Cache-Aware Approximate Computing for Decision Tree Learning

Orhan Kislal
The Pennsylvania State University
University Park, PA, USA
omk103@cse.psu.edu

Mahmut T. Kandemir
The Pennsylvania State University
University Park, PA, USA
kandemir@cse.psu.edu

Jagadish Kotra
The Pennsylvania State University
University Park, PA, USA
jbk5155@cse.psu.edu

***Abstract*—The memory performance of data mining applications became crucial due to increasing dataset sizes and multi-level cache hierarchies. Decision tree learning is one of the most important algorithms in this field, and numerous researchers worked on improving the accuracy of model tree as well as enhancing the overall performance of the learning process. Most modern applications that employ decision tree learning favor creating multiple models for higher accuracy by sacrificing performance. In this work, we exploit the flexibility inherent in decision tree learning based applications regarding performance and accuracy tradeoffs, and propose a framework to improve performance with negligible accuracy losses. This framework employs a data access skipping module (DASM) using which costly cache accesses are skipped according to the aggressiveness of the strategy specified by the user and a heuristic to predict skipped data accesses to keep accuracy losses at minimum. Our experimental evaluation shows that the proposed framework offers significant performance improvements (up to 25%) with relatively much smaller losses in accuracy (up to 8%) over the original case. We demonstrate that our framework is scalable under various accuracy requirements via exploring accuracy changes over time and replacement policies. In addition, we explore NoC/SNUCA systems for similar opportunities of memory performance improvement.**

I. Introduction

Decision tree learning is a well-studied predictive modeling approach that is widely used in data mining and related fields. While prior research [36], [32], [18], [19] has investigated different implementations of decision tree learning, evaluating and optimizing its performance in the context of emerging multi-level on-chip cache hierarchies and modern main memory systems took much less attention. Most decision tree learning implementations, like many other data mining algorithms, are memory-bound since they typically apply computationally-trivial operations to large amounts of data. In doing so, they require multiple accesses to the same data in order to create the tree levels. This increases the dependency on memory performance and, consequently, on the effectiveness of data access optimizations.

Prior work suggested different strategies to handle costly memory accesses in data-intensive applications, including memory request scheduling [20], [21], [34], on-chip network optimizations [28], [9], and aggressive cache optimizations (at both architectural and software levels) [10], [23], [8], [31]. While these optimizations are successful in certain cases, their effectiveness is ultimately limited as they try to preserve the *correctness of execution*. Approximate computing [12], [33], [37], [39], an emerging research area, instead suggests that, by relaxing the exact correctness requirement, it may be possible to reach a faster (and possibly more energy efficient) execution in certain applications.

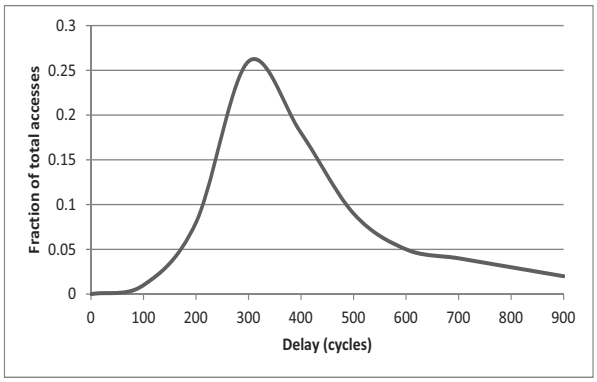


Fig. 1. The data access distribution for our decision tree learning implementation used in this work on the NoC/SNUCA system (details of our experimental platform will be given later in Section IV). We see that some data accesses are much costlier compared to others.

An important question then, in the context of decision tree algorithms, is whether approximate computing can be employed to reduce their memory access latencies without significantly affecting the correctness of the original applications. Focusing on one particular implementation of decision tree learning [19] and skipping some portion of costly data accesses (as our approximate computing strategy), this paper starts with two critical observations:

- The inaccuracy resulting from skipping data accesses depends more on the number of data accesses skipped, rather than which specific data accesses are skipped.
- In contrast, the performance benefits achieved through data access skipping depends strongly on which specific data accesses are skipped.

The reason for the first observation is the fact that every data point is accessed multiple times throughout the execution. Even if a data point is skipped and replaced with a (possibly incorrect) prediction, there is a very high probability that the same point will not be skipped again, and it will eventually

```
for every level do
  for every node do
    for every attribute do
      for every point do
        Load class value
        Calculate gini impurity
      end for
    end for
  end for
  Branch into new nodes
end for
```

Fig. 2. Pseudo-code of the data intensive section of our decision tree learning algorithm. Note that the dataset has to be sorted for every attribute beforehand. Traversing the data points in order makes it possible to keep track of the impurity of any given branch.

contribute to the decision tree. The reason for the second observation is the fact that costs of different data accesses (even in the case of data-parallel applications) are not uniform in current multicore and manycore systems. Specifically, a data access may hit at any level in the on-chip cache hierarchy (composed of L1, L2 and L3 in modern systems), or miss in all of them. In the latter case, a main memory request is made, whose cost depends on the on-chip network latency, memory queuing latency, and whether we hit in the row-buffer or not. A distribution of data access latencies in a typical execution of our target decision tree application is plotted in Figure 1 (note that according to prior research, such patterns are not unexpected [41]). One can observe from this graph that, data accesses exhibit a great deal of variations in their latencies. Consequently, when considering the two observations above, two alternate approximate computing strategies (for a given algorithm) that skip the *same* amount of data accesses are expected to result in similar inaccuracies but *significantly different* performance improvements. Motivated by this result, our goal in this work is to explore the potential performance improvements that can be achieved via data access skipping. Specifically, our proposed approach tries to skip a fraction of costly data accesses (last-level cache misses) in an attempt to maximize the performance benefits under a given inaccuracy bound. Our detailed experiments with this approach applied to a sample decision tree learning implementation show that:

- A small portion of the data accesses has a negligible impact on accuracy but a significant impact on performance. Specifically, skipping 3% of the data accesses reduces the accuracy by only 4% – 6%, but improves the memory performance by 50% – 55%. These savings translate to an average execution time improvement of 15%.
- Randomly skipping data accesses (i.e., without considering their latency costs) offers considerably smaller improvements compared to our scheme. For example skipping 3% of the data points "randomly" improves the memory performance only by 4% – 5%.
- Our cache miss skipping scheme is applicable for both uniform and non-uniform cache hierarchies. In fact skipping 3% of the data points improves the memory performance by 45% – 48% and the overall performance by 13% – 14% on the NoC/SNUCA based system.

