Efficient Code Region Characterization Through Automatic Performance Counters Reduction Using Machine Learning Techniques

Suren Harutyunyan[1](✉), Eduardo César[1], Anna Sikora[1], Jiří Filipovič[2], Akash Dutta[3], Ali Jannesari[3], and Jordi Alcaraz[4]

[1] Universitat Autònoma de Barcelona, Bellaterra, Spain
{suren.harutyunyan,eduardo.cesar,anna.sikora}@uab.cat
[2] Masaryk University, Brno, Czech Republic
fila@ics.muni.cz
[3] Iowa State University, Ames, IA, USA
{adutta,jannesar}@iastate.edu
[4] University of Oregon, Eugene, OR, USA
jordia@uoregon.edu

Abstract. Leveraging hardware performance counters provides valuable insights into system resource utilization, aiding performance analysis and tuning for parallel applications. The available counters vary with architecture and are collected at execution time. Their abundance and the limited number of registers for measurement make gathering laborious and costly. Efficient characterization of parallel regions necessitates a dimension reduction strategy. While recent efforts have focused on manually reducing the number of counters for specific architectures, this paper introduces a novel approach: an automatic dimension reduction technique for efficiently characterizing parallel code regions across diverse architectures. The methodology is based on Machine Learning ensembles because of their precision and ability at capturing different relationships between the input features and the target variables. Evaluation results show that ensembles can successfully reduce the number of hardware performance counters that characterize a code region. We validate our approach on CPUs using a comprehensive dataset of OpenMP regions, showing that any region can be accurately characterized by 8 relevant hardware performance counters. In addition, we also apply the proposed methodology on GPUs using a reduced set of kernels, demonstrating its effectiveness across various hardware configurations and workloads.

Keywords: Performance Counters · Automatic Dimension Reduction · Machine Learning Ensembles · Parallel Region Classification

This work was supported by the Ministerio de Ciencia e Innovación MCIN AEI/10.13039/501100011033 under contract PID2020-113614RB-C21, by the Catalan government under contract 2021 SGR 00574, and by the Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project "e-Infrastructure CZ - LM2023054.

1 Introduction

Modern HPC systems present challenges for performance analysis and tuning of parallel applications. Automated analysis during runtime contributes to this complexity, since it must be performed efficiently in a landscape of heterogeneous architectures.

Gathering relevant metrics, especially at the processor level, is crucial for effective behaviour and performance analysis. Hardware performance counters (HwPCs) are a set of special-purpose registers that store metrics about the use of system resources and hardware related activities. These metrics provide valuable insights and can be used for low-level performance analysis or tuning.

Several works [1,8,9,18] propose leveraging HwPCs' to identify and characterize parallel regions during runtime. However, each processor exposes a large number of HwPCs, which cannot be read simultaneously.

Two primary methods are used for collecting HwPC metrics: i) executing an application multiple times to gather accurate measurements at a high cost, ii) a less costly single execution that multiplexes the limited number of registers among performance monitoring events at the cost of precision. These problems underscore the importance of reducing the number of HwPCs used to characterize and classify a code region to achieve higher precision, reduce overhead, and enable a more efficient dynamic tuning process.

Recently, Alcaraz et al. [1] introduced a method that helped characterize a parallel code region using a reduced list of HwPCs. The methodology was effective if the original set of HwPCs and the number of code regions were not too big, because it requires human intervention. This manual intervention is necessary to check and discard redundant HwPCs.

In this work, we propose an automated technique that can accurately characterize parallel code regions. The methodology focuses on Machine Learning ensembles that can automatically identify, rank, and obtain a reduced list of HwPCs independently of the number of available counters and regions. The proposed approach is architecture agnostic and works equally well for both CPUs and GPUs. To prove its effectiveness, this ensemble methodology is tested with an exhaustive database with a large number of kernels on CPUs, and on 2 different GPU architectures with medium sized databases. Results show that ensembles models produce a reduced set of HwPCs that accurately characterizes OpenMP and CUDA parallel code regions.

