

On the (Dis)similarity of Transactional Memory Workloads

Clay Hughes[§], James Poe[§], Amer Qouneh, and Tao Li
Intelligent Design of Efficient Architecture Lab (IDEAL)
University of Florida, Gainesville, United States of America
{cmhug, jpoe, aqouneh}@ufl.edu, taoli@ece.ufl.edu

Abstract— Programming to exploit the resources in a multicore system remains a major obstacle for both computer and software engineers. Transactional memory offers an attractive alternative to traditional concurrent programming but implementations emerged before the programming model, leaving a gap in the design process. In previous research, transactional microbenchmarks have been used to evaluate designs or lock-based multithreaded workloads have been manually converted into their transactional equivalents; others have even created dedicated transactional benchmarks. Yet, throughout all of the investigations, transactional memory researchers have not settled on a way to describe the runtime characteristics that these programs exhibit; nor has there been any attempt to unify the way transactional memory implementations are evaluated. In addition, the similarity (or redundancy) of these workloads is largely unknown. Evaluating transactional memory designs using workloads that exhibit similar characteristics will unnecessarily increase the number of simulations without contributing new insight. On the other hand, arbitrarily choosing a subset of transactional memory workloads for evaluation can miss important features and lead to biased or incorrect conclusions.

In this work, we propose a set of architecture-independent transaction-oriented workload characteristics that can accurately capture the behavior of transactional code. We apply principle component analysis and clustering algorithms to analyze the proposed workload characteristics collected from a set of SPLASH-2, STAMP, and PARSEC transactional memory programs. Our results show that using transactional characteristics to cluster the chosen benchmarks can reduce the number of required simulations by almost half. We also show that the methods presented in this paper can be used to identify specific feature subsets. With the increasing number of TM workloads in the future, we believe that the proposed transactional memory workload characterization techniques will help TM architects select a small, diverse, set of TM workloads for their design evaluation.

Keywords-Transactional memory, performance analysis

I. INTRODUCTION

Transactional memory (TM) [3] simplifies parallel programming by providing atomic execution for an arbitrarily large block of code. However, because transactional memory has only returned to the forefront of software and computer engineering in recent years, there is a dearth of transactional programs. Complementing this is the lack of a modern cohesive parallel benchmark suite, forcing researchers to transactionalize older parallel programs while speculating about what future transactional and even parallel programs may look like.

The concern from an architecture standpoint is that blindly choosing benchmarks may not accurately provide a picture of the

entire design space – especially if their characteristics are unknown. Even if a program’s multi-threaded behavior is well known, as in the case of the SPLASH-2 suite, little attention has been paid to their similarities outside of the traditional lock/barrier model. This is a pitfall for computer architects. If too few benchmarks are chosen, the applications may not provide the stressors needed to evaluate a design. If too many benchmarks are chosen, their behavior may overlap and increase design time without providing additional useful information. In the past, several studies have investigated the similarity of sequential workloads and the results show that there is ample redundancy in the SPEC suite [1, 2, 15]. In this paper, we follow a method similar to previous researchers but use a set of transactional metrics to characterize the behavior of transactional programs. To our knowledge, there have been no studies on how to quantify the similarity of TM workloads.

To measure the behavior of a transactional program, we provide a new set of characteristics. Previous research [5, 16, 22] has shown that taken individually these characteristics can describe a single feature of a transactional program. We combine the metrics and monitor them over the lifetime of the program, which provides us with much more insight than the simple aggregate performance of a single trait. However, this is more data than can be accurately analyzed and some of these characteristics may be correlated with one another, further hindering any meaningful observations about which characteristics may be important. To solve these problems, principal component analysis is used to reduce the size of the input vector. Cluster analysis is then used on the reduced vectors to evaluate the similarity of the chosen benchmarks, providing an easy-to-understand representation of the complete data set. The clusters can then be analyzed according to their linkage distance, with short distances indicating strong clustering and larger distances indicating weak clustering, to determine their overall similarity.

The contributions of this work are:

- We propose a set of transactional characteristics that can be used to quantify the similarity of transactional workloads and show that the chosen set of transactional features is independent of the transactional model.
- We identify the (dis)similarities between 15 benchmarks taken from the SPLASH-2, STAMP, and PARSEC suites across the three most common dimensions of transactional memory.
- We show that by discarding similar programs and selecting a subset of the programs, we are able to reproduce the overall behavior of the entire set. Moreover, we are able to capture this behavior even when the transactional and microarchitecture models are varied.
- We show that using the proposed transactional characteristics, fine-grained subsets can be generated to explore specific architectural areas of interest to the transactional developer.

This paper is organized as follows. Section II discusses the transactional characteristics used in this study. Section III provides an overview of principal component analysis and clustering while Section IV discusses the experimental design. Section V shows that the chosen characteristics are independent of the transactional model and provides insight into the similarities of the different programs. In Section VI, we discuss the related work. The paper concludes in Section VII.

II. TRANSACTIONAL WORKLOAD CHARACTERISTICS

In general, transactional workloads can be characterized by a small set of features; an overview of these features is provided in Table I. From these traits, a characteristic vector is built and used as input to the principle component analysis (PCA) algorithm to classify the different transactional workloads. The goal in choosing these metrics was to provide attributes that were able to describe the unique characteristics of individual transactions. To this end, we wanted to define the different aspects of a transaction that play a dominant role in determining the runtime characteristics, contention, and interaction across transactional workloads. Moreover, we wanted these attributes to be architecture independent; this provides a single set of metrics for describing a workload 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* implementation – while the same workload will exhibit different conflict behavior on different transactional models (eager/eager or lazy/lazy), two different workloads may experience the same conflict behavior if they have similar characteristics when run on the same transactional model. From this point of reference, the conflict behavior is an *output* used to validate an *input*, allowing researchers to use our classifications to identify common traits among a set of workloads and use those to compare different implementations.

For this evaluation, we record the *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. Because many transactional workloads contain heterogeneous transactions and different synchronization patterns throughout runtime execution, characteristics are recorded as

TABLE I. TRANSACTION-ORIENTED WORKLOAD CHARACTERISTICS

Program Characteristics		Synopsis
1	Transaction Percentage	Fraction of instructions executed by committed transactions.
2-11	Transaction Size	Total number of committed instructions executed by each transaction.
12-21	Read Set Size Ratio	Total number of unique reads within a transaction divided by the number of addresses read.
22-31	Write Set Size Ratio	Total number of unique writes within a transaction divided by the number of addresses written.
32-41	Read Set Conflict Density	The total number of potential conflict addresses read by a transaction divided by that transaction's total read set.
42-51	Write Set Conflict Density	The total number of potential conflict addresses written by a transaction divided by that transaction's total write set.
52-61	Read Set Size	Total number of unique memory addresses read by committed transactions.
62-71	Write Set Size	Total number of unique memory addresses written by committed transactions.
72-81	Write Read Ratio	The total number of writes within a transaction divided by the number of reads within the transaction.

histograms.