The remainder of this paper is structured as follows. A general overview of decision tree learning is provided in Section II. Section III explains the details of our data access skipping strategy and discusses potential (hardware and software based) implementation options. The results from our experimental evaluation are presented in Section IV. We discuss prior work relevant to our study in Section V, and we conclude the paper in Section VI with a summary of our main observations and a brief discussion of the planned future work.

II. Decision Tree Learning

Classification via decision tree is achieved by traversing the tree from the root to a leaf node. Each level represents a test for one of the *attributes* of the data point, and each branch represents one of the *classes* available for the aforementioned attribute. One of the most important factors related to the accuracy of the decision tree is the way the branches are created. The algorithm evaluates every attribute via a statistical test, and calculates how accurate the model will be by assuming that the algorithm is completed at that point. While there are a number of different tests, such as information gain and variance reduction, that are designed to determine the most beneficial branch, studying the accuracies of these methods is beyond the scope of this study. The test chosen for our baseline algorithm is a test to find the branches with minimum *gini impurity* [6]. Gini impurity considers all of the possible classes for a given set and calculates the probability of "wrongly classifying" a random data point if its class is selected randomly from the available set. The most accurate case is when the gini impurity reaches zero, in which case all data points have the same class. A pseudo-code of the standard decision tree learning algorithm implementation is given in Figure 2. It is important to note that, after the initial sorting step, the values of the attributes are irrelevant to the calculation of the gini impurity; they will be required only as the end points of the branches for the testing phase. Since each attribute operates on a different list of *class* values, parallelizing this implementation is straightforward. Synchronizing at the end of every level to find the best possible branches for every node is enough to ensure the validity of the results.

Decision trees are employed in a wide variety of scenarios due to various factors. The basic algorithm can be improved to operate with real values instead of a fixed set. The model may have real-value outputs as well, even though the most common cases involve Boolean classification. The algorithms' robust nature allows them to tolerate errors or missing attributes in the training data, which is quite common when considering real-world scenarios.

The specific decision tree learning implementation used in this work is ScalParC [19] from the MineBench benchmark suite [44]. This formulation of decision tree based classification process is developed to address the scalability issues exhibited by the SPRINT classifier [40]. ScalParC uses a distributed hash table to reduce the memory requirements

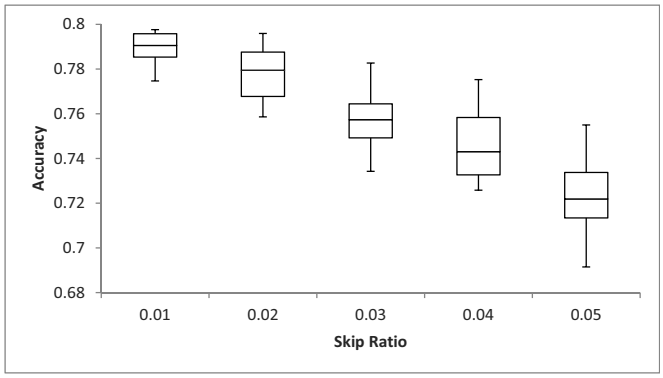


Fig. 3. The accuracy of models after skipping varies through repeated experiments due to the randomized nature of the optimization. The baseline accuracy values for this dataset (small) can be found in Figure 12.

and to ensure that the algorithm is scalable in terms of both runtime and memory requirements. We decided to use the implementation from the MineBench suite because the original implementation employs MPI to parallelize the workload, whereas the MineBench version provides a more suitable parallelization option for our target system via openMP.

III. Cache Miss Skipping

Data access latency is not constant in modern multicore and manycore architectures. Most existing systems include a multi-layer cache hierarchy involving L1, L2, and possibly L3 caches. In addition, a data access missing in the cache can cause a variable latency based on parameters such as row-buffer locality, bank parallelism, memory queue length, and memory (DRAM) access latency. Consequently, if it is permissible to skip a certain number of data references, it makes sense (from a performance viewpoint) to skip the "most costly" ones. Motivated by this observation, the data access skipping strategy employed in this work skips a fraction of the last-level cache misses (L2 in our case). That is, in our optimization, a data access that misses in the last-level cache is called a *costly access*, whereas a data access that hits in one of the on-chip caches is called a *cheap access*. Our approach selects a subset of the costly accesses to skip, thereby trading off accuracy for performance benefits. It is to be noted, however, that one could also adopt a different costly/cheap model, e.g., memory accesses that miss in the row-buffer could be considered costly (with the rest of the misses being considered cheap) and our approach would still be applicable. Our choice in this paper is driven by the observation that there are various techniques based on compiler or hardware support for predicting/identifying cache misses, making it easy to distinguish between the costly and cheap accesses.

In its original form, a decision tree algorithm traverses the entire input dataset in order to train the model for future use. The nature of this process is imprecise, and while creating deeper decision trees can increase the accuracy in most cases, such an approach may also lead to overfitting the model for training data, thereby reducing the overall accuracy. Instead of spending more time on developing a single model, most modern applications create multiple models and compare the models' accuracies to identify the best one. This approach overcomes the overfitting problem, and also leads to shorter decision trees, thereby reducing traversal time in future instances. Since these applications already have mechanisms to help them achieve their target accuracies, each individual model's level of accuracy loses importance in contrast. To support this claim, we conducted multiple experiments with varying degrees of data access skipping and measured their accuracies. A *skip ratio* of X% means that, for each data access, there is a X% chance that this particular access will be skipped (i.e., that the access will not be issued to the off-chip memory). Note that a skipped data access will *not* incur a memory access cost and it will assume a value based on a *prediction scheme* (how this prediction is implemented is discussed later in this section). Figure 3 plots the accuracy values of 20 experiments for each skip ratio (1% to 5%). Experiments using other datasets yield similar results. Our experiments show that the accuracy variance among various runs of a given skip ratio is fairly limited. The discrepancy becomes more noticeable as the percentage of skipped points increases, but even using the most aggressive policy (skip ratio: 5%), the accuracy gap between the best and the worst case scenarios is less than 8%. These results form the basis of the question we want to address: If skipping a certain number of data points can be considered acceptable, how should we choose which data points to skip in order to maximize performance improvement?

