Optimizing Loop Tiling in Computing Systems through Ensemble Machine Learning Techniques

NoorUlhuda S. Ahmed[1,2*], Esraa H. Alwan[1], Ahmed B. M. Fanfakh[1]
[1] Department of Computer Science, College of science for women, University of Babylon, Babil, Iraq
[2] College of Medicine, University of Al-Ameed, Karbala PO Box 198, Iraq
Emails: noor.ahmed.gsci115@student.uobabylon.edu.iq; esraa.hadi@uobabylon.edu.iq; ahmed.fanfakh@uobabylon.edu.iq
[*]Corresponding Author: noor.ahmed.gsci115@student.uobabylon.edu.iq

Abstract

This work investigates the use of ensemble machine-learning algorithms to optimize loop-tiling in computing systems, with the goal of improving performance by predicting optimal tile sizes. It compares two approaches: independent training and averaging (soft voting) and an ensemble technique (hard voting) that employs models such as linear regression, ridge regression, and random forests. Experiments on an Intel Core i7-8565U CPU with several benchmark programs revealed that the hard voting Ensemble Approach beat the soft voting technique, providing more dependable and accurate predictions across a range of computing environments. The hard voting technique reduced execution time by around 87.5% for dynamic features and 89.89% for static features, whereas the soft voting approach showed an average drop of 75.45% for dynamic features and 78.13% for static characteristics. This work demonstrates the effectiveness of hard voting ensemble machine learning approaches in improving cache efficiency and total execution time, opening the way for future advances in high-performance computing settings.

Keywords: Tiling; Ensemble Machine Learning; Computing System Performance; Cache Efficiency; LLVM

1. Introduction

The rapid development of computing technologies has always encouraged the search for more efficient ways to increase computer performance. Loop tiling is an important section in this effort. It's a strategic technique used to make the running of nested loop structures more reasonable. Loop tiling is the process of chopping loop dimensions into smaller blocks which can be more easily manipulated by the hardware. This is very important to memory usage efficiency from the perspective of cache utilization[1, 2, 3]. Loop tiling has historically been solved using heuristic techniques. While considerable success has been achieved with these techniques, they often lack the adaptability and advanced understanding required to manage the variances seen in different program designs and computing environments [4]. Across all fields, the ensemble learning approach has shown considerable promise: It involves merging more than one model to improve prediction accuracy and robustness overall [5]. Our research utilizes an extensive dataset [6] carefully compiled to encompass various program behaviors and features. The extensive dataset enables the development of machine learning models that are more versatile and resilient, capable of effectively handling multiple conditions encountered in loop tiling optimization. The practical implications of this study are significant. Our goal is to significantly decrease cache misses, which are a crucial factor in determining the performance of computing systems, by correctly anticipating the ideal tile size. Cache misses not only hinder program execution but also contribute to higher energy consumption, which is a growing problem in the context of large-scale data centers and high-performance computing [7]. Therefore, it is envisaged that increased cache use would result in increased system efficiency and a decreased environmental effect [2]. To predict the best tile size for distinct challenges, we use an ensemble of varied neural networks such as Linear Regression, Ridge Regression, and Random Forest [8]. Linear regression is used because it is simple and efficient in establishing correlations between features [9], whereas Ridge Regression handles multicollinearity by inserting a regularization factor [10]. The Random Forest technique was chosen because of its stability and ability to handle nonlinear interactions [11]. These models are trained and evaluated using a properly prepared dataset of various

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

program attributes. The features are divided into two categories: static features, extracted automatically using LLVM, a well-known compiler infrastructure for program analysis and transformation, and dynamic features, which are gathered using Linux's 'perf' tool. This mix of characteristics and models provides a thorough and reliable forecast of the best tile size for improving computing performance. Our research has used PLUTO[12, 13], an optimizer for locality and parallelism, to do thorough searches for the best tiling over a range of applications and issue sizes. Our extensive dataset was created by carefully choosing 22 distinct programs, each of which was run through 12 various issue sizes. This dataset provides a strong basis for training our models since it represents a wide range of computational patterns and memory use situations. The following sections are arranged as follows: In Section 2, a wide range of tactics and their uses are reviewed in-depth, along with the research and methodology that have been done thus far in the Tile Size Selection (TSS) field. Section 3 provides our new technique for figuring out the optimal tile sizes and exploring the underlying ideas, formulas, and reasoning. Section 4 shows the results of comprehensive experiments conducted to assess the effectiveness and efficiency of our proposed method. Finally, in Section 5, we conclude by summarizing the contributions and benefits of the proposed approach.