This paper is organized as follows. Section 2 presents the motivation of this work. Section 3 introduces the rationale behind ensembles, the techniques used in the proposed methodology, and how these are evaluated. Section 4 describes the main contribution of this paper: the ensemble methodology for automatic ranking and reduction of HwPCs. Section 5 shows the evaluation of the methodology on different architectures. Section 6 presents relevant related work. Finally, Sect. 7 concludes the work and discusses its future directions.

2 Motivation

Alcaraz et al. in [1] presented a methodology, based on Principal Component Analysis (PCA) and correlation analysis, for reducing the number of HwPCs needed for characterizing OpenMP parallel regions. Principal Component Analysis was used to visualize the different kernels and verify whether they can be classified visually. It can also be applied to discover relationships among HwPCs weights in each principal component, where similar weights potentially point out redundancies. In addition, if the PCA resulting from excluding a set of HwPCs remained similar to the previous PCA and there is minimal impact on the explained variance with fewer dimensions, the reduction can be deemed valid.

Finally, correlation analysis was used to reduce HwPCs. These were discarded if the correlation value between two, or more, counters was very high and could be logically explained. Very high correlation points to the possibility of counters explaining similar information. The methodology was tested on a dataset created by executing the STREAM benchmark [12], which consists of 4 different code patterns. The different execution configurations consisted of various problem sizes, compilation flags, etc.

This reduction method has some serious limitations. It involves the user intervention, and is effective when the number of HwPCs and code regions is small. As the number of HwPCs and tested configurations can be extremely large, applying the same methodology can become unwieldy. To overcome these limitations, we choose to leverage Machine Learning techniques because they have demonstrated unparalleled efficacy and precision in tackling tasks necessitating the analysis of extensive datasets.

3 Supervised Machine Learning Algorithms

In this section, a theoretical description of the concepts that are used in the presented research is provided.

The methodology designed for automatically identifying the set of relevant HwPCs consists of an ensemble of supervised models presented in Table 1. A ML ensemble is a general meta approach that combines different models trained simultaneously on the same dataset. The models are trained to operate jointly to produce better performance results than any individual model on its own because of the following reasons [17]:

1. **Reduce bias and variance:** Multiple models in an ensemble can complement each other and help reduce the bias and variance of the overall model. This can result in more accurate predictions.
2. **Combine different approaches:** Ensembles combine different approaches or models that may have different strengths and weaknesses. This can help overcome the limitations of individual models and lead to better overall performance.
3. **Robustness:** ML ensembles can still perform well even if some of the individual models are not performing well.

Table 1. Models and metrics used for classification

Algorithms	Metric
Logistic Model with Elastic Net	Accuracy
Random Forest	
XGBoost	
TabNet	

4. **Diversity:** ML ensembles incorporate a range of different models which make them more generalizable and better able to handle a variety of inputs.

Overall, ML ensembles represent a powerful instrument for enhancing predictive efficacy, including classification accuracy, and attaining superior results across diverse application domains. For the present work the models discussed below were selected based on their precision and their ability at capturing different relationships between the input features and the target variables. Additionally, other factors such as the training cost were considered.

Logistic Model with Elastic Net is a combination of the Logistic Model [7] with Elastic Net [23], which integrates both L1 (Lasso, or Least Absolute Shrinkage Selector Operator Regression) [19] and L2 (Ridge) [10] regularizations. This combination offers advantages in managing multicollinearity among predictors, mitigating overfitting, and facilitating feature selection. The model is computationally efficient and effective in scenarios where there exists clear class separation and simple relationships between features and target variables.

Random Forest [4] is a flexible meta-estimator that can be used for both regression and classification. It is based on Bagging (Bootstrap Aggregation), it is resilient to outliers, and is most effective for datasets with a small to medium number of features. It is mainly used to capture the nonlinear relationships between them. This approach first fits in parallel a number of decision trees on various random sub-samples of the dataset, called Bootstrap Samples (or bags). Second, it produces the output by averaging (Aggregating) the individual outputs, thus improving the predictive accuracy and controlling over-fitting.