The *transaction percentage* is the total number of committed transactional instructions divided by the total number of instructions retired. This ratio indicates how large of a role the transactional portions of a workload play in the overall execution of a benchmark. This metric is the only metric that is recorded as a scalar value. However, it is important because it helps to quantify the effect that the remaining characteristics have in the overall execution of a benchmark. For example, a workload comprised of transactions that are highly contentious but are only in execution for brief intervals may exhibit less actual contention than a workload comprised of fewer contentious transactions that occur with greater frequency. It is also important to note that we only consider committed and not aborted transactions within the transaction percentage. This is because while the amount of work completed or committed is determined by the workload and its inputs, aborted transactions are a function of the underlying architecture and can have vast variations depending on architectural decisions.

Transaction size is defined as the total number of instructions committed by a transaction. This characteristic is comprised of a histogram describing the individual sizes of transactions across the entire execution time of a workload. 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 unique reads divided by the number of reads and the total number of unique writes divided by the number of writes, respectively. These metrics provide insight into the spatial locality of the memory patterns of each individual transaction. While the *read-/write-set ratios* help describe overall spatial locality of the memory operations, they are insufficient to characterize a transaction. To assist in the overall characterization of the contentious memory access patterns we include the *read conflict density* and the *write conflict density*. The *read conflict density* is defined as the total number of potential contentious addresses within a transaction's read set divided by the total read set size of the transaction. The *write conflict density* is defined as the total number of potential contentious addresses within a transaction's write set divided by the total write set of the transaction. To calculate the addresses that could potentially result in contention within a transaction we first run the entire workload and calculate the read-/write-sets for each transaction. Next, each memory address within a read set is marked as potentially contentious if any other transaction that was not located within the same thread wrote to that address. For addresses belonging to the write-set, each memory address is marked as potentially contentious if any other transaction that was not located within the same thread either read or wrote to that address. Using this method, we can capture the worst-case contention rate of the read-/write-set for all possible thread alignments without the need to run exhaustive numbers of simulations, categorizing the contentiousness of a specific transaction not simply based on the aggregate size of a memory set but on the actual contentiousness of the memory locations within those sets.

We also include the *read-set size* and *write-set size* metrics, which 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 total 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 builds upon and expands 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 have 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.

When combined, the different transactional aspects described in Table I provide an excellent means of quantifying the behavior of transactional workloads. However, due to the extensiveness of the data, a means of processing the information is necessary.

III. DATA ANALYSIS

This section describes our data processing techniques: principal component and cluster analysis.

A. Principal Component Analysis

Principal component analysis (PCA) [24] is a multivariate analysis method that exposes patterns in a high-dimensional data set. PCA reduces the dimensionality of data by linearly transforming a set of correlated variables into a smaller set of uncorrelated variables called principal components. These principal components account for most of the information (variance) in the original data set and provide a different presentation of the data, making the interpretation of large data sets easier.

Principal components are linear combinations of the original variables. For a dataset with p correlated variables (X_1, X_2, \dots, X_p), a principal component Y_1 is represented as $Y_1 = a_{11}X_1 + a_{12}X_2 + \dots + a_{1p}X_p$, where (Y_1, Y_2, \dots, Y_p) are the new uncorrelated variables (principal components) and ($a_{11}, a_{12}, \dots, a_{1p}$) are weights that maximize the variation of the linear combination. A property of the transformation is that principal components are ordered according to their variance, λ , where $\lambda_1 > \lambda_2 > \dots > \lambda_p$. Thus, principal component Y_1 has more variance (information) than Y_2 and Y_2 has more variance than Y_3 , and Y_p has the least variance. If k principal components are retained, where $k \ll p$, then Y_1, Y_2, \dots, Y_k contain most of the information in the original variables. By retaining the first k principal components and ignoring the rest, one can achieve a reduction in the dimensionality of the dataset.

The sum of the variances of the principal components equals the sum of the variances of the original variables. The number of selected principal components control the amount of information retained. The amount of information retained is proportional to the ratio of the variances of the retained principal components to the variances of the original variables. The Kaiser Criterion suggests choosing only the PCs greater than or equal to one. In general, principal components are retained so they account for greater than 85% of the variance. These principal components are the exposed patterns and trends that exist in the dataset.

B. Cluster Analysis

Cluster analysis [6] is a statistical inference tool that allows researchers to group data based on some measure of perceived similarity. There are two branches of cluster analysis: hierarchical and partitional clustering. Our study uses hierarchical, which is a

bottom-up approach that begins with a matrix containing the distances between the cases and progressively adds elements to the cluster hierarchy – building a tree based on the similarity distance of the cases. Unlike partitional clustering where the researcher must pre-define the number of clusters, hierarchical clustering allows researchers to choose the number of clusters themselves based on the linkage distance.

In hierarchical clustering, each variable begins in a cluster by itself then the closest pair of clusters is matched and merged and the linkage distance (discussed later) between the old cluster and the new cluster is measured. This previous step is repeated until all of the variables are grouped into a single cluster. The resulting figure is a dendrogram (tree) with one axis showing the linkage distance between the variables. The linkage distance can be calculated in several ways: single linkage (SLINK) defines the similarity between two clusters as the most similar pair of objects in each cluster, complete linkage (CLINK) defines similarity as the similarity of the least similar pair of objects in each cluster, and average linkage (UPGMA) defines the similarity as the mean distance between the clusters.

IV. EXPERIMENTAL METHODOLOGY

In this section, we present a brief description of the hardware transactional model (HTM) simulation framework, TM workloads, evaluation metrics, and experimental setup.

A. Framework

The transactional memory framework used in this paper, SuperTrans [25], is built on top of SESC [7]. SuperTrans can model a variety of TM design points (e.g. Eager/Lazy conflict detection and Eager/Lazy version management) and report a wide range of TM execution statistics. While the majority of existing HTM studies have used in-order and single-issue processor models, this simulator models detailed multiple-issue out-of-order execution CMPs with parameterized core microarchitecture and TM models. At the center of the model is the coherence module, which coordinates the functional and cycle accurate aspects of the simulator. Correctness is provided by MINT, the MIPS emulator within SESC. The detailed pipeline provides the cycle accurate portion of the model and back-propagates timing information from the coherence module to the functional model.