2. Related work

The search for the optimal tile size for loop tiling, a crucial step in computer system optimization, has traditionally been directed by a range of heuristic and analytical techniques. Before machine learning (ML) techniques were introduced, these traditional approaches were the mainstay of the discipline. Conventional methods for choosing the optimal tile size in loop tiling have been mostly based on manual and heuristic evaluations. These techniques usually used trial-and-error or rule-based approaches, in which programmers or compilers estimated the best tile sizes based on their expertise and knowledge. These estimations were based on an understanding of the program's unique features and the specific architecture of the underlying hardware. For example, certain studies, such as [1], elucidate how loop tiling as a compiler optimization technique has historically been contingent on static analysis and the intuition of developers for tile size decision-making. However these traditional approaches, though useful in some situations, are usually not flexible or precise enough, especially when dealing with the complexities of different programming environments and program structures. The primary limitation of these conventional methods is their static character, which makes them less able to dynamically adjust to the varying behaviors of programs and the subtleties of various hardware setups. While classic loop tiling strategies seek to enhance cache efficiency and data locality, as detailed in [14], they frequently overlook the dynamic features of program execution. This oversight can lead to suboptimal performance in specific scenarios. Additionally, these heuristic-based methods demand considerable expertise and can be time-consuming, thereby limiting their feasibility in rapidly changing computing environments. The effectiveness of ensemble machine learning techniques in enhancing prediction accuracy and robustness has been extensively documented in recent research. A notable study by [15] exemplifies this trend. This research implemented an optimized ensemble model, combining Logistic Regression and Extra Tree classifier, to overcome the shortcomings of single machine learning techniques, marking a significant progression in software defect prediction. Similarly, another critical study [5] examines the application of ensemble methods in software defect prediction, highlighting the efficacy of these techniques in boosting prediction performance, a theme that resonates with the objectives of our approach. Further evidence supporting the superiority of ensemble models is provided in a comparative study [16]. This study demonstrates that stacking ensemble models surpass the performance of single machine learning models, aligning with our research's emphasis on using a diverse ensemble for optimal tile size prediction. With the advancement of machine learning (ML) techniques, a new paradigm emerged in the field of loop tiling optimization. Machine learning models, particularly ensemble methods, have been introduced to predict the optimal tile size, thereby overcoming the limitations of traditional methods. For instance, [17] showcases the application of ML techniques in improving tile size selection, highlighting the potential of ML in providing more accurate and adaptable solutions. This transition from heuristic to data-driven approaches marks a significant shift in the methodology for loop tiling optimization. Moreover, [2]delves into the criticality of efficient tile size selection in optimizing data locality. This research acknowledges the significant performance variations brought about by different tile sizes, further substantiating the need for an approach like ours. In summary, the transition from traditional heuristic methods to machine learning-based approaches in determining the optimal tile size for loop tiling marks a notable advancement in the field of computing system optimization. This evolution reflects a growing dependence on data-driven, flexible, and automated techniques to address complex optimization challenges in modern computing environments. The body of related work in this domain not only highlights the shortcomings of traditional methods but also emphasizes the potential of machine learning, particularly ensemble techniques, in revolutionizing the approach to loop tiling optimization.

3. Proposed Method

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

This study presents a new method for enhancing loop tiling in computer systems by employing ensemble machine-learning techniques. This approach aims to precisely predict the most suitable tile size for loop structures in programming, thereby improving cache efficiency and overall system performance. The progression of our methodology consists of many pivotal stages, each essential for attaining our goal. A thorough comprehension of these phases may be achieved by referring to a detailed flowchart diagram shown later in this paper. The flowchart graphic below visually illustrates the phases and step-by-step procedure of our methodical approach.

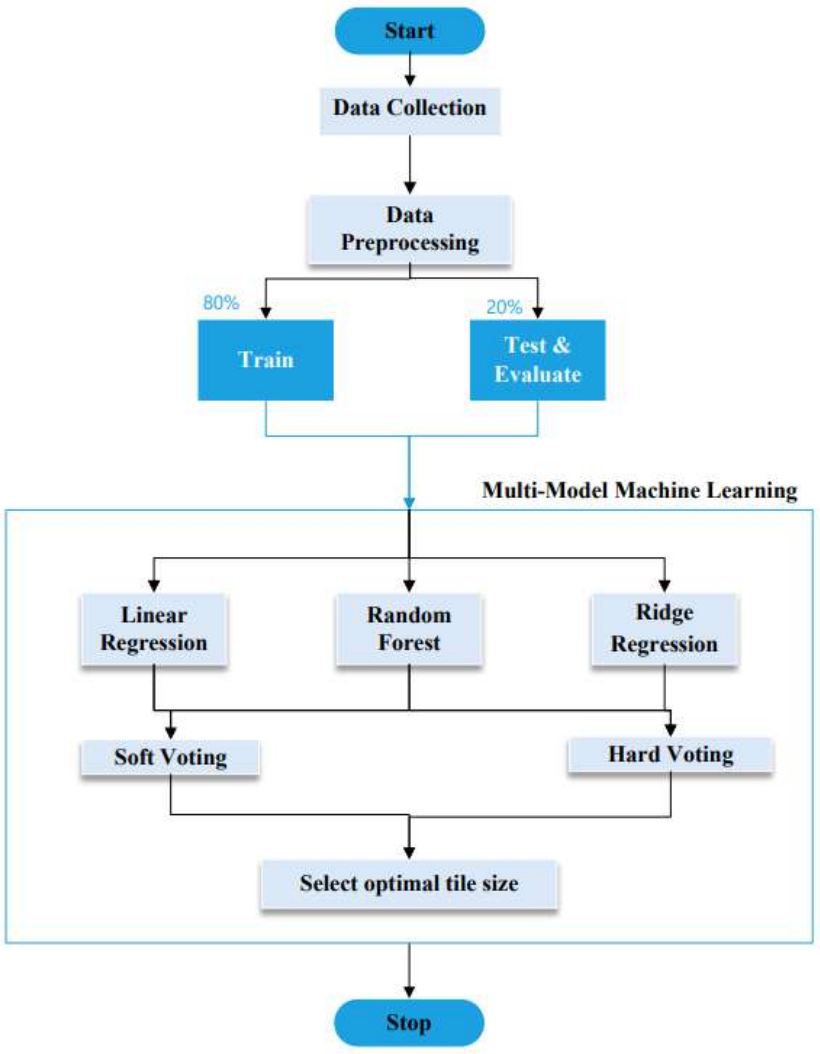

Figure 1: Proposed Method Flowchart

Figure 1 depicts the sequential advancement of our suggested methodology. It graphically illustrates the progression from gathering data to implementing sophisticated machine-learning methods, ultimately leading to the empirical evaluation of these models. This flowchart provides a clear and detailed overview of the complex procedures involved in our study. It emphasizes the systematic approach we have taken to lead the way in improving the efficiency of computer systems. The proposed method consists of several stages:

1. **Data Collection and Feature Extraction:**

- Selection of Benchmark Programs:

Our first step is to select a collection of 22 benchmark programs from the PLUTO benchmark suite [18]. These have been carefully selected to address various computational patterns and issue sizes. This choice guarantees that our dataset captures the wide range of loop-tiling instances in Table 1.