XGBoost (Extreme Gradient Boosting) [6] is a fast and scalable implementation of gradient-boosted trees. It reduces the tree search space via information given by the distribution of the features. This approach can be used for both regression and classification, handling a wide range of data sizes. It is effective on datasets with imbalanced class distributions. It is based on Boosting, an iterative technique that sequentially builds models, assesses their errors, and assigns higher weights to incorrectly predicted values to improve the next models. Since each new model aims to rectify the errors of the previous one, the result is a model with the highest accuracy.

TabNet [3] is a Deep Neural Network designed for tabular data capable of performing both classification and regression tasks. It is especially effective in scenarios where the dataset has a large number of features with complex relationships to the target variable. It employs a learner that resembles decision tree-based models, aiming to achieve interpretability and sparse feature selection. TabNet performs both output mapping and feature selection, making it a powerful tool for analyzing and understanding complex tabular datasets.

Accuracy has been used as the metric to score the component models of the ensemble. It measures the proportion of correctly classified instances among all instances, indicating the closeness of the model's predictions to the ground truth value. It is commonly used because of its simplicity and intuitive interpretation, making it easy to understand.

The following is the formulation of the accuracy score in the case of a binary classification:

$$\text{Binary Classification Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{1}$$

where TP denotes the True Positives, FP are the False Positives, TN are the True Negatives, and FN are the False Negatives.

Conversely, in the case of multiclass classification, the accuracy score is the fraction of correct classifications and is formulated as follows:

$$\text{Multiclass Classification Accuracy} = \frac{\text{Correct Classifications}}{\text{All Classifications}} \tag{2}$$

After the ensemble is trained, during the inference phase, each model generates its own classification prediction. The final class is determined through a majority vote mechanism, where the most predicted class is the final label. This serves to mitigate the individual variance of the predictions, thereby enhancing the robustness of the results.

4 Performance Counters Reduction Using ML Ensembles

In this section we describe the proposed HwPC reduction methodology presented on Fig. 1. Given a database containing values of HwPCs for various code regions, the methodology reduces the number of HwPCs to a minimal set capable of accurately characterizing and classifying any code region. This minimal set is established through the scoring and ranking of HwPCs using a Machine Learning ensemble. Step 1 is composed of a training and validation of an ensemble of classifier models. The models analyze the relationship between the code regions and a set of HwPCs. After training/validation, Step 2 is used to extract the importance of each HwPC for each model. Step 3 reduces the ranked HwPCs to the minimal set that characterize a code region.

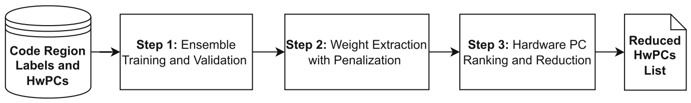

Code Region Labels and HwPCs
Step 1: Ensemble Training and Validation
Step 2: Weight Extraction with Penalization
Step 3: Hardware PC Ranking and Reduction
Reduced HwPCs List

Fig. 1. Performance counters reduction with model ensembles.

Code Regions Labels and HwPCs. First, we provide a brief description of the data sources and the input for the methodology. For each code region, data has been collected on execution time, tuning parameters, and HwPCs' values. As we are looking for the most important HwPCs that characterize code regions, new datasets that only include the code region label and HwPCs' values are generated. These datasets are pre-processed for deleting features with 0 variance.

Step 1: Ensemble Training and Validation. Figure 2 shows Step 1 (ensemble training and validation) components. The methods included for the classification task are Logistic Regression with Elastic Net, Random Forest, XGBoost, and TabNet. As discussed in Sect. 3, each model has a different set of strengths and captures different relationships between the input features and the target variable. Altogether, they capture linear and non-linear relationships that may arise between code regions and HwPCs. For example, Logistic Regression focuses on linear relationships, while XGBoost focuses on non-linear ones.

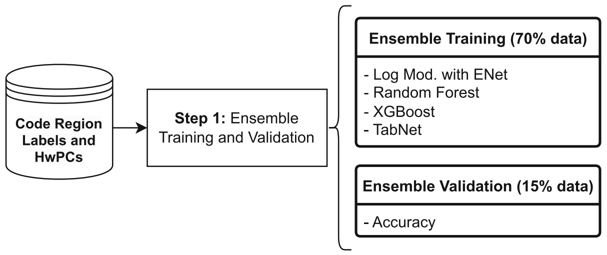

Code Region Labels and HwPCs
Step 1: Ensemble Training and Validation
Ensemble Training (70% data)
- Log Mod. with ENet
- Random Forest
- XGBoost
- TabNet
Ensemble Validation (15% data)
- Accuracy

Fig. 2. Step 1: Ensemble training and validation.