As mentioned earlier, the memory accesses in our decision tree learning algorithm take up a significant portion of the overall execution time, and skipping some of the costly data accesses (last-level cache misses) might result in a considerable boost in performance. The goal of this study is to establish a performance/accuracy tradeoff framework by which the user can explore various levels of aggressiveness in cache miss skipping optimization. In doing so, he or she can decide to sacrifice a limited amount of accuracy per model in order to obtain shorter execution times and in the end efficiently identify a satisfactory model.

At a high level, we employ a *Data Access Skipping Module* (DASM), which is configured to skip a certain fraction of L2 misses (off-chip memory accesses). A pseudo-code of the decision tree learning algorithm that employs DASM is given in Figure 4. First, the user identifies the target memory accesses and marks them for the compiler. For our decision tree learning algorithm, we mark the loads for the $class$ values of the $points$ since the attribute values are used only during the initial sorting phase. This is necessary because a blanket approach in which every memory access is considered a candidate for skipping may lead to serious accuracy and performance problems. In addition, most of the memory accesses involving temporary variables are not costly enough to be considered for skipping. Throughout the execution, if a memory access from the set of marked memory accesses is missed at the L2 cache, the data access skipping module will intervene. Depending on the

```
for every level do
  for every node do
    for every attribute do
      for every point p do
        Load p.class with DASM
        if p.class is NaN then
          p.class = (p − 1).class
        end if
        Calculate gini impurity
      end for
    end for
  end for
  Branch into new nodes
end for
```

Fig. 4. Pseudo-code of the decision tree learning algorithm with DASM. The last point heuristic is used for replacement. Reading the class information for the point is achieved through a special command that enables DASM. If the access is skipped the value will not be in the expected range of classes which will force the approximation heuristic to replace the value.

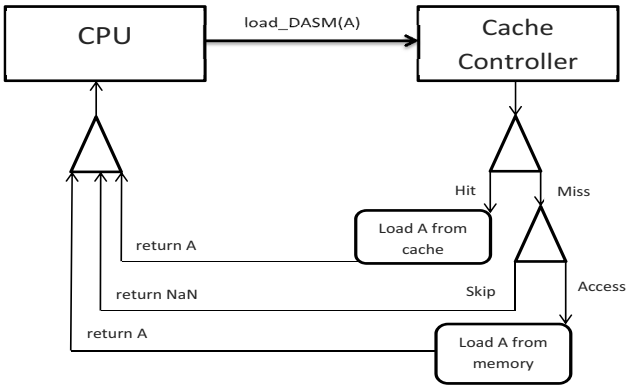


Fig. 5. DASM implementation at the hardware level.

"skip ratio" employed, DASM will decide to either let the access proceed or return a predicted value.

DASM employs a hardware-based implementation that is able to intercept the requests for costly memory accesses and replace them when skipping is deemed beneficial. Similar techniques have been investigated for different optimization goals [25]. As shown in Figure 5, DASM is implemented as an addition to the cache controller. When a marked load operation is executed, the cache controller checks if the access is a hit or miss. A cache hit will proceed as expected, but a cache miss will alert DASM while waiting for the actual value from the memory. In this case, DASM might force the cache controller to return an invalid value based on a probability ($skip\ ratio$) defined by the user beforehand. Handling the invalid value will be left to the user based on the specifics of the use case. Please note that this operation does not have any effect on the cache coherency; the actual value of the skipped data is not changed in the memory or non-shared caches. In fact, our motivation relies on the fact that a skipped point will be accessed again in the subsequent iterations and will contribute its actual value, thus reducing the effect of the skip on the accuracy of the final model.

Even though we have argued that the value of a particular skipped data point has a negligible impact on accuracy, an accurate prediction would further reduce the negative effect of skipping. While assuming a "random" value from the variables' possibility space is a valid solution, it would likely be more beneficial for the accuracy of the model to use a simple heuristic. It is important to note that this heuristic should be light-weight and should not result in extra memory accesses. To this end, we employ a heuristic based on the observation that a leaf in a perfect decision tree has a group of variables with the same target values. Since the decision tree algorithm dissects the dataset into pieces with higher uniformity, it is plausible that an unknown variable located anywhere within the dataset has a value similar to that of the previous data point. The sequential traversal of the points increases the probability that this study's point of inspiration (the replacement for our skipped value) has been accessed recently and still exists at an easy-to-access location (L1 cache). Thus, our last-point heuristic predicts that a skipped data point will have the same *class* value as the previous point in the same attribute order. The last-point heuristic is designed with the assumption that data points are traversed sequentially. While this is a widely used method, we acknowledge that not every algorithm will adhere to this principle; thus, we have extensively explored the performance and accuracy of random replacement to provide a baseline for any potential application.

Please note that the logic behind DASM and both of the replacement policies is independent of gini impurity testing. Our incorrect predictions after skipping a data point are practically indistinguishable from an erroneous entry or an outlying case in the dataset. Since similar occurrences are common for data mining applications, testing algorithms such as the gini impurity are developed and employed with that assumption in mind. To the best of our knowledge, the gini impurity test do not exhibit any specific tolerance for incorrect data points; it has been selected only because of its ubiquity.

IV. Experimental Evaluation

We used the ScalParC application from the MineBench suite as a baseline [44] for our experiments to show the viability of the data access skipping module. We employed the same synthetic datasets as MineBench does (those generated by the IBM Quest Data Generator) for the sake of fairly evaluating our proposed technique. There were 125,000, 250,000, and 500,000 points with 32 dimensions per point grouped into small, medium and large datasets, respectively. These datasets are divided into training and testing sets. The training set is created by selecting 80% of the dataset randomly, and the testing dataset consists of the remaining points unless otherwise noted. We used two different simulated systems for our analysis. The first one is based on Intel Nehalem Xeon X5550 with a uniform cache architecture. The second one is a Network-on-Chip (NoC) based system with static non-uniform cache access policy (SNUCA [17]). The cache hit rates were collected using the GEM5 [2] simulation tool. We also employed the Sniper full system simulation tool [5] to collect data to be used in our motivation section as well as to

TABLE I
EXPERIMENTAL SETUP.