B. Transactional Model/Microarchitecture

Table II describes the baseline 8-core CMP transactional model and microarchitecture configuration that was used for comparison throughout the study. The *trans_model* parameter describes the conflict detection and version management schemes that were employed. Conflict detection describes the method used to detect conflicts and can occur either immediately when a memory request is made (Eager), or as a batch operation when a transaction is committed (Lazy). Version management refers to whether new values are written directly to memory while old values are logged (Eager) or whether new values are buffered and then written in batch at the time of commit (Lazy). Throughout our analysis, we evaluate all three forms of commonly accepted transactional models: Eager/Eager, Eager/Lazy, and Lazy/Lazy. Conflict detection granularity refers to the level of granularity at which conflicts are detected (usually the size of a cache line), and backoff policy is the backoff method that is used after a transaction is aborted. Because TransPlant is implementation independent, in addition to the actual time taken to write memory values back, values can be assigned to transactional model characteristics and

TABLE II. BASELINE CONFIGURATION

Parameter		Value
TM Model	Trans_model	Conflict Detection / Version Management
	Back_off	Backoff Policy
	Conf_det_g	Conflict Detection Granularity
	PBL	Primary Baseline Latency
	PVL	Primary Variable Latency
	SBL	Secondary Baseline Latency
Processor Core	issue_width	Processor Issue Width
	ROB_size	Reorder Buffer Size
	LSQ_size	Load/store queue size
	int_regs	Integer Registers
	fp_regs	Floating Point Registers
	int_issue_win	Integer Issue Win Size
	fp_issue_win	Floating Point Issue Win Size
	L2_size	L2 cache size
	L2_lat	L2 cache latency
	il1_size	L1 instruction cache size
	il1_lat	L1 instruction cache latency
	d11_size	L1 data cache size
	d11_lat	L1 data cache latency

varied independently from the microarchitecture configuration. Primary/secondary baseline latency and primary variable latency quantify the latencies associated with a commit or an abort. In a lazy version management system, the primary latency is associated with a commit (since new values must be written back), while in an eager version management system the primary latency is associated with an abort (since the old values must be written back). The baseline latency is the static overhead required (e.g. arbitrating for the bus, maintenance, cleanup, etc) and the variable latency is the additional latency required based upon the write-set size. Finally, the microarchitecture core parameters describe a set of processor core features that previous studies have shown to have a large impact on performance. This configuration was chosen because it is representative of an average machine. In the evaluation section, we vary the transactional and microarchitecture parameters independently to evaluate our results across conservative as well as aggressive machines.

C. TM Workloads

For our evaluation, we use a set of 15 benchmarks from three different benchmarking suites commonly used in the analysis of multithreaded workloads: SPLASH-2 [8], STAMP [9, 22], and PARSEC [10, 23]. First, we transactionalized a set of SPLASH-2 benchmarks by converting the locked regions directly into their equivalent transactional counterparts. Since we are primarily interested in transactional activity, we only included those benchmarks that provide a sufficient amount of lock-based activity. While SPLASH-2 provides a good comparison of design points for fine-grained transactions and highly optimized lock-behavior, it is believed that transactional workloads of the future will also be comprised of coarse granularity transactions that may not be well tuned. To capture this trend, we also used the workloads from the STAMP suite (ver. 0.9.6) of transactional benchmarks in our evaluation. For the STAMP suite, we modified the transactions by adopting the transactional annotation used on our HTM model. Additionally, we modified several portions of the thread spawning routines to use lower level Pthread API to match the subset of Pthread functions supported on the base SESC simulator. All other code remains the same. Since the STAMP suite does not provide lock-based equivalents of the transactional

TABLE III. TM WORKLOAD TRANSACTIONAL CHARACTERISTICS (8 PES)

Benchmarks	Trans Model	Commit	Abort	NACK Cycles (M)	Avg. R-Set Size*	Avg. W-Set Size*	Avg. Commit (Inst.)
Barnes 16K particles	EE	6.9E+04	1.6E+03	2.3E+00	6.7	6.5	2.0E+02
	EL	6.9E+04	5.1E+02	6.7E-01	6.7	6.5	2.0E+02
	LL	6.9E+04	3.6E+02	6.8E+00	6.7	6.5	2.0E+02
Fmm 16K particles	EE	4.5E+04	3.0E+00	1.8E-03	13.4	7.3	1.7E+02
	EL	4.5E+04	0.0E+00	4.7E-03	13.4	7.3	1.7E+02
	LL	4.5E+04	2.6E+01	5.2E-01	13.4	7.3	1.7E+02
Cholesky tk15.0	EE	1.6E+04	1.9E+01	1.6E-02	3.1	1.9	2.7E+01
	EL	1.6E+04	6.0E+00	1.2E-02	3.1	1.9	2.7E+01
	LL	1.6E+04	7.8E+01	5.8E-02	3.1	1.9	2.7E+01
Ocean-con 258x258	EE	1.7E+03	5.0E+02	9.2E-02	3.0	0.3	1.0E+01
	EL	1.7E+03	1.9E+02	3.1E-02	3.0	0.3	1.0E+01
	LL	1.7E+03	1.4E+02	2.2E-02	3.0	0.3	1.0E+01
Ocean-non 66x66	EE	2.0E+03	5.2E+03	7.8E-01	3.0	0.4	1.3E+01
	EL	2.0E+03	9.7E+02	9.3E-02	3.0	0.4	1.3E+01
	LL	2.0E+03	7.8E+02	5.7E-02	3.0	0.4	1.3E+01
Raytrace Teapot	EE	7.7E+04	6.4E+04	2.2E+01	6.4	2.5	6.1E+01
	EL	7.7E+04	5.3E+04	2.2E+01	7.2	2.5	6.9E+01
	LL	7.7E+04	2.3E+05	2.6E-01	7.5	2.5	7.4E+01
Water-nsq 512 molecules	EE	1.0E+04	2.2E+01	2.7E-03	10.9	2.9	5.9E+01
	EL	1.0E+04	1.0E+00	6.0E-03	10.9	2.9	5.9E+01
	LL	1.0E+04	1.1E+02	6.5E-01	10.9	2.9	5.9E+01
Water-sp 512 molecules	EE	1.5E+02	0.0E+00	1.5E-04	2.5	1.4	1.3E+02
	EL	1.5E+02	0.0E+00	4.7E-03	2.5	1.4	1.3E+02
	LL	1.5E+02	7.3E+01	4.0E-03	2.6	1.5	3.7E+02
Bayes 384 records	EE	1.7E+02	9.7E+01	3.0E+00	793.2	713.7	4.5E+05
	EL	1.7E+02	2.0E+01	3.0E+00	793.2	713.7	4.5E+05
	LL	1.4E+02	2.8E+01	7.8E-03	437.7	371.5	2.3E+05
Genome g256s15 n16k	EE	6.6E+03	3.3E+02	4.9E-01	28.0	5.1	1.2E+03
	EL	6.6E+03	1.1E+02	5.4E-01	27.9	5.1	1.2E+03
	LL	6.6E+03	2.8E+02	1.8E+00	27.8	5.1	1.2E+03
Kmeans Random 1000_12	EE	6.7E+03	0.0E+00	5.0E-03	6.9	2.5	1.0E+02
	EL	8.1E+03	3.0E+00	1.9E-02	6.9	2.5	1.0E+02
	LL	6.7E+05	4.7E+04	6.4E+00	6.9	2.5	1.0E+02
Labyrinth 512 molecules	EL	1.1E+02	2.7E+03	1.5E+04	601.7	403.1	5.2E+05
	LL	1.1E+02	3.4E+02	4.7E-03	606.8	408.1	5.2E+05
Vacation 4096 tasks	EE	4.1E+03	2.9E+03	7.4E+00	67.1	13.2	1.6E+03
	EL	4.1E+03	1.5E+03	6.0E+00	66.5	13.2	1.6E+03
	LL	4.1E+03	4.0E+02	4.8E-02	66.2	13.2	1.6E+03
Fluidanimate Simsmall	EE	1.8E+06	4.6E+01	2.6E-02	1.6	1.1	7.0E+00
	EL	1.8E+06	3.4E+01	5.6E-02	1.6	1.1	7.0E+00
	LL	1.8E+06	1.8E+04	4.4E+01	1.6	1.1	6.9E+00
Streamcluster Simsmall	EE	2.0E+04	0.0E+00	6.8E-05	2.9	1.9	2.1E+01
	EL	2.0E+04	0.0E+00	2.2E-03	2.9	1.9	2.1E+01

