Brief Announcement: Transactional Memory and the Birthday Paradox

Craig Zilles
University of Illinois at Urbana-Champaign
zilles@cs.uiuc.edu

Ravi Rajwar
Intel Corporation
ravi.rajwar@intel.com

ABSTRACT

Many word-based Software Transactional Memory systems (STMs) have been proposed using tagless ownership tables, where read and write permissions are granted at the granularity of all addresses that map to a given ownership table entry. This optimization to reduce overhead potentially results in false conflicts. Using address traces from a multithreaded program, we demonstrate that the frequency of these false conflicts grows superlinearly with both the TM data footprint and concurrency and that increasing the size of the ownership table results in only a sub-linear reduction in conflict rate. These somewhat surprising relationships have a theoretical foundation that is also responsible for the (naively) unintuitive statistical result generally referred to as the "Birthday Paradox." We present an analytical model based on random population of an ownership table by concurrently executing transactions that correctly predicts the trends in measured data. These results call into question the viability of such an optimization that can undermine the scalability and concurrency claims of software transactional memory.

Categories and Subject Descriptors: D.1.3 [**Software**]: Programming Techniques–*Concurrent Programming*, E.4 [**Data**]Data Storage Representations

General Terms: Algorithms, Performance, Theory

Keywords: Transactional Memory, Concurrency, Birthdays

1. TAGLESS OWNERSHIP TABLES

Word-based STMs [4] use a centralized ownership table that permits the detection of conflicts between transactions. To detect conflicts, the ownership table tracks the read and write footprints of each transaction. Even STM implementations that do not visibly track readers would need to assign an ownership table entry for the read location to record version numbers. To enable conflict detection (when at least one concurrent access is a write), the table must record the addresses of the data items accessed and whether the accesses to that address were read-only or if it included writes. In addition, each thread executing transactions maintains a (private) per-thread *log* that tracks the state of the transaction (*e.g.*, *active*, *committed*) and the transaction's footprint including speculative values for writes.

Figure 1 shows a proposed organization for an ownership table common to many STM proposals [1, 2, 3, 5]. It consists of a table of entries, each with two fields. The first field is the type of entry: *Read*, *Write*, or *Free*. The second field holds either an *owner* that identifies the one thread writing the entry (for *Write*s) or a count of the *number of sharers* for the entry (for *Read*s). Program data is mapped to ownership table entries by hashing the (virtual) address.

SPAA'07, June 9–11, 2007, San Diego, California, USA.
ACM 978-1-59593-667-7/07/0006.

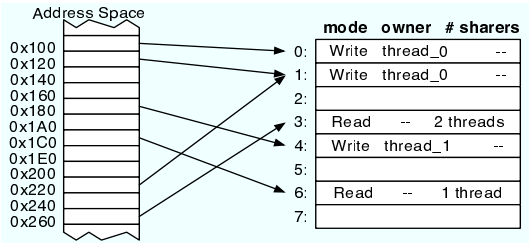

Figure 1: A previously proposed organization for a tagless ownership table. A mapping granularity of 32-byte cache blocks is shown.

Because the ownership table will, in general, be much smaller than a program's data memory footprint, this mapping process results in multiple data items mapping to the same entry of the ownership table. For example, in Figure 1, cache blocks at both address 0x120 and 0x220 map to ownership table entry 1. Because the table is *tagless*—the particular address being accessed is not recorded in the ownership table—entry 1 provides `thread 0` exclusive access to both blocks 0x120 and 0x220. Tagless tables are chosen for their relative simplicity and space efficiency; no tag needs to be stored, nor tag comparison performed, and the hassles of maintaining a linked list for chaining is avoided.

However, this tagless design has the significant drawback of creating false conflicts. Without tags, any two accesses (involving a write) from distinct transactions must be conservatively considered a conflict. Due to the all-or-nothing nature of transactions, a single conflict forces a transaction to either abort or stall until the conflicting transaction commits. Either way, these false conflicts can reduce the achieved concurrency.

1.1 Empirical measurements of conflicts

To explore the relationship between the likelihood of an alias-induced conflict in the ownership table and the characteristics of the transactions being executed. For this study, we perform experiments using synthetic transactions (using a real application), primarily due to the lack of available applications that use transactional memory. We collected address traces from a 4-processor (4-warehouse) execution of the SPECJBB2005 multithreaded benchmark. Using these traces, we populate an ownership table (with N entries) using C concurrent address streams until each stream has written to W cache blocks. As we consume these traces, we remove any true conflicts so we can focus on the aliasing-induced conflicts found in real address streams. For each data point, we run roughly 10,000 trace samples to compute a likelihood of an alias occurring before all traces complete W writes.

From the data shown in Figure 2(a), relationship between conflict rate and transaction size is clearly superlinear and, for configurations with modest conflict rates (*e.g.*, $< 20\%$) the conflict rate is almost proportional to the square of the size of the write foot-