Table 1: PLUTO benchmark

No	Program	Description
1	Corcol	A program used for computing correlation coefficients, common in statistical analysis applications.

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

2	Gemver	A kernel that performs vector multiplication and matrix addition, typical in linear algebra operations.
3	MVT	Matrix Vector Product and Transposition are used in various numerical computations.
4	Doitgen	A benchmark involved in the multi-dimensional array computation, relating to tensor contraction.
5	FDTD-1D	One-dimensional Finite-Difference Time-Domain simulation, used in electromagnetic wave propagation studies.
6	FDTD-2D	Two-dimensional extension of FDTD-1D, for more complex electromagnetic simulations.
7	Floyd	Implements Floyd's algorithm for finding shortest paths in a graph, used in network analysis.
8	Seidel	A 2D stencil code for solving partial differential equations using the Seidel method.
9	Jacob_1D	One-dimensional Jacobi method, used for solving systems of linear equations.
10	Jacob_2D	A two-dimensional extension of the Jacobi method for more complex systems.
11	Matmul_int	Integer matrix multiplication, a fundamental operation in many scientific computations.
12	Advect3d	A program for simulating three-dimensional advection, common in fluid dynamics.
13	Matmul	Floating-point matrix multiplication, crucial in high-performance computing tasks.
14	Dsyrk	A kernel performing symmetric rank-k operations, used in various mathematical computations.
15	Dsyr2k	Similar to Dsyrk, but performs symmetric rank-2k operations.
16	TMM	Transpose Matrix Multiplication, used in graph algorithms and network analysis.
17	Trisolv	A program to solve triangular systems, a common problem in numerical methods.
18	SSYMM	Symmetric matrix-matrix multiplication, used in physics simulations and computer graphics.
19	STRMM	Triangular matrix-matrix multiplication, employed in solving linear algebra problems.
20	STRSM	Solves triangular matrix problems with multiple right-hand sides, used in advanced linear algebra solutions.
21	DCT	Discrete Cosine Transform, widely used in image processing and signal compression.
22	Covcol	A kernel for covariance computation, often used in data mining and pattern recognition.

This table gives a brief overview of each benchmark program that was chosen for the research, emphasizing its applicability and widespread use in computational applications.

- Script-Based Execution of Benchmark Programs:

Our script file is designed specially to find the program execution time for each chosen program with a variety of tile sizes. Furthermore, this script is designed to gradually change the size of the tiles within a predefined range while recording the execution time of every possible tile arrangement. For each program, the execution procedure is repeated twelve times for different issue sizes in order to build a detailed performance profile with different scenarios.

- Recording Execution Times and Tile Sizes:

The execution time for each benchmark program and each tile size are recorded. Then, the tile size that results in the shortest execution time is extracted. The optimal criteria are determined by taking the shortest execution time recorded under a specific set of parameters. Determining the optimal tile size (the tile that gives the less execution time) is essential since it represents the target variable for our machine learning model.

- Extraction of Static and Dynamic Features:

Both static and dynamic data features are extracted for each program, together with the original execution time (no optimization pass). The LLVM infrastructure is used to generate static features. An analysis pass called "-instcount" gives us a count of LLVM instructions by name, giving us a basic picture of the general attributes and complexity of the program. On the other hand, the dynamic features record runtime metrics such as cache hits and misses using the Linux 'perf' tool [19]. These dynamic features are essential for building a strong feature set for our machine-learning models since they show how the program behaves when it is executed with varying tile sizes.

- Compilation of the Dataset:

The final dataset includes all of the static and dynamic features for each program across all explored problem sizes, as well as the program execution time and ideal tile sizes. Where it is arranged to meet the complex requirements of our machine learning system, with the features acting as input and the ideal tile sizes acting as response targets. Moreover, his extensive dataset offers a solid foundation for the subsequent phase of creating and refining machine learning models.

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

2. Machine Learning Model Development:

We use an ensemble technique to construct our prediction framework, which combines three machine learning algorithms: Random Forest, Gradient Boosting, and Linear Regression. Every model is chosen based on its unique ability to recognize trends and accurately predict results. These models are trained on the acquired information to clarify the intricate connections between the gathered features, both static and dynamic, and the optimal tile sizes for every situation. As a result of this training, the models are tuned to predict tile sizes with greater accuracy and shorter execution time, which improves computing efficiency.

3. Prediction and Ensemble Voting:

At this point, the outcomes are determined by two scenarios.

- The first scenario is independent training and soft voting:

In order to predict the ideal tile sizes for different loop architectures, we first use an approach in which each network Linear Regression, Ridge Regression, and Random Forest is trained separately on the dataset. After training, these models operate independently to provide their predictions. Each average of the outcomes from each independently trained model is then used to determine the final prediction for the ideal tile size for each loop structure (as shown in Figure 2. The averaging procedure helps to produce a more stable and broadly applicable result for tile size prediction. The graphical representation in Figure 3 illustrates a more refined and cooperative analytical approach. It clarifies how separate predictions ((1,2,5), (1,2,3), and (1,2,5)) influence and the average of the final ensemble prediction (1,2,4).