*Set size calculations based on 32B granularity

benchmarks, we created equivalent lock based versions using the same level of granularity for use in the transaction-to-lock speedup metric.

We also converted several of the benchmarks from the recently announced PARSEC benchmark suite to our simulation framework. We focused on the benchmarks that contain the highest level of lock-based activity. Of these, we were unable to include the benchmarks that require architecture-specific assembly code for which MIPS is not yet supported. We have included fluidanimate, the benchmark that contains the highest level of lock based activity, as well as streamcluster. Since a primary synchronization primitive used in streamcluster is conditional variables, we also implemented transactional equivalents of the conditional primitives within SESC that make use of transactions as their underlying locking mechanism.

Table III summarizes the transactional behavior of each of the workloads on an 8-core CMP described in Table II. Included in the table are results for Eager/Eager, Eager/Lazy, and Lazy/Lazy models of conflict detection and version management respectively. As can be seen, many of the values (e.g. read/write set size) are inherent to the algorithm and do not vary based on the choice of transactional dimensions while other values, such as the number of aborts, are largely determined by policy decisions. This table also clearly demonstrates the need for more workload diversity than is achievable through SPLASH-2 alone. While the number of transactions across SPLASH-2 differs, the size of the transactions, read sets, and write sets, is nearly uniform. In contrast, STAMP provides a greater level of workload diversity. We include the two PARSEC benchmarks to provide a ‘first look’ at their performance, which seems to indicate that they will fall somewhere in between SPLASH-2 and STAMP.

D. Transactional Metrics

To examine the accuracy of similarity classification we use the following transactional metrics: speedup, speedup-total cycles, abort-trans ratio, NACK-trans ratio, and percent committed instructions (see Table IV). Our goal in choosing these metrics was to capture the global effects of the transactional workload as well as the effects that are more specific to the transactional model. To this end, we have included both forms of speedup metrics as well as contention metrics specific to transactional memory.

Speedup is defined as the amount of time in cycles a lock-based version of the program takes to complete divided by the amount of time the transactional version takes to complete. We use this definition of speedup as opposed to a comparison with a sequential version of the benchmark because it allows us to accentuate the differences caused by the transactional portions of the workload. For example, in many of the SPLASH-2 benchmarks where the synchronization has been heavily tuned, the overall percentage of transactional code is relatively small compared to the entire execution time of the benchmark. Because of this, varying parameters that affect the transactional characteristics would result in very small differences in the overall execution time of the program and provide us with a false sense of similarity. Instead, by comparing the transactional version to an equivalent lock version, we are able to magnify the differences introduced by the transactions. In addition to our normal definition of speedup, we also use a speedup total cycles metric that compares the total number of cycles consumed across all threads on the lock version to the total number of cycles consumed across all threads on the transactional version. This allows us to elicit more of the transactional behavior by accentuating the differences in transactional behavior across all threads as opposed to simply the longest running thread.

The three remaining metrics all focus on transaction-specific behavior. The aborted cycles to transactional cycles ratio is defined as the total number of aborted cycles divided by the total number of transactionally executed cycles. This metric provides insight into the percentage of work that was “wasted” due to aborts. The NACK cycles to transactional cycles ratio is defined as the total number of cycles spent stalled due to NACK conditions divided by the total number of executed transactional cycles. While both metrics give an indication of the cycles lost due to contention, it is important to analyze both metrics because this provides for a more comprehensive analysis across transactional models. For example, a simple analysis of aborts may not be as interesting on an Eager/Eager system that uses NACK stalls to

TABLE IV. TRANSACTIONAL METRICS

Name	Equation
Speedup (Su)	Lock Max Cycle / Transaction Max Cycle
Su Total Cycles	Lock Total Cycles / Transaction Total Cycles
Aborted - Trans Ratio	Total Aborted Cycles / Total Transactional Cycles
NACKed - Trans Ratio	Total NACKed Cycles / Total Transactional Cycles
Commit / Abort - Instruction Ratio	Committed Instructions / (Committed + Aborted Instructions)

avoid aborts. A similar argument can be made for NACK stall cycles on a Lazy/Lazy system. Finally, we include commit/abort instruction ratio, which is defined as the total number of committed instructions divided by the total number of transactional instructions. This metric is similar to the previous contention based metrics except that it goes further to describe the percentage of transactional work that was useful using instruction counts instead of cycles, allowing for a less architecturally dependant analysis.

E. Experimental Design

The transactional workload characterization presented consists of the following steps:

- 1) For each of the three transactional models (EE/EL/LL), the program characteristics in Table I are gathered using SuperTrans. This produces a vector of 81 components for each input program; a total of 81 variables and 15 cases for each transactional model (an 81x15 matrix when considered independently and an 81x45 matrix when combined).
- 2) Instead of using the variables directly, PCA is used to remove variables that are correlated. The p variables are normalized to a unit normal distribution and then PCA is performed using STATISTICA [11], producing p principal components.
- 3) These p principal components are reduced using some heuristic – in this case choosing a minimum amount variance that needs to be retained by the principal components. The chosen components can then be used for further analysis.
- 4) STATISTICA is used to hierarchically cluster the principal components chosen in (3); a similarity study is performed using these clusters and the configuration from Table II.
- 5) From the dendrogram produced in (4), the program set can be divided into smaller subsets. The representative performance of the subsets is determined using a sensitivity study where the transactional and microarchitecture models are varied independently.