	Flat-Cache Hierarchy	NoC/SNUCA Based System
Processor	4 in-order cores	4x8 out-of-order cores
L1 Caches	Private (data and instruction)	
L1 Cache Size	32 KB	
L1 Cache Latency	3 cycles	
L1 Block Size	64 Bytes	
L2 Caches	Private	
L2 Cache Size	256 KB	512 KB
L2 Cache Latency	15 cycles	10 cycles
L2 Block Size	64 Bytes	
Memory Configuration	DDR-800, Memory Bus Multiplier: 5, Bank Busy Time: 22 cycles, Rank Delay: 2 cycles, Read-Write Delay: 3 cycles, Memory CTL latency: 20 cycles, Refresh Period: 3120	
Cache Coherency Protocol		MOESI CMP DIRECTORY
NoC Parameters		5-stage router, flit size: 128 bits, buffer size: 5 flits,

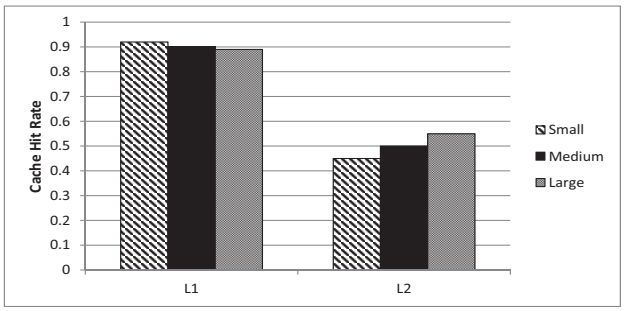
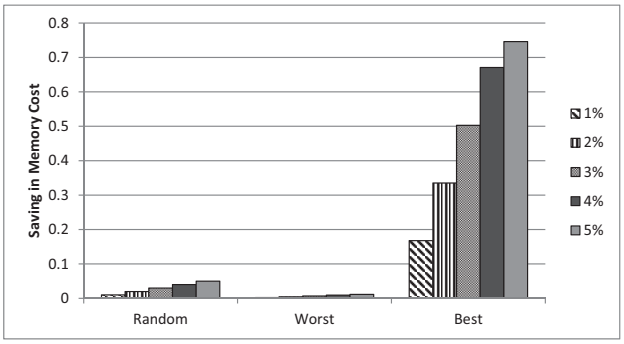


Fig. 6. Breakdown of the L1 and L2 cache hit rates for each dataset.

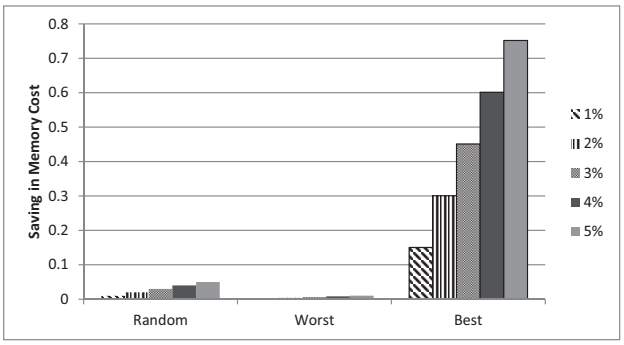
verify our results from our earlier experiments. The specifics of our simulated systems are given in Table I. For the majority of our experiments, we use the first system. The memory performance results for the SNUCA based system are also explored fully, but the detailed accuracy analysis is omitted, since the effect of DASM on accuracy is not dependent on the architecture.

A. Performance Analysis

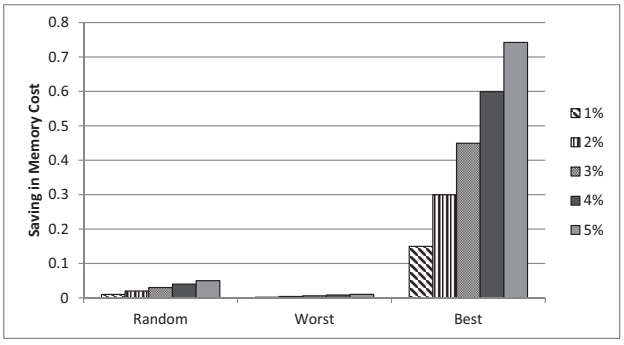
1) Uniform Chip Architecture: To begin our analysis on the memory performance improvements DASM provides, we collected the cache hit rates for the baseline implementation (see Figure 6). It should be noted that, all of the datasets that we used during these experiments are larger than the total L2 cache specified in the simulator. The increase in the L2 hit rates as the dataset size increases may appear counterintuitive; however, the L2 hit rates were not computed using the total number of accesses but the number of accesses that reach the L2 cache (L1 misses). These cache hit rate values are not outside the expected ranges for memory-intense



(a) Savings in data access cost for the small dataset.



(b) Savings in data access cost for the medium dataset.



(c) Savings in data access cost for the large dataset.

Fig. 7. The increase in memory performance when using data access skipping under three different datasets. DASM follows the best case. Note that 5% of the small dataset will be large enough to cover all L2 misses in that dataset.

data mining applications. Using these values and the cache access latencies given in Table I, we compute the number of cycles the application spends on memory accesses. This forms our baseline, and indicates in a sense how many cycles could be saved by skipping data accesses.

We explored 5 different skip ratios (1% - 5%) and 3 different skip scenarios (see Figure 7). The random case (the first group of bars) assumes that the data accesses to be skipped are chosen randomly, with no respect to the cache hit/miss status. The worst case (the second group of bars) assumes that every

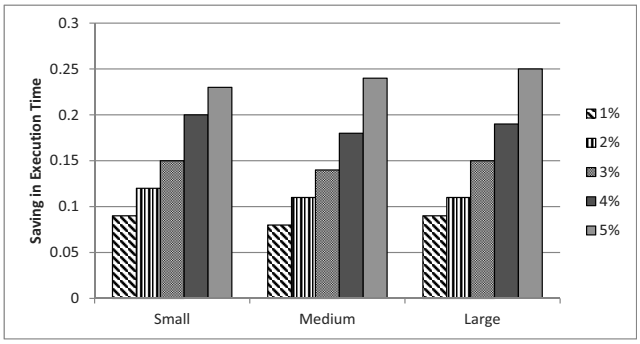


Fig. 8. The improvement in execution time via data access skipping for various datasets.

