TransMetric: Architecture Independent Workload Characterization for Transactional Memory Benchmarks

James Poe  
ECE Department  
University of Florida  
Gainesville, FL 32611  
jpoe@ufl.edu

Clay Hughes  
ECE Department  
University of Florida  
Gainesville, FL 32611  
cmhug@ufl.edu

Tao Li  
ECE Department  
University of Florida  
Gainesville, FL 32611  
taoli@ece.ufl.edu

ABSTRACT
Transactional memory (TM) has emerged as a parallel programming paradigm for multi-core processors yet there is no standardized set of metrics with which to describe their behavior. In this work, we propose a set of transaction-oriented workload characteristics that can accurately capture the behavior of transactional memory programs. We apply principle component analysis and clustering algorithms to analyze the proposed transactional workload characteristics and show that these characteristics are architecturally independent.

Categories and Subject Descriptors
D.1.3 [Programming Techniques]: Concurrent Programming;  
C.4 [Performance of Systems]: Modeling Techniques,  
Performance Attributes

General Terms
Measurement, Performance.

Keywords
Transactional memory, workload characterization.

1. INTRODUCTION
Transactional memory (TM) is a parallel programming model that uses transactions for synchronization. In this context, a transaction is a block of code that is guaranteed to execute atomically. Using transactions over traditional locks has the potential to reduce some of the complexity of parallel programming. However, because transactional memory has only returned to the forefront of software and computer engineering in recent years there is no common language with which to discuss their execution characteristics.

To measure the behavior of a transactional program, a set of metrics must be defined. In this paper, we provide a set of characteristics to describe transactional behavior. Previous research [1, 2, 3] has shown that taken individually these characteristics can describe an overall feature of a program. We combine the metrics and monitor them over the lifetime of the program, giving much more insight than a simple aggregate value of a single trait can provide. However, this is more data than can be accurately analyzed by hand and some of these characteristics may be correlated with one another, hindering any meaningful observations about which characteristics may be important for a given program. To solve these problems, principal component analysis is used to reduce the size of the characteristics vector. Cluster analysis is then used on the reduced vectors to evaluate the similarity of the benchmarks. The clustering algorithm provides an easy-to-understand representation of the complete data set. Using their linkage distances, the clusters can be analyzed to determine the overall similarity, with short distances indicating strong clustering and larger distances indicating weak clustering.

2. TRANSACTIONAL WORKLOAD CHARACTERISTICS
In general, transactional workloads can be characterized by a small set of features. Table 1 gives an overview of these traits, that when combined, provide an excellent means of quantifying the behavior of transactional workloads. Moreover, these attributes are largely architecture independent; they provide a single set of metrics to describe the characteristics of a workload set regardless of the underlying transactional memory system. For example, conflict behavior is the result of the intrinsic characteristics of a workload when run on a specific architecture design. While the same workload will exhibit different conflict behavior when run on two different transactional models (Eager/Eager or Lazy/Lazy), two different workloads will experience similar conflict behavior if they have similar characteristics when run on the same architecture.

<table>
<thead>
<tr>
<th>Program Characteristics</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Percentage</td>
<td>Fraction of instructions executed by committed transactions.</td>
</tr>
<tr>
<td>Transaction Size</td>
<td>Total number of instructions executed by committed transactions.</td>
</tr>
<tr>
<td>Read Set Size Ratio</td>
<td>Total number of reads within a transaction divided by the number of unique addresses read.</td>
</tr>
<tr>
<td>Write Set Size Ratio</td>
<td>Total number of writes within a transaction divided by the number of unique addresses written.</td>
</tr>
<tr>
<td>Read Set Conflict Density</td>
<td>The total number of potential conflict addresses read by a transaction divided by that transactions total read set.</td>
</tr>
<tr>
<td>Write Set Conflict Density</td>
<td>The total number of potential conflict addresses written by a transaction divided by that transactions total write set.</td>
</tr>
<tr>
<td>Read Set Size</td>
<td>Total number of unique memory addresses read by committed transactions.</td>
</tr>
<tr>
<td>Write Set Size</td>
<td>Total number of unique memory addresses written by committed transactions.</td>
</tr>
<tr>
<td>Write Read Ratio</td>
<td>The total number of writes within a transaction divided by the number of reads within the transaction.</td>
</tr>
</tbody>
</table>
underlying model. This allows the researcher to use the characteristics described here to identify common traits among a set of workloads and use those to compare different architecture implementations. The characteristics proposed are transaction percentage, transaction size, read-/write-set ratios, read-/write-set conflict densities, read-/write-set sizes, and the write-read ratio of each transaction.