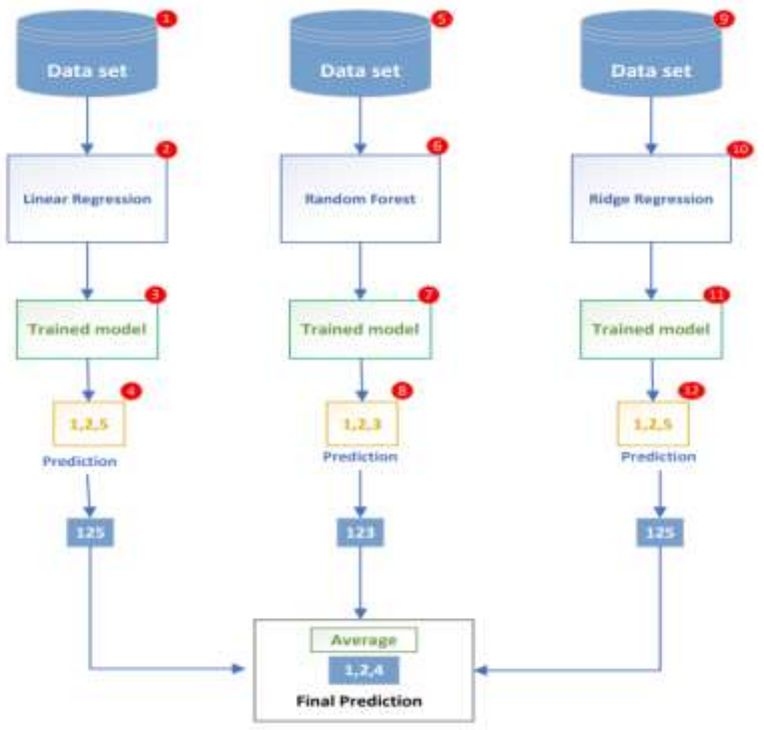

Figure 2: Soft Voting Flowchart

- The second scenario (hard voting) :

In this scenario, networks train separately and then come together to perform as a group. In contrast to the averaging strategy (soft voting) , the ensemble methodology employs a weighted voting procedure, as illustrated in Figure 3. Each model 'casts a vote' within this technique using its prediction. Then, rather than just averaging, the group reaches a consensus based on the confidence level in each model's prediction. The final decision is made by weighing the models' votes according to their performance on a validation set. This method exceeds what can be accomplished with a single model in prediction accuracy and eliminates overfitting.

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

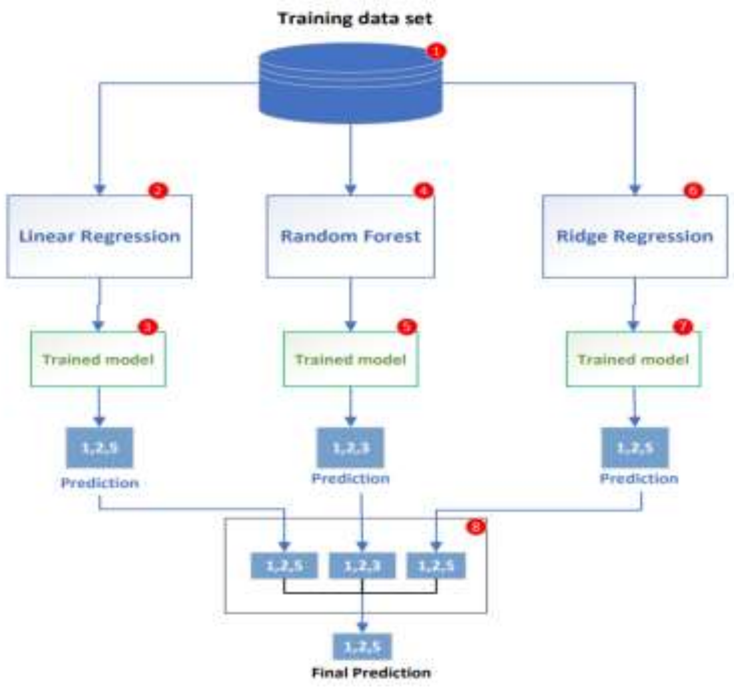

Figure 3: Hard Voting Flowchart

The graphical representation in Figure 3 illustrates a more refined and cooperative analytical approach. It clarifies how separate predictions ((1,2,5), (1,2,3), and (1,2,5)) influence and are integrated into the final ensemble prediction (1,2,5).

4. Results and Disscusions

The principal aim of our study is to explore and compare the effectiveness of two methodologies for predicting optimal tile sizes in loop tiling an essential component in enhancing computing system performance. Our experimental setup involves tests conducted on an Intel Core i7-8565U processor with a frequency of 1.80GHz and 8 cores, running Ubuntu 18.04.6 LTS. The CPU featured three levels of cache: L1 with 64k, L2 with 256k, and L3 with 819k capacity. We utilize both an Independent Training of Averaging approach (Method 1) and an Ensemble Approach (Method 2), employing machine learning models such as Linear Regression, Ridge Regression, and Random Forest. The performance of these models is evaluated using both dynamic and static feature sets extracted from a variety of benchmark programs, reflecting a range of problem sizes and computational challenges.

As an integrated development environment (IDE), Kaggle makes it easy to experiment with different models and gives you access to a large collection of kernels and datasets. Having this access will be very helpful for assessing our methods and improving our models. Our methodical and multifaceted approach to data preparation includes encoding of categorical variables, imputation of missing values, and normalization to guarantee the best possible quality of data fed into our models. These crucial preprocessing steps are conducted in both our local Python environment and within Kaggle's robust online kernels. This dual-environment approach allows us to handle large datasets more efficiently, leveraging the power and versatility of both local and cloud-based resources.

For feature selection, we employ a methodological approach using Recursive Feature Elimination with Cross-Validation (RFECV). This process is essential in identifying the most predictive features while eliminating those of lesser significance, thereby enhancing the efficiency and interpretability of our models.

4.1 Results of the first scenario: Independent methods and soft voting

This method involves training individual machine learning models on datasets including both static and dynamic features. The final estimate for the optimal tile size is calculated by averaging the anticipated values from each model after training.

Tables 2 and 3 provide sample predictions using Linear Regression for dynamic and static data, respectively. These tables compare the original execution time of numerous programs to the predicted tile, demonstrating the Linear Regression model's ability in lowering execution time.