Additionally, each different model is more effective for datasets of different sizes and class distributions. Random Forest and XGBoost excel on large datasets with large numbers of features and samples. On the other hand, TabNet offers the advantage of a competent classifier when the labeled data is limited. This is paramount since the collected kernel execution data can differ not only on the number of features (HwPCs) but also on the number of available tuning configurations/parameters, and samples. Thus a combination can leverage the strengths of each model and improve overall performance.

The data is split into 70% for training, 15% for validation, and 15% for testing. After training, the models are validated using the accuracy metric. It is used to determine the most performant models, and is later used to form the weights given to the predictions of each model.

Step 2: Weight Extraction with Penalization. Once the training is complete, the models assign scores to the HwPCs, which represent the relative importance of each feature in making the prediction. These scores are normalized from 0.0 to 1.0. The scores are computed as follows:

1. **Logistic Model with Elastic Net**: The importance of features is determined by the magnitude of the coefficients learned during model training. A higher absolute magnitude of the coefficient indicates greater importance of the corresponding HwPC in influencing the predicted outcome.
2. **Random Forest**: **Random Forest**: Feature importance is determined by the average decrease in impurity, measured by either Gini impurity or entropy, caused by each HwPC when used to split the data across all decision trees within the forest. HwPCs that consistently result in larger reductions in impurity across multiple trees are assigned higher importance scores.
3. **XGBoost**: Feature importance is determined by the average gain of each feature when employed to split the data across all decision trees within the ensemble. Gain is defined as the improvement in model performance achieved by splitting the data based on a particular HwPC, typically measured by the reduction in loss. HwPCs with higher average gain across all trees are considered more influential in the model.
4. **TabNet**: **TabNet**: Feature importance scores are determined by the contribution of each feature to the final decision made by the neural network architecture. The model utilizes an attention mechanism to dynamically weigh the importance of each HwPC at every decision step within the network.

To combine the different scores across the models we use a weighted average that assigns higher weights to the more accurate models. The weights are determined by accuracy of the models during the validation process. In this way, models with higher accuracy have more influence in the HwPC's score. Accuracy directly reflects the performance of each model, making models with higher accuracy generally more reliable and capable of producing better predictions.

Let $HwPC = \{h_0, h_1, ..., h_{N-1}\}$ be a set of N HwPCs. The ith HwPC has a set of corresponding model scores or coefficients $\{c_{i_0}, c_{i_1}, c_{i_2}, c_{i_3}\}$. There is also a set of discount factor weights extracted during the validation of the ensemble $\{w_0, w_1, w_2, w_3\}$. The weighted average of the ith HwPC's score is calculated with Expression 3, where j denotes the model in the ensemble.

$$W_i = \frac{\sum_{j=1}^{4} c_{i_j} w_j}{\sum_{j=1}^{4} w_j}, \tag{3}$$

Figure 3 illustrates the weight extraction process. Each model has a weight (accuracy) associated to its predictions and assigns a score to each HwPC. The final score is the weighted average of the scores of the HwPCs.

Step 3: Hardware Performance Counter Ranking and Reduction. Once the weighted average of the scores for each HwPC is calculated, it is used to arrange the list of HwPCs in descending order.