TABLE V. TRANSACTIONAL MEMORY PARAMETERS

	EE/EL/LL 0	EE/EL/LL 1	EE/EL/LL 2	EE/EL/LL 3	EE/EL/LL 4
back_off	Random Linear	Random Linear	Random Linear	Random Linear	Random Linear
conf_det_g	32	32	32	32	32
PBL	10	30	70	90	110
PVL	4	6	10	12	14
SVL	5	7	11	13	15

F. TM and Microarchitecture Parameters

In addition to the baseline configuration (Table II) used to generate the program clusters, five transactional models, shown in Table V, and four microarchitecture models, shown in Table VI, are used to verify the accuracy of benchmark subsetting. The results are normalized against the baseline case and the average is used to compare each subset against the performance of the whole benchmark set across the five transactional dimensions from section 4.4. To cover a wide range of designs, the dimensions in

TABLE VI. MICROARCHITECTURE PARAMETERS

	A0	A1	A2	A3
issue width (IW)	2	4	4	8
ROB Size	9*IW+32	14*IW+32	36*IW+32	36*IW+32
LSQ Size	5*IW+32	7*IW+32	20*IW+32	20*IW+32
int regs	4*IW+32	6*IW+32	16*IW+32	16*IW+32
fp regs	3*IW+32	4*IW+32	12*IW+32	12*IW+32
int issue win	3*IW+32	4*IW+32	12*IW+32	12*IW+32
fp issue win	2*IW	3* IW	6* IW	6* IW
L2 size	1024	2048	8192	8192
L2 lat	8	10	20	20
III size	8	16	64	128
DII size	8	16	64	128
DII lat	1	2	3	3

each parameter set were chosen to range from conservative (EE0/EL0/LL0 and A0) to unconventional (EE4/EL4/LL4 and A3).

V. RESULTS

In this section, we show that the characteristics in Table I are independent of the underlying transactional model. Then, we show that nearly half of the chosen benchmarks can be eliminated and evaluate the results by varying the transactional memory model while holding the microarchitecture constant and by varying the microarchitecture while holding the transactional memory model constant. We obtain results for the complete benchmark set and then show how clustering can be used to eliminate redundant benchmarks using subsets as well as the speedup obtained by eliminating the redundant benchmarks. Finally, we show a method that architects can use to choose which programs to run based on the transactional feature that needs to be evaluated.

A. Independence

Data was collected using the methodology described above for LL, EE, and EL transactional models. Figure 1 shows the dendrogram produced from the characteristics in Table I. 10 principal components are retained (Figure 2) that account for 91% of the variance. The resulting clusters are formed using the Euclidean distance between the benchmarks and assembled hierarchically using the single linkage distance. The linkage distance (y -axis) is used to derive the similarity between the benchmarks. The smaller the linkage distance, the more tightly the programs are clustered; 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 no regard to the underlying conflict detection and version management policies. This shows that, regardless of the transactional model, the

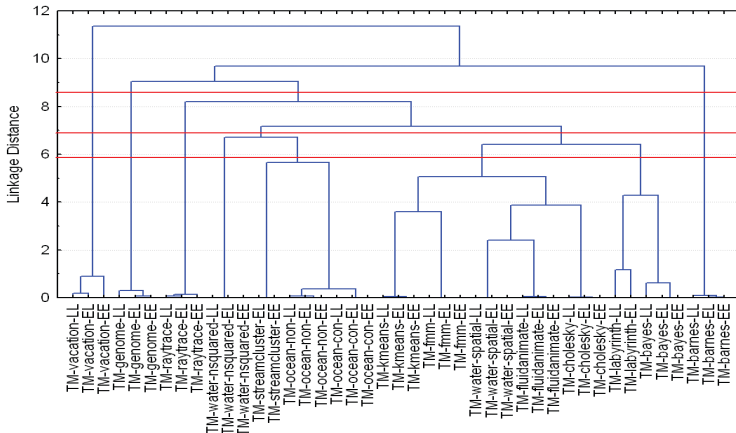


Figure 1. Dendrogram Showing Clustering Based on Characteristics in Table I

characteristics in Table I provide a firm foundation for evaluating the performance of transaction memory programs.

B. Benchmark Subsets

By eliminating benchmarks with similar behavior, researchers can reduce the time needed to evaluate a design. The effectiveness of the subsets is evaluated as the transactional model is varied and as the microarchitecture model is varied. From Figure 1, the linkage distance can be used to select a subset of benchmarks representative of the entire group, allowing researchers to reduce the number of required simulations. Sets of 4, 6, and 8 benchmarks were chosen based on their linkage distance. For the set of 4 benchmarks (*vacation*, *genome*, *barnes*, and *water-N2*), shown in Figure 1, the cutoff was set at a linkage distance of 8.6. For the set of 6 benchmarks (*vacation*, *genome*, *raytrace*, *water-N2*, *barnes*, and *bayes*), a distance of 6.8 was chosen as the cutoff. Finally, a cutoff distance of 5.9 was chosen for the set of 8 benchmarks. This set is comprised of *vacation*, *genome*, *raytrace*, *water-N2*, *ocean-non*, *barnes*, *water-spatial*, and *bayes*. By changing the cutoff value, researchers can effectively choose any subset of benchmarks. Interestingly, the STAMP benchmarks appear to have more diversity than the SPLASH-2 suite with *kmeans* showing some similarity to *fmm*, *water-spatial*, and *cholesky*. [12] suggested using both single-linkage and complete-linkage to give an indication as to whether the clusters are well-defined or are artifacts of the clustering method. To validate the clusters, complete linkage was also used to cluster programs (not shown) and only produces a single deviation in the 4-case subset switching *water-N2* for *bayes* and no deviations in the 6- and 8-case subsets.