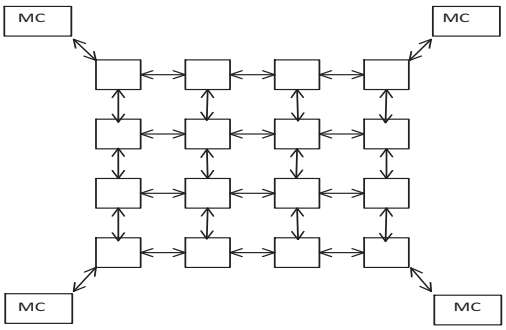


Fig. 9. A $4x4$ NoC based multicore system. Every rectangle represents a core, L1 and L2 caches and a router. MC denotes memory controller.

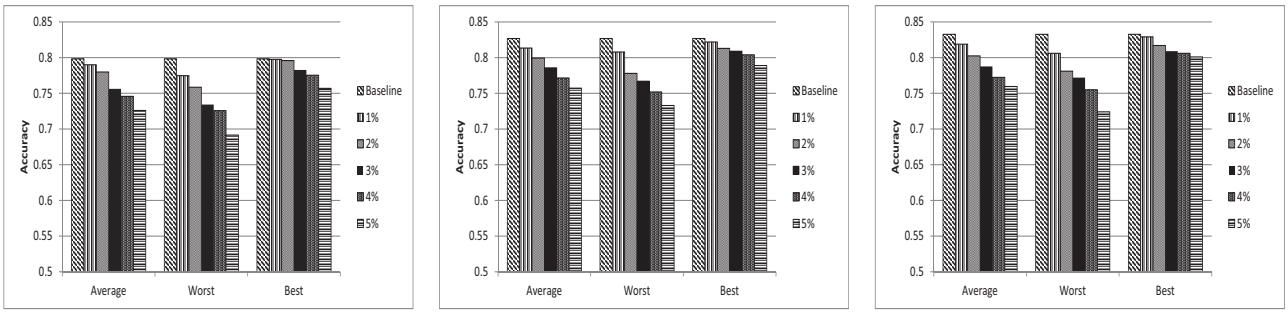
skipped access is an L1 cache hit. The best case (the last group of bars) assumes that every skipped access is an off-chip memory access. The skip ratios represent the number of skipped points as a fraction of the total dataset size. Using the aforementioned cache hit rate values and latencies; we estimated how the memory access costs will be reduced for every skip ratio and scenario compared to the baseline case. The results were similar across the various dataset sizes; thus, we discuss our findings in general terms. As expected, the worst case offered very little increase in performance. On the other hand, the disparity between the random and best cases underlines the importance of skipping more costly data accesses. We see that the time spent on data accesses gets reduced by 45% – 50% when there is a moderate amount of skipping (at 3%), and by up to 76% when there is a high amount of skipping (at 5%).

To complete our analysis of the effect of DASM on performance, we calculated the savings in execution time for various skip ratios and datasets. The results plotted in Figure 8 indicate that skipping 3% of the costly data accesses improves the overall performance by 15% on average for all 3 database sizes. Further, the savings in performance rise up to 25% with a 5% skipping ratio.

2) Non-uniform Chip Architecture: Our second target architecture is a network-on-chip (NoC) based multicore system. An example of such a system is shown in Figure 9. This NoC system employs a number of nodes, memory controllers and connections in between. Each node represents an out-of-order core, L1 caches for instruction and data, an L2 cache, and a router to handle the communication with the neighboring nodes. These nodes are connected in a 2D grid fashion. Since DASM focuses on off-chip memory accesses, the details of the L2 cache and memory controllers are of utmost importance. The L2 cache employs a banked organization scheme and our target cache mapping scheme, static non-uniform cache access (SNUCA [17]), maps every cache bank to an equal sized memory statically. The memory controllers handle the transfer of data to-and-from the main memory and are traditionally placed at the periphery of the grid. Each of the memory controllers is tasked with a memory channel which might include one or more group of memory banks (ranks). The memory banks provide multiple simultaneous access, but the address and data busses are shared.

Using the SNUCA mapping scheme provides a unique opportunity to utilize DASM. Even if a data access can be found in L2 cache, the cost of retrieving it might be very high if the distance (on NoC) between the two nodes involved is too large. On the other hand, if the skip ratio is not high enough to cover all of the off chip memory accesses, it might be beneficial to skip the accesses mapped to further memory controllers rather than closer ones. We explored both of these options in our experiments by calculating the distribution of the data access latencies and skipping the most costly ones. The distribution is provided in Section I (Figure 1). We omit the L1 and L2 cache hit rates for the baseline algorithm as they are effectively the same (within some small margin of error) as the previous system. Our experiments with various skip ratios and different datasets show that DASM is just as effective with NoC/SNUCA system as with the uniform (flat) cache system discussed in Section IV-A1 (see Figure 10). A medium level of aggressiveness in skipping policy (skip ratio of 3%) is sufficient to reduce the overall cost of memory accesses by 45% to 48%. Further, we studied the effect of variable cost accesses at higher and lower skip ratios. The impact of skipping only 1% of data accesses is still significant by itself. The next three segments (2%, 3%, and 4%) are not as effective since the data accesses with the highest costs have already been eliminated. The effects of these three segments are similar to each other, as misses with similar costs can be found at various distances from a given core. However, this trend does not extend to more aggressive policies. The performance improvement gained by increasing the skip ratio from 4% to 5% is not as significant, since most of the costly accesses have already been eliminated by skipping 4% of the data accesses.

Now, we discuss the effect of our savings in memory performance on the overall execution time. Figure 11 shows that a moderate skip ratio such as 3% offers 13% - 14% decrease in execution time. The performance can be improved by up to 20% via skipping 5% of the data accesses.



(a) Accuracy comparison for small dataset. (b) Accuracy comparison for medium dataset. (c) Accuracy comparison for large dataset.

Fig. 12. The accuracy of the decision tree models for the baseline algorithm under various skip ratios.

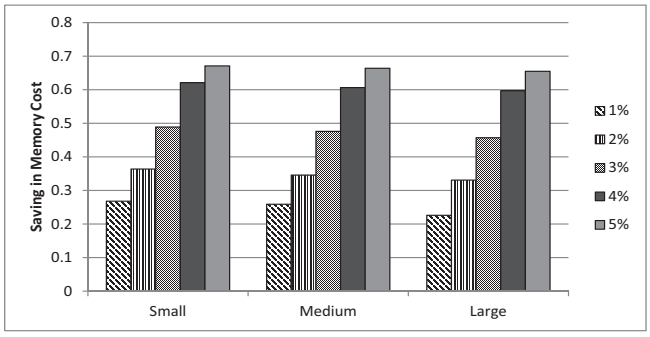


Fig. 10. The memory performance improvement via data access skipping on a 4x8 NoC based multicore system with various datasets.