Transaction percentage is the total number of retired committed transactional instructions divided by the total number of instructions retired, providing insight into how large of a role the transactional portions of a workload play in the overall execution of an entire benchmark.

Transaction size is defined as the total number of instructions committed by a transaction. The granularity of a transaction is directly related to the period that a transaction maintains ownership over its read/write set as well as the amount of work lost on an abort.

The read and write set ratios are defined as the total number of reads divided by the number of unique reads and the total number of writes divided by the number of unique writes, respectively. This metric provides insight into the spatial locality of the memory patterns of each individual transaction.

While the read/write set ratios help to describe overall spatial locality of memory operations, they are insufficient to fully characterize a transaction. For the overall characterization of contentious memory patterns, read and write conflict densities are included. The read/write conflict densities are defined as the total number of potentially contentious addresses within a transaction’s read set divided by the total read set and the total number of potentially contentious addresses within a transaction’s write set divided by the write set of the transaction, respectively. These metrics capture the worst-case contention rate of the read/write sets for all possible thread alignments without the need to run exhaustive numbers of simulations. This characteristic categorizes the contentiousness of a specific transaction not based on the aggregate size of a memory set but on the actual contentiousness of the memory locations within those sets.

The read and write set sizes quantify the number of unique memory addresses from which a program reads (read set size) as well as the number of unique memory addresses to which a program writes (write set size). The size of the read and write sets are important because they affect the data footprint of each transaction as well as the period commits and aborts take.

Finally, the write-read ratio describes the relative frequency of writes to reads within a transaction. This metric expands upon the previous memory related metrics by relating the number of writes to the number of reads. While multiple transactions are permitted to read an address, only a single write is allowed. Thus even if two transactions share a similar read and write conflict ratios, if one transaction is heavily weighted with the more contentious writes and the other is more heavily weighted with less contentious reads, this can result in different workload execution.

3. PRELIMINARY RESULTS

Using the proposed metrics, we characterize the similarity of existing TM workloads. All benchmarks are run to completion with 8-threads using SuperTrans [4]. SuperTrans is built on SESC and is a cycle accurate, multiple-issue, out of order common chip multiprocessor (CMP) simulator that supports cycle accurate simulation of eager and lazy conflict detection and eager and lazy version management. All program characteristics are recorded over time and stored as a vector. Principle component analysis is used to remove correlated values and STATISTICA is used to hierarchically cluster the principal components.

So that architects do not need to redefine their program characteristics for each transactional model, it is necessary to show that the metrics in Table 1 are implementation-independent. Figure 1 shows the dendrogram produced from the characteristics in Table 1 using a Lazy/Lazy, Eager/Lazy, and Eager/Eager model for each program. The resulting clusters are formed using the Euclidean distance between the benchmarks and are generated hierarchically using the single linkage distance. The y-axis represents the linkage distance, which is used to derive the similarity between the benchmarks. The smaller the linkage distance, the more tightly the programs are clustered and the larger the linkage distance, the more loosely the programs are clustered. From Figure 1 it can be seen that each benchmark is tightly clustered with itself (with average distances less than 1), no matter what the underlying conflict detection and version management policy. This shows that regardless of the transactional model, the characteristics in Table 1 can provide a good indication of the expected performance.

4. CONCLUSION

This paper provides an architecturally independent set of transactional memory program attributes. With the increased interest in transactional memory workloads, we believe that the proposed characteristics will help the TM community with design evaluations and provide a common platform for discussing TM programs.

5. REFERENCES