TABLE VII. AVERAGE ERROR AND C_v (σ/μ) AS THE TRANSACTION MODEL VARIES

	C_v	S4	S6	S8
Speedup	0.68	4.34	3.95	3.50
Speedup Total Cycles	1.14	5.66	4.95	4.04
Aborted Cycles:Trans Cycles	1.72	7.40	6.36	6.60
Nacked Cycles:Trans Cycles	1.39	13.70	9.96	8.05
Percent Committed	0.32	4.16	3.85	3.57
Average		7.05	5.81	5.15

1) Transactional Model Evaluation

While we would like to show graphs for all of the results we are limited by space. As such, Figure 3 shows the comparison of the complete benchmark set against the NACK cycle ratio (left) and the percentage of committed instructions as the transactional model is varied (right). Table VII shows the average error for all metrics as well as the coefficient of variation across all models. The coefficient of variation (C_v) is included with each metric as an indication of the cross-benchmark variance. A value of less than 1

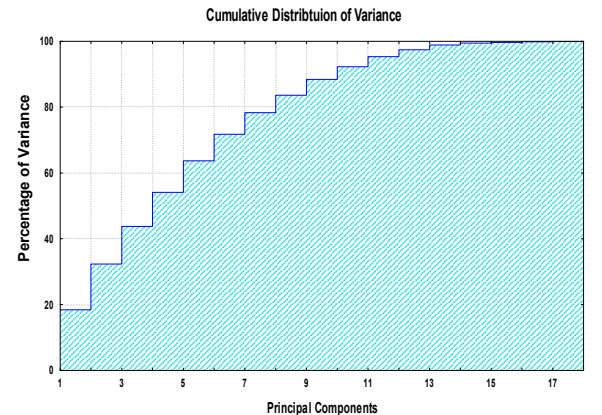


Figure 2. Cumulative Distribution of the Variance By PC

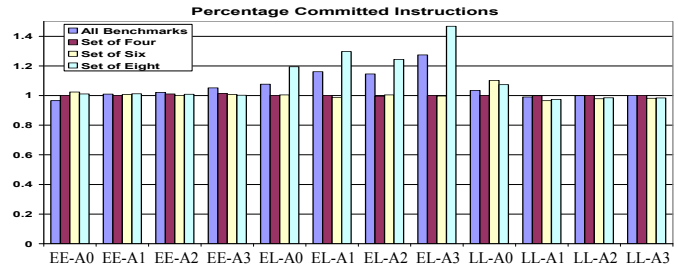
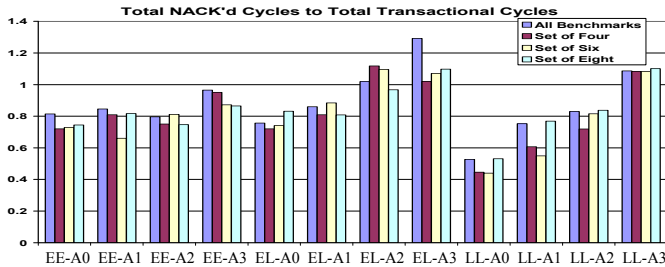


Figure 3. Performance comparison of NACKd Cycles:Trans Cycles (left) and Percentage of Committed Instructions (right) as the Microarchitecture Model is Varied (All Cases Normalized to the Baseline Configuration, Which Can Produce Values Greater Than One)

typically shows little variation while a value larger than 1 indicates that the domain has a high degree of variance. This is important because it shows that the individual program results are diverse and that the error is not low simply because there is no variation in the chosen benchmarks.

The average error for the 4-benchmark subset was 7.05%. As expected, the 6-benchmark performance improves, reducing the average error 5.81%. Adding the 2 additional programs in the 8-benchmark subset offers no significant improvement except in the NACK-trans ratio. Further analysis allows us to draw several conclusions about the effectiveness of the workload characteristics chosen to describe the similarity of the TM workloads. First, we notice that as the transactional memory model is varied, metrics that allow us to capture aggregate information about the overall execution of the workload (e.g. *speedup* and *total speedup*) as well as metrics that focus specifically on transactional performance (e.g. *percent committed*) perform very impressively; even with as few as four representative benchmarks. Finer granularity metrics, such as those that monitor the precise number of cycles aborted or cycles stalled due to NACKs, have higher error rates. This is because these metrics are not governed solely by the aggregate statistical nature of the workload composition, but vary based on both the statistical nature of the workload and specific timing characteristics of the workload. For example, while the overall behavior of two workloads in which thirty percent of the transactions were aborted may be very similar, the actual number of aborted cycles or cycles spent stalled due to NACKs can vary widely between the two dependant upon when the conflicts occurred, which transactions were aborted, how long into the transactions the abort took place, etc. Thus, the similarity of the workloads and the number of benchmarks that must be run varies dependant upon the granularity of the results the architect is looking for. While four benchmarks may be enough to determine the overall execution time, speedup, or commit-percentage, eight benchmarks may be more appropriate for an in-depth analysis of the periods of stall time.

TABLE VIII. AVERAGE ERROR AND C_v (σ/μ) AS THE MICROARCHITECTURE MODEL VARIES

	C_v	S4	S6	S8
Speedup	0.72	11.49	11.56	7.95
Speedup Total Cycles	1.32	11.91	11.98	8.37
Aborted Cycles:Trans Cycles	1.95	18.92	16.05	15.39
Nacked Cycles:Trans Cycles	13.2	10.64	9.23	6.49
Percent Committed	0.37	4.86	5.43	4.57
Average		11.57	10.85	8.55

2) Microarchitecture Model Evaluation

Next, microarchitecture model was varied using Table VI as inputs. Figure 4 shows the comparison of the complete benchmark with the 4-, 6-, and 8-benchmark subsets comparing NACKs (left) and percentage of committed instructions (right). Although the trend is the same as we increase the number of programs, the total

error is higher than that shown in section 5.2.1. The 4-benchmark subset had an average error 11.57%; the 6-benchmark subset had an average error of 10.85%, and the 8-benchmark subset had an average error 8.55% across all of the dimensions. The complete error chart is shown in Table VIII.

Because the analysis parameters focus on transactional behavior, as the underlying microarchitecture varies in ways not directly related to the transactional performance, differences within the benchmarks that occur outside the bounds of transactions result in a higher overall error across our metrics. We see that the error across metrics that are a direct result of transactional performance (i.e. aborted cycles, NACKed cycles, and percent committed) increase, but much less than those that incorporate both transactional qualities and microarchitecture features (i.e. speedup and speedup total cycles).

TABLE IX. SPEEDUP OF SUBSETS TO FULL SUITE

	Speedup
S4	5.73
S6	2.58
S8	2.38

3) Time Savings

Table IX shows the speedup of the subsets compared to running the entire set of benchmarks. As can be seen, there are considerable timesavings between running the full set of benchmarks and those running the 4-, 6-, and 8-benchmark subsets. The speedup decreases as the number of included benchmarks is increased, allowing architects to make a tradeoff between fidelity and simulation time.

C. Variable Subsets

As was shown in section 5.2, the complete set of transactional characteristics chosen for clustering allows an architect to substantially reduce the number of simulated benchmarks and obtain coarse-grained results with minimal error. Fine-grained results can be obtained with minimal error as well. However, in many cases the number of simulations must be increased. The difference lies in the fact that the complete set of transactional characteristics is targeted at describing the similarity of all aspects of the workload. For some evaluations, architects may not be interested in the complete characteristics of a program. For example, if the target of an evaluation is the conflict manager, the designer may only want to choose those programs with potentially large amounts of contention. In this section, we show how a subset of the characteristics in Table I can be used to select a group of benchmarks that stress a specific architectural feature.

