significant advantage to developers using TM in that they no longer need to specify the number of threads with which the application should execute, as the adaptive algorithm automatically changes it to suit the application's commit profile.

For the resource utilization results, we approximated one processor to one thread, and calculate resource usage as number of threads multiplied by the period for which that number of threads executed. Table 2 shows the ratio of adaptive-enabled resource usage compared to non-adaptive. For single-threaded (initial number), adaptive-enabled uses nearly twice as many resources as non-adaptive, and correspondingly gets significant speedup as well, as shown in Table 1. At 8 threads, adaptive-enabled resouce usage drops significantly, yet still the execution time is only slightly slower.

These results show that adaptive-enabled execution correctly utilizes more resources when exploitable parallelism is available, and correctly conserves resources when parallelism is not available. In the former case, performance improves, and in the later case performance remains comparable, which is a positive result for adaptive-enabled execution.

4. Summary

This work has presented the first application of adaptive concurrency based on the percentage of committed transactions over a period of time. It has shown that, for applications that exhibit dynamic amounts of exploitable parallelism such as Lee's routing algorithm, adaptive concurrency can result in either good resource savings, or increased resource usage, but with a corresponding increase in performance.

The algorithm presented is basic, and many improvements are possible [4]. Furthermore, the key parameters (lower threshold, upper threshold, sample period) have not been thoroughly explored. Finally, from a TM perspective, we have not explored other TM statistics to guide the adaptive concurrency algorithm; this may yield even better results.

References

[1] William Scherer III and Michael Scott, Contention Management in Dynamic Software Transactional Memory, *Proceedings of the ACM PODC Workshop on Concurrency and Synchronization in Java Programs*, St. John's, NL, Canada, Jul, 2004

[2] Maurice Herlihy and Victor Luchangco and Mark Moir, A flexible framework for implementing software transactional memory, *OOPSLA '06: Proceedings of the 21st Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, Portland, Oregon, USA, Oct, 2006

[3] Ian Watson and Chris Kirkham and Mikel Luján, A Study of a Transactional Parallel Routing Algorithm, *PACT '07: Proceedings of the 16th International Conference on Parallel Architectures and Compilation Techniques*, Brasov, Romania, Sept, 2007

[4] Mohammad Ansari and Christos Kotselidis and Kimberly Jarvis and Mikel Luján and Chris Kirkham and Ian Watson, Adaptive Concurrency Control for Transactional Memory, *Technical Report CSPP-43, School of Computer Science, University of Manchester*, Sept, 2007