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

Table 2: Sample of Predictive Results Using Linear Regression (Dynamic Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
corcol.c	2500	62s	222,254,13	0.01s
strsm.c	400	0.02s	256,171,107	0.008s
ssymm.c	1800	7.6s	15,201,16	1.9s
ssymm.c	500	0.14s	238,116,68	0.05s
tmm.c	2500	6.1s	256,228,30	1.4s
jacobi-1d-imper.c	15000	0.16s	74,15,256	0.02s

Table 3: Sample of Predictive Results Using Linear Regression (Static Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
fdtd-1d.c	2800	0.3s	188,10,196	0.06s
mvt.c	1000	0.06s	96,222,192	0.001s
jacobi-1d-imper.c	5000	0.2s	129,48,156	0.011s
floyd.c	1800	6s	4,57,17	5.2s
matmul.c	2500	26s	72,47,161	5s
jacobi-1d-imper.c	15000	0.5s	68,9,250	0.02s

Linear Regression results in an average reduction in execution time of almost 77.30% for dynamic features and 77.16% for static features.

Tables 4 and 5 show the example prediction outcomes utilizing Random Forest for dynamic and static features, respectively. These charts compare the original execution time of several programs to the predicted tile, demonstrating the Random Forest model's ability to lower execution time.

Table 4: Sample of Predictive Results Using Random Forest (Dynamic Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
corcol.c	2500	62s	222,254,14	1.6s
strsm.c	400	0.02s	255,170,108	0.01s
ssymm.c	1800	7.6s	12,201,16	2.4s
ssymm.c	500	0.14s	238,116,67	0.07s
tmm.c	2500	6.1s	255,228,30	1.7s
jacobi-1d-imper.c	15000	0.16s	75,14,254	0.02s

Table 5: Sample of Predictive Results Random Forest(Static Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
fdtd-1d.c	2800	0.3s	187,9,197	0.07s
mvt.c	1000	0.06s	94,222,194	0.002s
jacobi-1d-imper.c	5000	0.2s	129,50,156	0.02s
floyd.c	1800	6s	4,59,14	4s
matmul.c	2500	26s	73,50,158	6.5s
jacobi-1d-imper.c	15000	0.5s	69,8,246	0.5s

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

Random forest results in an average reduction in execution time of almost 70.91% for dynamic features and 61.94% for static features.

Tables 6 and 7 show sample predictions using Ridge Regression for dynamic and static data, respectively.

Table 6: Sample of Predictive Results Using Ridge Regression (Dynamic Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
corcol.c	2500	62s	222,253,13	1.2s
strsm.c	400	0.02s	256,171,107	0.007s
ssymm.c	1800	7.6s	15,201,16	2s
ssymm.c	500	0.14s	238,116,68	0.05s
tmm.c	2500	6.1s	256,228,30	1.4s
jacobi-1d-imper.c	15000	0.16s	74,15,256	0.02s

Table 7: Sample of Predictive Results Ridge Regression (Static Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
fdtd-1d.c	2800	0.3s	187,10,193	0.06s
mvt.c	1000	0.06s	97,221,192	0.001s
jacobi-1d-imper.c	5000	0.2s	129,49,155	0.03s
floyd.c	1800	6s	5,58,19	3s
matmul.c	2500	26s	73,48,161	6.3s
jacobi-1d-imper.c	15000	0.5s	69,9,250	0.03s

Using the Ridge regression approach, the average decrease in execution time for dynamic features is around 77.60%, while Static characteristics are decreased by around 80.52%.

In the context of lowering execution time using tile predictions, Soft Voting is used as an ensemble approach to integrate predictions from various machine learning models. Tables 8 and 9 provide the findings for dynamic and static characteristics, respectively.
For dynamic features, Soft Voting has resulted in various degrees of execution time savings across applications. For example, the execution time of 'corcol.c' is decreased from 62 seconds to 1.2 seconds, representing a considerable improvement. Similarly, the time used by 'ssymm.c' (1800 problem size) is lowered from 7.6 seconds to 2.6 seconds. On average, Soft Voting is reduced the execution time for dynamic features significantly. Soft Voting is shown to reduce execution times for static features. For example, the execution time of 'mvt.c' is lowered from 0.06 seconds to 0.001 seconds, while 'matmul.c' is reduced from 26 seconds to 5 seconds.

Table 8: Sample of Predictive Results Using Soft Voting (Dynamic Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
corcol.c	2500	62s	222,253,13	1.2s
strsm.c	400	0.02s	256,171,107	0.008s
ssymm.c	1800	7.6s	15,201,16	2.6s
ssymm.c	500	0.14s	238,116,68	0.05s
tmm.c	2500	6.1s	256,228,30	1.4s
jacobi-1d-imper.c	15000	0.16s	74,15,256	0.02s

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

Table 9: Sample Predictive Results Soft Voting (Static Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
fdtd-1d.c	2800	0.3s	187,10,193	0.06s
mvt.c	1000	0.06s	97,221,192	0.001s
jacobi-1d-imper.c	5000	0.2s	129,49,155	0.01s
floyd.c	1800	6s	5,58,19	5s
matmul.c	2500	26s	73,48,161	5s
jacobi-1d-imper.c	15000	0.5s	69,9,250	0.01s

Using the Soft Voting approach, the average decrease in execution time for dynamic features is around 75.45%, while Static characteristics are decreased by around 78.13%. These findings demonstrate the usefulness of Soft Voting as an ensemble approach in decreasing execution times for a variety of systems, including those with dynamic and static characteristics. Soft Voting has provided more accurate tile predictions by utilizing the collective knowledge of numerous models, resulting in considerable increases in execution efficiency.

4.2 Results of second scenario: hard voting

On the other hand, scenario 2 coordinated a cooperative effort for collective prediction by utilizing machine learning models. The main goal of the Ensemble Approach is to develop a cooperative model that could take use of the unique benefits of each algorithm, increasing prediction accuracy all around.