Figure 5 shows the dendrogram produced if only the transaction size characteristic is used as input into the PCA algorithm. Here, the linkage distances are relatively low (implying strong clustering) and that the architect can obtain good performance estimates by using a small number of simulations. If

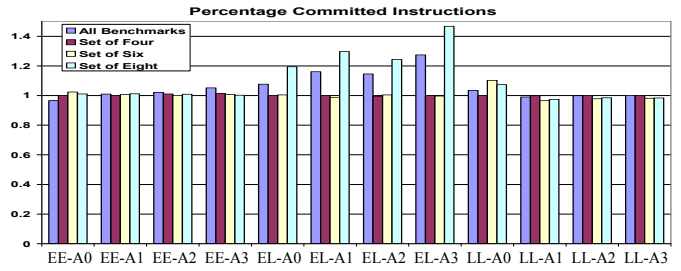
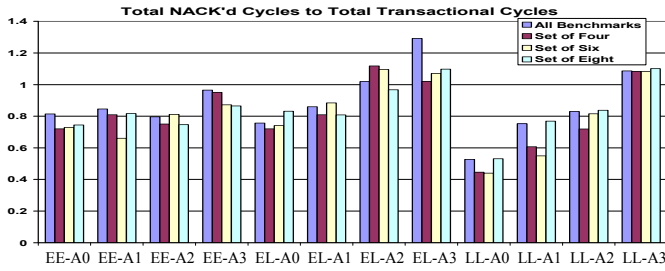


Figure 4. Performance comparison of NACK'd Cycles:Trans Cycles (left) and Percentage of Committed Instructions (right) as the Microarchitecture Model is Varied (All Cases Normalized to the Baseline Configuration, Which Can Produce Values Greater Than One)

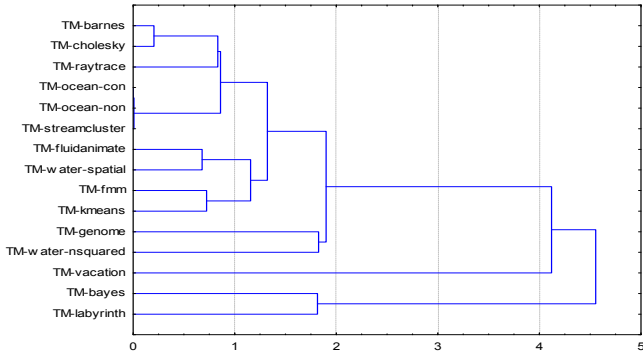


Figure 5. Dendrogram Based on Transaction Size

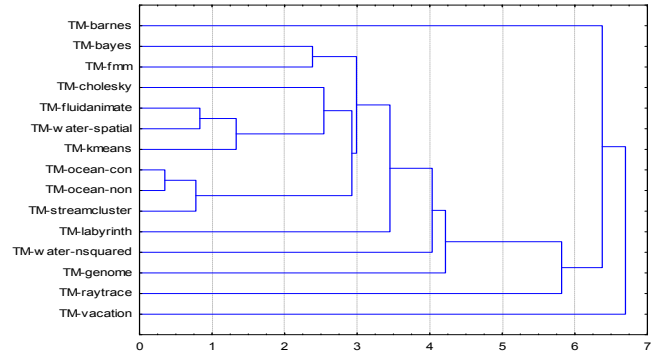


Figure 6. Dendrogram Based on Conflict Density

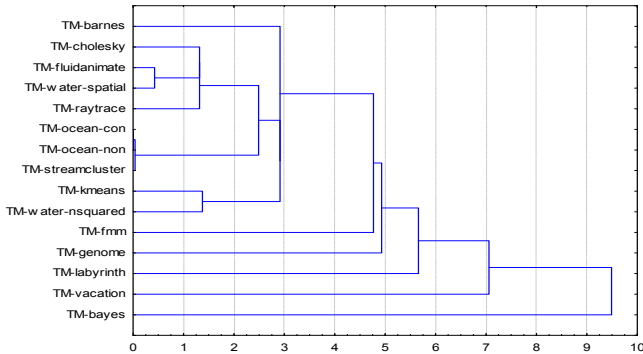


Figure 7. Dendrogram Based on Read- and Write-set Sizes

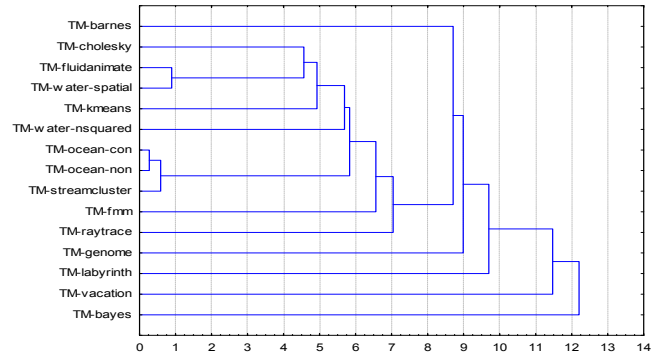


Figure 8. Dendrogram Based on Tx size, R-/W-set Sizes, and Conflict Ratio

we compare the suggested clustering to our workload analysis table (Table III) in section 4.3, we see that the clusters largely agree with our simulated results. *Bayes* and *labyrinth* are similar to one another, but largely different from *vacation*, which is different from *genome*, *water-N2*, and the rest of the SPLASH-2 suite. Of particular interest is the relative proximity of *water-N2* and *genome*. Had we simply elected to use the aggregated average values presented in the workload characterization table to determine our clustering, it is unlikely we would have seen much similarity in the transactional size between these two benchmarks. However, over the lifetime of the program, more than half of the transaction sizes are the same for *water-N2* and *genome* and the average size of a transaction in *genome* is artificially increased by several large outliers.

Figure 9 shows a plot of the first two principal components (which comprise more than 70% of the total variance). Here, the first principal component is positively dominated by transaction sizes between 5 and 25 instructions and negatively dominated by transactions larger than 390k instructions. The second component is positively dominated by transactions between 5 and 25 instructions and negatively dominated by transactions between 625 and 3125 instructions. From this figure, we can see that there are two very strong clusters, one weak cluster, and two isolated

programs. Although *bayes* has the most diversity in transaction size, both it and *labyrinth* are dominated by large transactions. The other clusters are comprised of programs that are dominated by one or two specific transaction sizes. As with the cumulative result, the STAMP benchmarks show a wider range of transaction sizes than SPLASH-2 or the two PARSEC benchmarks.