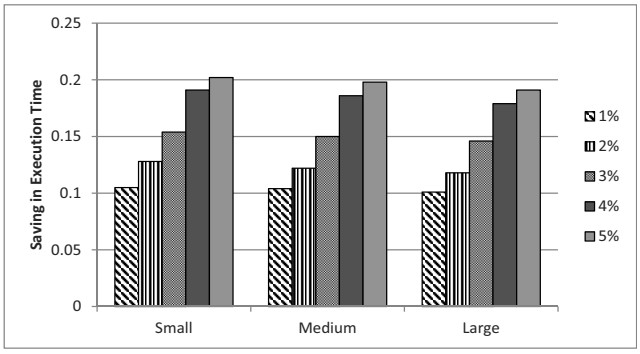


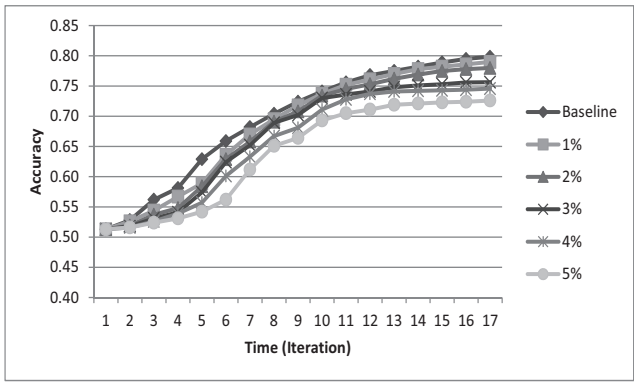
Fig. 11. The improvement in execution time via data access skipping on a 4x8 NoC based multicore system with various datasets.

B. Accuracy Analysis

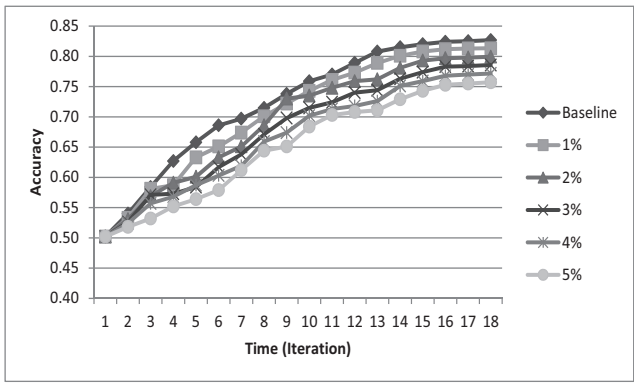
1) Model Accuracy: As we discussed before, the accuracy of the decision tree models is not affected by the inherent random element of DASM drastically. To show this effect, we performed experiments with different skip ratios and skip scenarios similar to Section IV-A. We repeated the experiment 20 times and collected the best and worst accuracy values in addition to the average for every skip ratio. The graphs in Figure 12 indicate that our framework offers a viable selection of optimizations for any target dataset size and accuracy. Medium level of skipping (3%) offers a reasonable balance in the performance/accuracy tradeoff, The memory performance is improved by approximately 47%, while the accuracy is reduced by approximately 5%. These results clearly demonstrate the intrinsic flexibility of decision tree learning algorithm, as the user can set his or her expected accuracy and find the ideal level of skipping for maximum performance.

Due to the iterative nature of our decision tree learning algorithm, interim trees are valid and viable classification models. These models are especially important for decision tree learning as some pruning techniques remove some of the branches from the final model, resulting in a leaf node that is similar to an interim model. Hence, it is important that the accuracy of interim models using DASM do not diverge significantly from the baseline algorithms' results. We present the accuracy of the decision tree models at every level to understand the impact of our approximation over iterations in Figure 13. These values are collected by taking a snapshot of the model at the end of every level and evaluating the test dataset using these partial model snapshots. Our results suggest that the accuracy with the skipping module follows the same trajectory as our baseline model. While variations from the baseline become more prominent as the skip ratio increases, towards the final iterations of the model, the accuracy values stabilize for every case. These results suggest that, even if a pruning technique is applied to the tree, our accuracy predictions from Figure 7 will still be valid for the models constructed using our approximation technique.

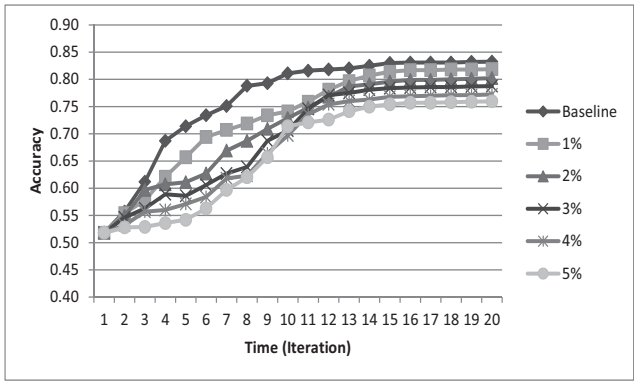
2) Replacement Accuracy: Even though the decision tree learning algorithm compensates for most of the errors in predicting the skipped data access values thanks to its inherent mechanics, it is still beneficial to have a more accurate way of predicting the skipped values than selecting a replacement randomly. Figure 14 plots how accurate the decision tree models use random and heuristic replacement using various skip ratios. We conducted 20 separate experiments for each



(a) Accuracy over time for small dataset.



(b) Accuracy over time for medium dataset.

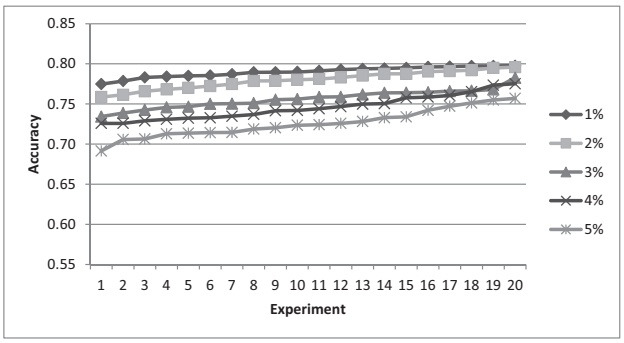


(c) Accuracy over time for large dataset.