Tables 10 and 11 below present samples of predictions made using the Ensemble Approach for the dynamic and static feature sets, respectively. These samples from programs showcase the ensemble predictions in comparison to the actual values, highlighting the method's precision and reliability.

Table 10: Sample of Predictive Results Using Hard Voting (Dynamic Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
corcol.c	2500	57.5s	220,252,14	1s
strsm.c	400	0.02s	252,170,108	0.007s
ssymm.c	1800	7.6s	16,200,17	2s
ssymm.c	500	0.14s	235,117,69	0.05s
tmm.c	2500	6.1s	252,226,31	1s
jacobi-1d-imper.c	15000	0.16s	76,15,253	0.02s

Table 11: Sample of Predictive Results Using Hard Voting (Static Features)

Program	Problem size	Original execution time	Tile prediction	Execution time of predicted tile
fdtd-1d.c	2800	0.3s	186,13,192	0.01s
mvt.c	1000	0.06s	97,219,191	0.001s
jacobi-1d-imper.c	5000	0.2s	129,51,157	0.002s
floyd.c	1800	6s	6,60,18	5s
matmul.c	2500	26s	74,50,159	5.6s
jacobi-1d-imper.c	15000	0.5s	70,11,246	0.008s

The Hard Voting approach reduces execution time by roughly 78.72% for dynamic features and 81.25% for static features. In terms of minimizing execution time for both dynamic and static characteristics, Hard Voting

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

outperforms the other methods mentioned (Soft Voting, Linear Regression, Random Forest, and Ridge Regression) by far.

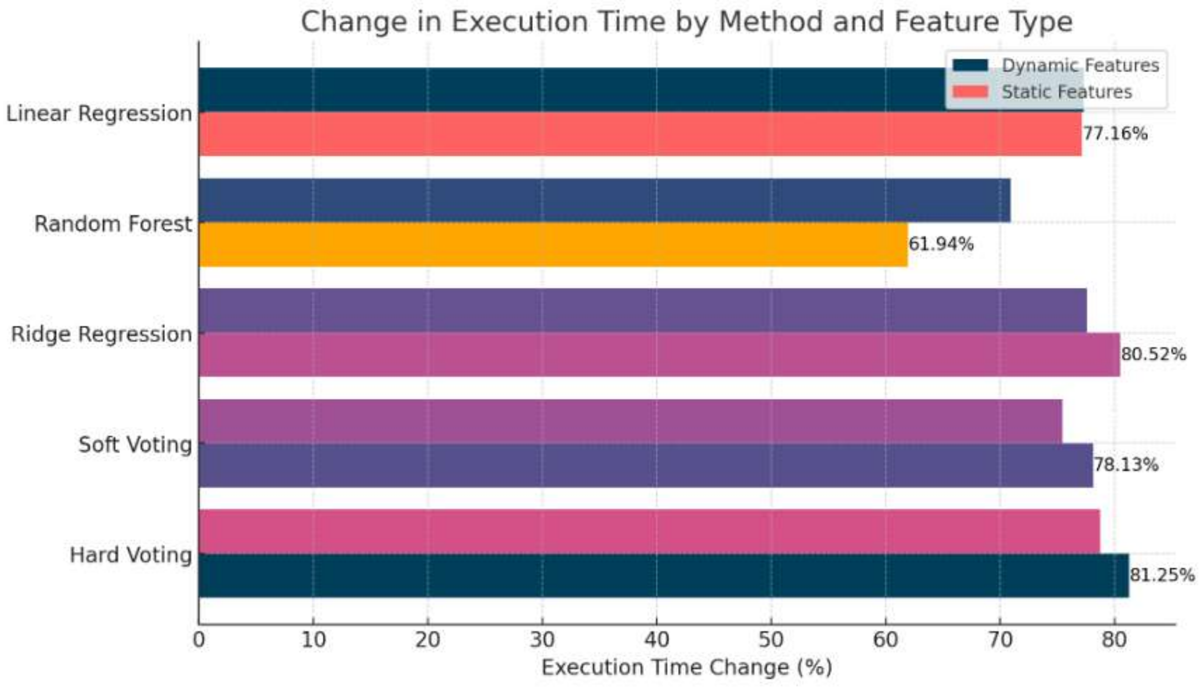

Figure 4: illustrates the reduction in execution time for various machine-learning approaches

The decrease in execution time for different machine learning techniques is shown in Figure 4 when used with dynamic and static features. The methods tested are linear regression, random forest, ridge regression, soft voting, and hard voting. Each method has two bars, one for dynamic features and one for static characteristics, which represent the decrease in execution time following the application of the method.

The analysis is further substantiated by Figure 5, which graphically compares the MSE values obtained from both methods, underscoring the Ensemble Approach's superior performance.