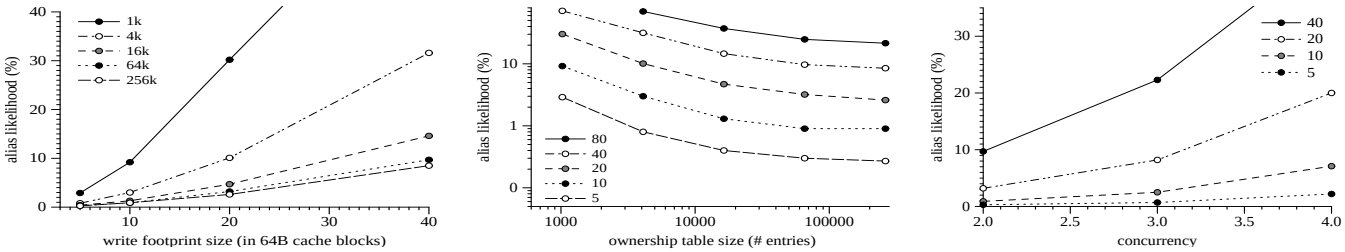

Figure 2: Aliasing likelihood as a function of data footprint F (a), ownership table size T (b), and concurrency C (c). (a,b) data shown for $C = 2$, $F = [5, 10, 20, 40, 80]$ and $T = [1k, 4k, 16k, 64k, 256k]$ entries. (c) data shown for $F = [5, 10, 20, 40]$ and $N = 64k$.

print. The conflict rate as a function of the ownership table size is initially slightly less than inversely proportional (*i.e.*., a 4-fold increase in table size yields a 3-fold reduction in alias likelihood) but the effectiveness of further reductions diminishes as the alias likelihood approaches an asymptote at very large table sizes (Figure 2(b)). Finally, Figure 2(c) plots the conflict rate as a function of concurrency. A strong superlinearity is evident here as well, with a concurrency of 4 having an almost 6-fold larger conflict rate than a concurrency of 2.

2. ANALYSIS

In this section, we demonstrate that these observed trends can be explained through statistical analysis of the likelihood of collisions occurring from acquiring read and write ownership of random entries of a sparsely-populated ownership table. This effect strongly relates to the so-called "Birthday Paradox," which predicts that the likelihood of two people sharing a birthday is greater than 50% in an unintuitively small number of people (23). We compute closed formed solutions for the relationships between the conflict rate and transaction size, ownership table size. This model can be easily extended to arbitrary concurrency [6].

For this analysis, we have simplified transaction execution in the following ways.

1. there are no true conflicts between transactions.
2. the cache blocks accessed by a transaction have addresses that are equally likely to map (using a hash function) to any of the N ownership table entries.
3. ratio of cache block reads-to-writes is a constant, α, such that α new cache block reads proceed every additional cache block written.
4. the ownership table is sufficiently sparsely populated that aliasing within a transaction's footprint is negligible.
5. we are primarily concerned with tables with low conflict rates, permitting the use of a sum of probabilities formulation.

A transaction can only commit if it reaches its end without a conflict. A false conflicts occur if addresses from two transactions (A and B) map to the same entry of the ownership table and at least one of them is a write. Specifically, when a read is performed to a new cache block, its likelihood of causing a conflict is proportional to the fraction of the ownership table containing write entries belonging to the other transaction: a W/N chance. When a write is performed to a new cache block, its likelihood of causing a conflict is proportional to the fraction of the ownership table containing read *or* write entries belonging to the other transaction: a (R+W)/N chance.

Given that $R = \alpha W$, we can compute the likelihood that a conflict occurs every time transaction A reads α new cache blocks and writes one new cache block, in terms of the current footprints of transaction B:

$$\alpha\frac{(W_B-1)}{N} + \frac{(\alpha+1)W_B}{N} = \frac{(1+2\alpha)W_B - \alpha}{N} \quad (1)$$

Since both transactions are executed in lock step ($W_A(t) = W_B(t) = W(t)$), there is the corresponding rate for transaction B. Thus, as we extend both transactions, the likelihood of a conflict is almost twice the likelihood computed in Equation 1; to compensate for double counting the likelihood of the n_{th} write by A conflicting with the n_{th} write by B, we need to subtract out $1/N$. To compute the likelihood of a conflict for transactions of a given size, we can sum the incremental likelihoods for each step, and we expose the observed relationships between conflicts and transaction footprint and table size.

$$\sum_{w=1}^{W} \frac{(2+4\alpha)w - 2\alpha - 1}{N} = \frac{(1+2\alpha)W^2}{N} \quad (2)$$

The practical implications of this relationship are extremely acute for a hybrid TM (*e.g.*, [2]), where the STM will be used for transactions that exceed the size of hardware resources. Empirically, we have found a 4-way set-associative 32KB cache with 64-byte cache blocks to overflow at $W = 71$ and $\alpha = 2$, on average. Solving Equation 2 for N indicates that achieving a commit probability above 50% requires an ownership table with more than 50,000 entries. A 95% commit probability requires a table with over a half million entries. With the modestly-sized tables proposed in the literature, a tagless organization will almost guarantee a maximum concurrency of 1 for overflowed transactions.

3. CONCLUSIONS

From this study, we conclude that tagless ownership tables are not a robust approach to implement transactional memory. In contrast, tagged ownership tables, which record addresses and use chaining to handle aliasing, do not result in false conflicts and, when appropriately sized, only infrequently require chaining.

4. REFERENCES

[1] A.-R. Adl-Tabatabai et al. Compiler and Runtime Support for Efficient Software Transactional Memory. In *PLDI*, 2006.

[2] P. Damron et al. Hybrid Transactional Memory. In *ASPLOS*, 2006.

[3] T. Harris and K. Fraser. Language Support for Lightweight Transactions. In *OOPSLA*, Oct 2003.

[4] J. R. Larus and R. Rajwar. *Transactional Memory*. Morgan and Claypool, Dec. 2006.

[5] N. Shavit, D. Dice, and O. Shalev. Transactional Locking II. In *TRANSACT*, 2006.

[6] C. Zilles and R. Rajwar. Transactional Memory and the Birthday Paradox. Technical Report UIUCDCS-R-2007-2835, University of Illinois at Urbana-Champaign, 2007.