While transaction size is a very simple example, and it may be possible to detect such patterns directly from the input data in such cases, as the complexity of the characteristics increases this becomes infeasible. If we return to our original example, an architect that is interested in designing a conflict manager and is specifically looking for benchmarks of interest as they relate to contention, the transaction size is just one small piece of the puzzle. Using our transactional characteristics, actual contention is a function of the transaction size, the size of the read- and write-sets and the conflict density of those read- and write-sets. Figures 6 and 7 show the dendrograms for read/write conflict density and read/write set size, respectively while Figure 8 shows the combination of transaction size, read/write set size, and read/write conflict density, what will be referred to as the contention domain. If we compare these dendrograms, we can see that there are some similarities between the clusters for several of the characteristics, yet there are also distinct differences. The PC-plots help reveal

narrow set of design features that should be considered. This paper takes the opposite approach by providing a set of architecturally independent characteristics and allowing researchers to choose which features they believe are most important feature for an evaluation. In [4], Bobba et al. explore different performance pathologies that can arise across different transactional models. Our work has benefited from this study by incorporating several of the fixes to the protocols in an effort to provide a fair cross-dimension comparison. In this regard, the framework used in this paper has been designed to emulate different HTM systems such as Stanford's L/L system, TCC [17]; E/L systems like MIT LTM [18] and Intel/Brown VTM [19]; and E/E systems like CC DBMSs [20], MIT UTM[18], and LogTM [21].

VII. CONCLUSION

With no universally accepted transactional memory program suite, researchers are left in the dark about what may or may not constitute an acceptable benchmark. If the wrong benchmarks are chosen, they may not be able to stress the design in the ways that the architect expects or may provide superfluous results. In fact, previous research has shown that many of the SPEC CPU programs exhibit similarities in conventional design spaces and are, in fact, redundant. In this work, we have provided a set of characteristics that architects can use to evaluate transactional memory programs and have shown how programs can be selected based on all or a subset of these traits. Using these characteristics, we show that using SPLASH-2 to evaluate a transactional memory system may not expose all of the benefits or flaws in a design and that the PARSEC programs used in this evaluation have features that make them redundant when using SPLASH-2. When the STAMP benchmarks are considered, there is more diversity in the entire benchmark set but is still limited in its scope; more emphasis should be placed on the design and implementation of transactional memory programs if this field is going to continue to grow. The program traits and mathematical methods described in this paper can be used to guide and evaluate the comprehensiveness of these new programs. Finally, we show how picking and choosing feature sets can be used to select a set of programs that can stress a particular element in a design, allowing architects to quickly select a benchmark to test a specific design implementation.

ACKNOWLEDGMENTS

This work is supported in part by NSF grant CNS-0834288, SRC grant 2008-HJ-1798, and by three IBM Faculty Awards. The authors acknowledge the UF HPC Center for providing computational resources.

REFERENCES

- [1] R. H. Saavedra and A. J. Smith, "Analysis of Benchmark Characteristics and Benchmark Performance Prediction," *ACM Trans. Computer Systems*, 1996.
- [2] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Quantifying The Impact of Input Data Sets On Program Behavior and Its Applications," *Journal of Instruction Level Parallelism*, 2003.
- [3] M. Herlihy and J.E. Moss, "Transactional Memory: Architectural Support For Lock-free Data Structures," *In Proceedings of International Symposium on Computer Architecture*, 1993.
- [4] J. Bobba, K. Moore, L. Yen, H. Volos, M. Hill, M. Swift, and D. Wood, "Performance Pathologies in Hardware Transactional Memory," *In Proceedings of the International Symposium on Computer Architecture*, 2007.
- [5] A. McDonald, J. Chung, H. Chafi, C. C. Minh, B. Barlstrom, L. Hammond, C. Kozyrakis and K. Olukotun, "Characterization of TCC on Chip-Multiprocessors," *In Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, 2005.
- [6] C. H. Romesburg, "Cluster Analysis for Researchers" Lifetime Learning Publications, 1984.
- [7] SESC: A Simulator of Superscalar Multiprocessors and Memory Systems with Thread-Level Speculation Support, <http://sourceforge.net/projects/sesc>
- [8] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *In Proceedings of International Symposium on Computer Architecture*, 1995.
- [9] C. C. Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis, and K. Olukotun, "An Effective Hybrid Transactional Memory System With Strong Isolation Guarantees," *In Proceedings of International Symposium on Computer Architecture*, 2007.
- [10] C. Bienia, S. Kumar, J.P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," *Princeton University Technical Report*, 2008.
- [11] StatSoft, Inc, STATISTICA. <http://www.statsoft.com>
- [12] M. Hughes, "Exploration and Play Re-visited: A Hierarchical Analysis," *International Journal of Behavioral Development*, 1979.
- [13] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Designing Computer Architecture Research Workloads," *Computer*, 36, 2, pp. 65-71, 2003.
- [14] H. Vandierendonck and K. Bosschere, "Many Benchmarks Stress the Same Bottlenecks," *In Proceedings of the Workshop on Computer Architecture Evaluation Using Commercial Workloads*, 2004.
- [15] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of Redundancy And Application Balance In The SPEC CPU2006 Benchmark Suite," *In Proceedings of International Symposium on Computer Architecture*, 2007.
- [16] J. Chung, H.Chafi, A. McDonald, C. C. Minh, B. Carlstrom, C. Kozyrakis, and K. Olukotun, "The Common Case Transactional Behavior of Multithreaded Programs," *In Proceedings of International Conference on High Performance Computer Architecture*, 2006.
- [17] L. Hammond, *et. al.*, "Transactional Memory Coherence and Consistency," *In Proceedings of the International Symposium on Computer Architecture*. 2004.
- [18] C. Ananian, K. Asanovic, B. Kuszmaul, C. Leiserson, and S. Lie, "Unbounded Transactional Memory," *In Proceedings of the International Conference on High Performance Computer Architecture*, 2005.
- [19] R. Rajwar, M. Herlihy, and K. Lai, "Virtualizing Transactional Memory," *In Proceedings of International Symposium on Computer Architecture*, 2005.
- [20] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions Of Consistency And Predicate Locks In A Database System," *Communications ACM* 19, 1976.
- [21] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood, "LogTM: Log-based Transactional Memory," *In Proceedings of the International Symposium on High-Performance Computer Architecture*, 2006.
- [22] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford Transactional Memory Applications for Multi-Processing," *In Proceedings of International Symposium on Workload Characterization*, 2008.
- [23] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors," *In Proceedings of International Symposium on Workload Characterization*, 2008.
- [24] G.H. Dunteman, "Principal Component Analysis," Newbury Park, California: Sage, 1989.
- [25] J. Poe, C. Cho, and T. Li., "Using Analytical Models to Efficiently Explore Hardware Transactional Memory And Multi-core Co-design," *In Proceedings of International Symposium on Computer Architecture and High Performance Computing*, 2008.