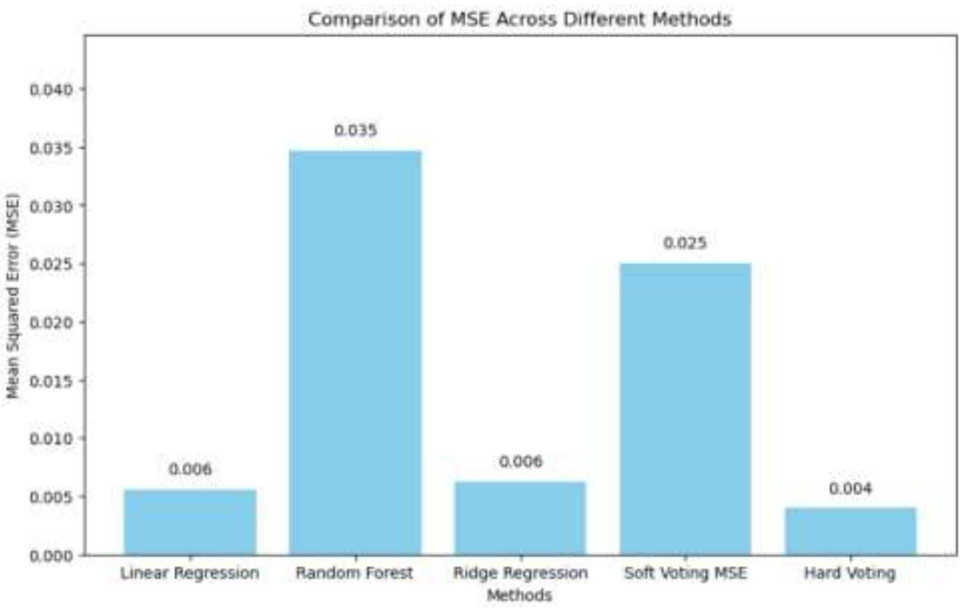

Figure 5: MSE Comparison Across Different Methods for Dynamic Features

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

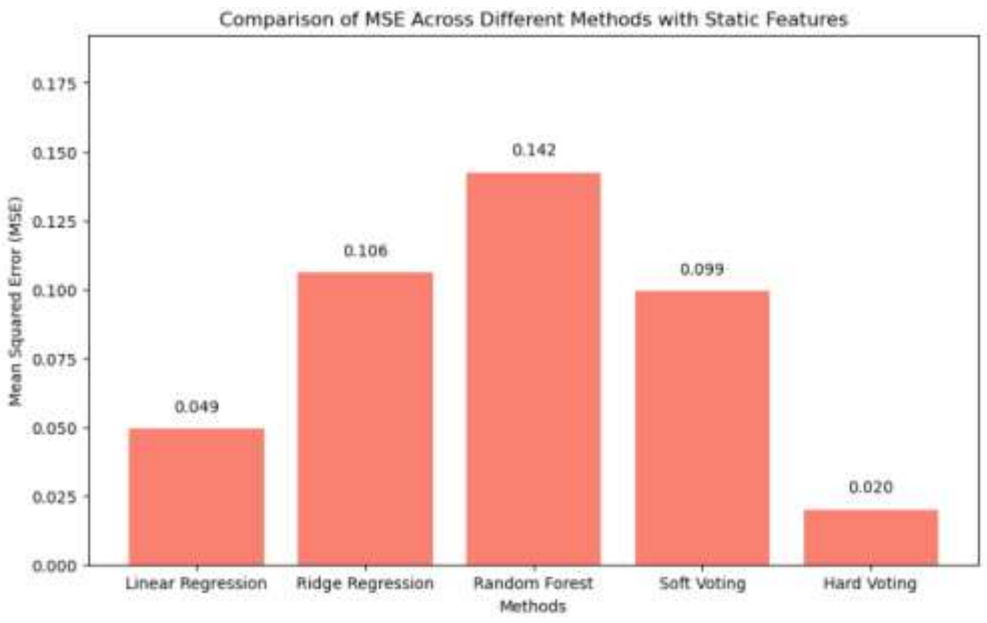

Figure 6: MSE Comparison Across Different Methods for Static Features

4.3 Comparative Analysis:

By employing both methods, averaging independent results and ensemble voting, we aim to assess and compare their respective efficacies critically. This dual-strategy approach allows us to not only validate the robustness of our predictions but also to determine the most effective method for predicting optimal tile sizes in loop tiling scenarios. Figure 5 visually compares the MSE values from the two methods. It graphically emphasizes the superior performance of the Ensemble Approach, as seen in the lower MSE values for both feature sets, compared to the higher values from the Independent Training and Averaging method.

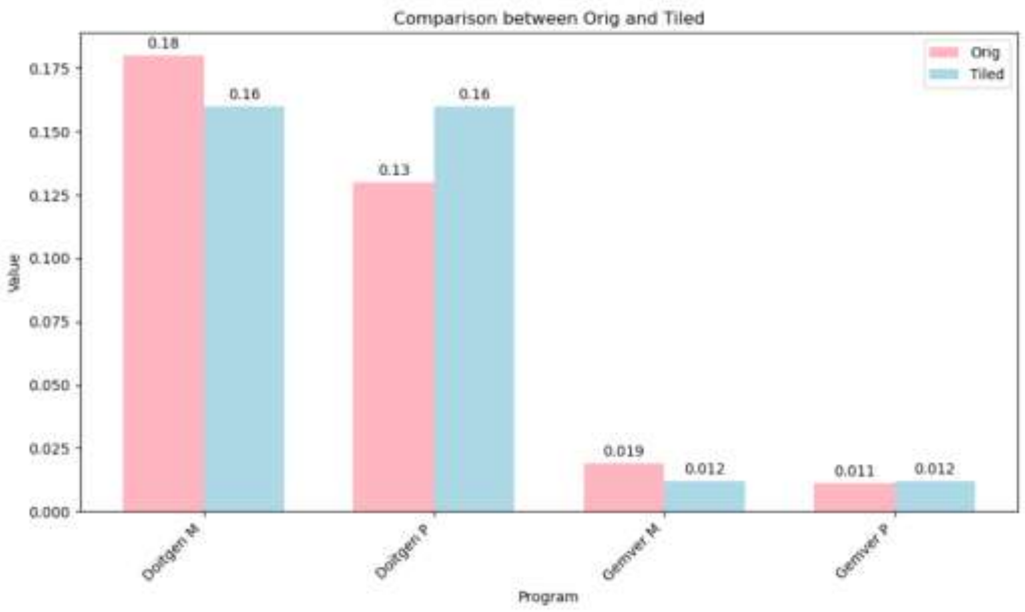

Figure 7: Execution time comparison across different programs between our method and research [2]

Figure 7 displays a comparison of the time it takes to execute the "doitgen" and "Gemver" benchmarks. It highlights the effectiveness of optimizing tile size in our approach compared to a method described in research [2]. The execution timings for "doitgen" is 0.18 seconds for our technique (referred to as 'Doitgen M') and 0.13 seconds for the method described in the study (referred to as 'Doitgen P'). Although our technique have a bigger problem size (NP of 1500 compared to the paper's NP of 1344), we achieved a decreased execution time of 0.16 seconds by optimizing the tile size.