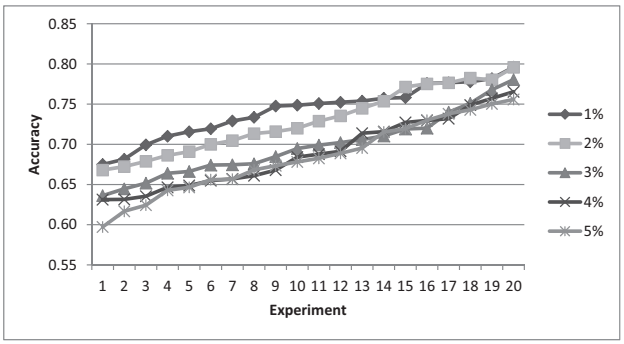
Fig. 13. The accuracy of the interim models created by the decision tree learning algorithm and various skipping policies.

ratio, and sorted them to better show the discrepancy among them. The most accurate experiments were close in accuracy for random and heuristic schemes, but the deviation between experiments is significantly higher in the random case. These figures show that our heuristic not only offers a higher average accuracy; it also provides more consistent results over iterations.

Up to this point we explored the accuracy of overall models with DASM from different angles. These experiments obfuscate most of the inaccuracies stemmed from incorrect replacements. To give a better overall understanding of the



(a) Accuracy of the heuristic selection.



(b) Accuracy of the random selection.

Fig. 14. The accuracy of the models after skipping depends on the replacement policy. While there are a number of sophisticated techniques to alleviate this problem, our lightweight approach stabilizes the output considerably. The baseline accuracy values for this dataset (small) can be found in Figure 12.

optimization, we measured the accuracy of individual approximated values throughout the entire program. Figure 15 plots the accuracy of random and heuristic replacement techniques using various policies (skip ratios). Our experiments show that our heuristic predicts the correct class for the skipped point with 82% accuracy. The random replacement accuracy is at 50% as predicted for a model with two classes. While the improvement of our heuristic over the random replacement is clear, it is important to note that both of the accuracy values are well below the accuracy of the overall models we reported in Figure 14. These experiments support our aforementioned claim on the inaccuracy tolerance of our target algorithm.

V. Related Work

Decision tree learning algorithms have been the subject of extensive research. Rokach and Maimon have explored the creation and evaluation of decision trees, including their splitting criteria, feature selection, and hybridization [36]. Quinlan has proposed the top-down induction of decision trees, which remains one of the most common strategies for creating decision tree models [32]. Hyafil and Rivest have proven that constructing an optimal binary decision tree is an NP-complete problem, paving the way for the development of heuristic

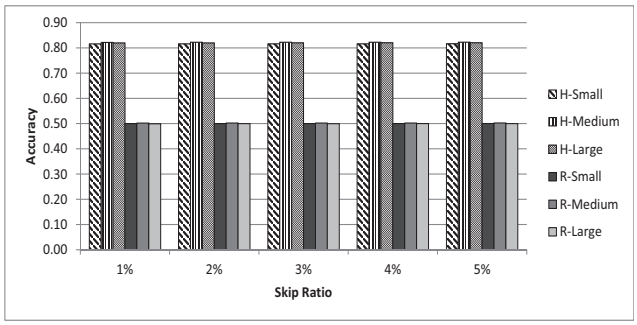


Fig. 15. The accuracy of individual cache miss skips approximations. The results of heuristic and random replacement are denoted by "H" and "R" prefixes, respectively.

schemes [18]. Joshi et al. have proposed a parallel and scalable formulation of the decision tree algorithm using a parallel hash table [19]. A number of variations on the standard algorithm such as boosted trees and rotation forrest have been proposed to adapt decision tree learning to specific purposes [14], [16], [35], [3].

In recent years, approximate computing research has led to a number of interesting optimizations. Grigorian and Reinman have developed a platform-agnostic technique, called Light-Weight Checks that exploits the imprecision tolerance of inverse kinematics [15]. Sui et al. have explored fine-tuning options for approximate computing via a machine learning algorithm [42]. Esmaeilzadeh et al. have proposed a neural network accelerator for transforming certain code regions, as well as architectural support for aproximate computing [12], [11]. The load value approximation technique by Miguel et al. has revealed the viability of such optimizations for a number of different workloads [27]. Research by Sampson et al. has shown that both DRAM cells' lifetimes and performance may be improved through approximate storage [39]. Samadi et al. have developed SAGE, a compiler with approximate computation optimizations for GPUs [38]. Alvarez et al. have focused on power consumption in their work, introducing an instruction memoization technique based on the fuzzy computation paradigm [1]. Yetim et al. have focused on streaming applications, developing a technique to enable error-tolerant communication [43]. Fang et al. have evaluated the error tolerance of various GPGPU applications [13]. Approximate computing has also been used to improve other performance-related metrics such as synchronization overhead [33].

Various prediction-based optimizations have been proposed to solve problems similar to ours. Checkpoint-assisted value prediction has been proposed by Ceze et al. in order to conceal L2 cache misses [7]. Lipasti et al. have introduced value locality, offering a load-value prediction model that exploits this concept [22]. Burtscher has created a compact and high-performing load value predictor entirely at the software level [4]. Value prediction has been shown by Liu and Gaudiot to be useful for reducing communication latency within many-core systems [24]. Martin et al. have explored the effects of erroneous value prediction on multithreaded systems and have discussed how to counter such problems [26]. Ozturk et al. have developed a method for analyzing the source code at run-time for branch prediction [30]. Odaira has explored hardware transactional memory and its implications for thread-level speculation [29].

Our proposed optimization differs from these prior techniques in a number of ways. First, DASM is less intrusive than most of the techniques discussed in this section. With its simple module for redirecting some data accesses, DASM is easy to implement and use in conjunction with other optimizations. This is especially important for problems requiring large datasets and/or a large number of approximations. Such constraints would invalidate most of the approximations employing history tables, buffers, or similar constructs. While prediction-based models are useful for a variety of algorithms, they are unsuitable for our purpose (eliminating costly data accesses). It is important to note that the values of the data points never change during the execution; this means that predicting a given value might increase the cost of accessing a data point unless the prediction algorithm can ensure that it will never (or only in rare circumstances) incur costly data accesses. To the best of our knowledge, a value prediction algorithm that accounts for such a strict restriction does not currently exist. Broadly speaking, our approach is the first one that employs approximate computing paradigm in an architecture-aware fashion.