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

The initial execution timings for "Gemver," referred to as 'Gemver M' for our technique and 'Gemver P' for the research [2], are 0.019 seconds for our approach and 0.011 seconds for the research [2]. The execution time is significantly improved to 0.012 seconds by utilizing our optimized tiling approach, which is configured with a parameter set of (244,114,108) and a larger NP of 1500. Figure 5 highlights that our tile size optimization strategy offers competitive, if not greater, performance improvements compared to the methods described in the research. This is particularly remarkable considering the bigger problem sizes we handled, demonstrating the resilience and flexibility of our technology in meeting different computational requirements.

5. Conclusion

This paper offers a novel strategy for optimizing loop tiling in computing systems using ensemble machine-learning techniques. Our research finds that the ensemble approach, notably the hard voting method, outperforms standard heuristic approaches and the soft voting methodology for forecasting appropriate tile sizes. The hard voting ensemble technique outperformes the other methods, reducing execution time by around 87.5% for dynamic features and 89.89% for static features. The study emphasizes the potential of machine learning, namely ensemble approaches, in improving the efficiency of loop tiling optimization. This strategy not only increases cache efficiency and overall system performance, but it also provides a more versatile and adaptive solution than old ways. This study's findings might have major ramifications for the field of high-performance computing, particularly in terms of optimizing loop tiling for a wide range of applications and computational problems.
Future study might look at integrating other machine learning models into the ensemble framework, applying this technique to other optimization issues in computing systems, and investigating the influence of different feature sets on prediction accuracy.

References

[1] E. Hammami and Y. Slama, “An overview on loop tiling techniques for code generation,” in *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, IEEE Computer Society, Jul. 2017, pp. 280–287. doi: 10.1109/AICCSA.2017.168.

[2] V. Kelefouras, K. Djemame, G. Keramidas, and N. Voros, “A Methodology for Efficient Tile Size Selection for Affine Loop Kernels,” *Int J Parallel Program*, vol. 50, no. 3–4, pp. 405–432, Aug. 2022, doi: 10.1007/s10766-022-00734-5.

[3] Alwan, E., Al Baity, R. "Optimizing Program Efficiency with Loop Unroll Factor Prediction", Information Sciences Letters, 2023, 12(6), pp. 2207–2213

[4] S. Parsa and M. Hamzei, “NESTED-LOOPS TILING FOR PARALLELIZATION AND LOCALITY OPTIMIZATION,” *Computing and Informatics*, vol. 36, pp. 566–596, 2017, doi: 10.4149/cai.

[5] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, “Ensemble Machine Learning Paradigms in Software Defect Prediction,” in *Procedia Computer Science*, Elsevier B.V., 2022, pp. 199–209. doi: 10.1016/j.procs.2023.01.002.

[6] https://www.kaggle.com/datasets/noorasalam/tile-size-selection

[7] D. Joseph, J. L. Aragón, J.-M. Parcerisa, and A. González, “TCOR: A Tile Cache with Optimal Replacement.”

[8] H. Wu and D. Levinson, “The ensemble approach to forecasting: A review and synthesis,” *Transp Res Part C Emerg Technol*, vol. 132, Nov. 2021, doi: 10.1016/j.trc.2021.103357.

[9] T. M. Hope, "Linear regression," in Machine Learning, Academic Press, 2020, pp. 67-81.

[10] M. P. Rajan, "An efficient Ridge regression algorithm with parameter estimation for data analysis in machine learning," SN Computer Science, vol. 3, no. 2, p. 171, Mar. 2022.

[11] P. Jain, A. Choudhury, P. Dutta, K. Kalita, and P. Barsocchi, "Random forest regression-based machine learning model for accurate estimation of fluid flow in curved pipes," Processes, vol. 9, no. 11, p. 2095, Nov. 2021.

[12] U. Kumar, R. Bondhugula, B-Tech, P. Sadayappan, A. Atanas, R. Gagan, and A. J. Ramanujam, “Effective Automatic Parallelization and Locality Optimization Using The Polyhedral Model Dissertation Committee.”

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024

[13] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model."

[14] J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, "Chapter 5-Source code transformations and optimizations," in Embedded Computing for High Performance, pp. 137-183, 2017.

[15] F. Johnson, O. Oluwatobi, O. Folorunso, A. Ojumu, and A. Quadri, "Optimized ensemble machine learning model for software bugs prediction," in Innovations in Systems and Software Engineering, vol. 19, pp. 1-11, 2022, doi: 10.1007/s11334-022-00506-x.

[16] S. Afrifa, V. Varadarajan, P. Appiahene, T. Zhang, and E. A. Domfeh, "Ensemble Machine Learning Techniques for Accurate and Efficient Detection of Botnet Attacks in Connected Computers," Eng, vol. 4, no. 1, pp. 650–664, Mar. 2023, doi: 10.3390/eng4010039.

[17] S. Liu, Y. Cui, Q. Jiang, Q. Wang, and W. Wu, "An efficient tile size selection model based on machine learning," *J Parallel Distrib Comput*, vol. 121, pp. 27–41, Nov. 2018, doi: 10.1016/j.jpdc.2018.06.005.

[18] https://www.ece.lsu.edu/jxr/pluto/index.html

[19] A. C. De Melo, "The new linux 'perf' tools," in Slides from Linux Kongress, vol. 18, pp. 1-42, Sep. 2010.

Doi: https://doi.org/10.54216/FPA.150117
Received: August 26, 2023 Revised: December 12, 2023 Accepted: March 15, 2024