VI. Concluding Remarks

Targeting decision tree learning applications, this study proposes architecture-aware approximation. Our optimization framework has two key components: a data access skipping mechanism, and a user assist for various levels of aggressiveness in optimization. We evaluate the proposed optimization using various metrics. Our experiments reveal that the proposed framework yields significant improvements in memory performance (up to 76%) and consequently in overall performance (up to 25%), with a limited reduction in accuracy (up to 8%) with respect to the unoptimized case.

Our future work will focus on investigating the interaction between our proposed technique and various other approximation heuristics. Work is also underway in testing our algorithms with other datasets.

References

[1] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 922–927, July 2005.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[4] M. Burtscher, "Improving context-based load value prediction," Ph.D. dissertation, University of Colorado, Department of Computer Science, 2000.

[5] T. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2011, pp. 1–12.

[6] L. Ceriani and P. Verme, "The origins of the gini index: extracts from variabilit e mutabilit (1912) by corrado gini," *The Journal of Economic Inequality*, vol. 10, no. 3, pp. 421–443, 2012.

[7] L. Ceze, K. Strauss, J. Tuck, J. Torrellas, and J. Renau, "Cava: Using checkpoint-assisted value prediction to hide l2 misses," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 3, no. 2, pp. 182–208, Jun. 2006.

[8] M. Chaudhuri, "Pagenuca: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," in *IEEE 15th International Symposium on High Performance Computer Architecture*, Feb 2009, pp. 227–238.

[9] W. J. Dally and B. Towles, "Route packets, not wires: On-chip inteconnection networks," in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 684–689.

[10] W. Ding and M. Kandemir, "Improving last level cache locality by integrating loop and data transformations," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2012, pp. 65–72.

[11] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *SIGPLAN Not.*, vol. 47, no. 4, pp. 301–312, Mar. 2012.

[12] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45, 2012.

[13] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2014, pp. 221–230.

[14] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, Feb. 2002.

[15] B. Grigorian and G. Reinman, "Improving coverage and reliability in approximate computing using application-specific, light-weight checks," *First Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[16] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning : Data mining, inference, and prediction*. Springer Verlag, 2001.

[17] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler, "A nuca substrate for flexible cmp cache sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1028–1040, Aug 2007.

[18] L. Hyafil and R. L. Rivest, "Constructing Optimal Binary Decision Trees is NP-Complete," *Information Processing Letters*, vol. 5, pp. 15–17, 1976.

[19] M. Joshi, G. Karypis, and V. Kumar, "Scalparc: a new scalable and efficient parallel classification algorithm for mining large datasets," in *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, Mar 1998, pp. 573–579.

[20] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *16th International Conference on High-Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[21] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 65–76.

[22] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VII. New York, NY, USA: ACM, 1996, pp. 138–147.

[23] J. Liu, Y. Zhang, W. Ding, and M. Kandemir, "On-chip cache hierarchy-aware tile scheduling for multicore machines," in *9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, April 2011, pp. 161–170.

[24] S. Liu and J.-L. Gaudiot, "Potential impact of value prediction on communication in many-core architectures," *Computers, IEEE Transactions on*, vol. 58, no. 6, pp. 759–769, June 2009.

[25] K. Malkowski, G. Link, P. Raghavan, and M. Irwin, "Load miss prediction - exploiting power performance trade-offs," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, March 2007, pp. 1–8.

[26] M. M. K. Martin, D. J. Sorin, H. W. Cain, M. D. Hill, and M. H. Lipasti, "Correctly implementing value prediction in microprocessors that support multithreading or multiprocessing," in *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, ser. MICRO 34. Washington, DC, USA: IEEE Computer Society, 2001, pp. 328–337.

[27] J. S. Miguel, M. Badr, and N. E. Jerger, "Load value approximation," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 127–139.

[28] A. K. Mishra, "Design and analysis of heterogeneous networks for chip-multiprocessors," Ph.D. dissertation, University of Colorado, Department of Computer Science, University Park, PA, USA, 2011, aAI3483797.

[29] R. Odaira and T. Nakaike, "Thread-level speculation on off-the-shelf hardware transactional memory," in *IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2014, pp. 212–221.

[30] C. Ozturk, I. Karsli, and R. Sendag, "Automatic source code analysis of branch mispredictions," in *IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2014, pp. 82–83.

[31] X. Pan and B. Jonsson, "A modeling framework for reuse distance-based estimation of cache performance," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.

[32] J. R. Quinlan, "Induction of decision trees," in *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich, Eds. Morgan Kaufmann, 1990, originally published in *Machine Learning* 1:81–106, 1986.

[33] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener, "Programming with relaxed synchronization," in *Proceedings of the 2012 ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability*, ser. RACES '12. New York, NY, USA: ACM, 2012, pp. 41–50.

[34] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2. ACM, 2000, pp. 128–138.

[35] J. Rodriguez, L. Kuncheva, and C. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619–1630, Oct 2006.

[36] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*. World Scientific Pub Co Inc, 2008.

[37] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 35–50.

[38] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 13–24.

[39] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 25–36.

[40] J. C. Shafer, R. Agrawal, and M. Mehta, "Sprint: A scalable parallel classifier for data mining," in *in Proceedings of the 22th International Conference on Very Large Data Bases*, ser. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 544–555.

[41] A. Sharifi, E. Kultursay, M. Kandemir, and C. Das, "Addressing end-to-end memory access latency in noc-based multicores," in *45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2012, pp. 294–304.

[42] X. Sui, A. Lenharth, D. Fussell, and K. Pingali, "Tuning variable-fidelity approximate programs," *Second Workshop on Approximate Computing Across the System Stack (WACAS)*, 2015.

[43] Y. Yetim, M. Martonosi, and S. Malik, "Parallel streaming computation on error-prone processors," *First Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[44] J. Zambreno, B. Ozisikyilmaz, G. Memik, and A. Choudhary, "Performance characterization of data mining applications using minebench," in *9th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2006.