From this ordered list, we apply Algorithm 1 to determine the minimal set of HwPCs capable of characterizing the parallel regions in the dataset. Firstly, we calculate the reference line shown in Fig. 4b, which passes through points (0,0) and (N-1, accuracyN-1). This line extends from the point with the lowest

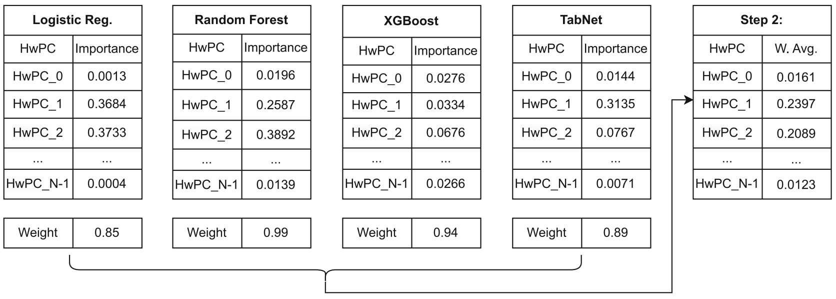


Fig. 3. Step 2: HwPC importance per model and weighted average.

Algorithm 1 . HwPCs Reduction

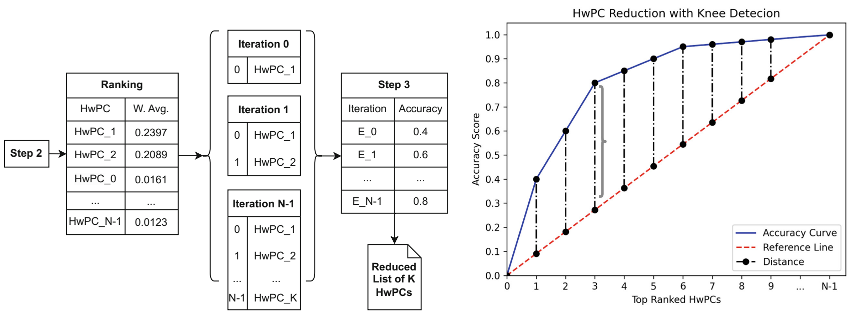
```
Input: List $H$ of $N$ HwPCs ordered by rank
Output: Reduced list of $K$ HwPCs
/* The reference line is the one going from the accuracy using 0 HwPCs
(0) and the accuracy using all HwPCs (already computed in Step 1) */
 1: Compute straight line $L$ between $(0,0)$ and $(N-1, Accuracy_{N-1})$
 2: $K \leftarrow 1$
 3: $OldDiff \leftarrow 0$ // The accuracy with 0 HwPCs
 4: Obtain sub-array $S_1$ of first HwPC from $H$
 5: Train and validate an Ensemble using $S_1$
 6: $Acc \leftarrow$ validation accuracy
 7: $NewDiff \leftarrow$ distance between $(1, Acc)$ and $L$
 8: while $NewDiff > OldDiff$ do
        /* Determining the knee point */
 9:         $K++$
10:         Obtain sub-array $S_k$ of first $k$ HwPCs from $H$
11:         Train and validate an Ensemble using $S_k$
12:         $Acc \leftarrow$ validation accuracy
13:         $OldDiff \leftarrow NewDiff$
14:         $NewDiff \leftarrow$ Distance between $(k, Acc)$ and $L$
15: end while
16: $Result \leftarrow$ First $K-1$ HwPCs from $H$
```

accuracy (0 HwPCs yields 0 accuracy) to the point corresponding to the accuracy achieved in the validation of the model trained using all HwPCs. The latter is known as it was calculated in Step 1 of the methodology.

Subsequently, we iteratively proceed to use the ensemble to train models using an increasing number of HwPCs taken in order from the ranked list, as shown in Fig. 4a. We start by training a model with the highest-ranked HwPC, then a new model using the top two ranked HwPCs, and so on. In each case, we compute the distance between the reference line and the accuracy obtained by the model using a certain number of HwPCs, as shown in Fig. 4b. This iterative process concludes when this distance ceases to increase upon adding a new

HwPC, indicating that the knee point has been reached and that the increase in accuracy obtained becomes marginal from this point.

Finally, the minimum set of HwPCs capable of characterizing the dataset's regions consists of those for which the distance continued to increase.



(a) Weight ranking and reduction process (b) Knee detection of the accuracy curve

Fig. 4. Step 3: HwPC reduction considering the knee point.

It is worth noting that if the dataset used to apply the methodology includes representative regions of the space of possible parallel regions, the model generated for the reduced set of HwPCs would likely classify regions not included in the dataset with high accuracy.

5 Evaluation

In this section, we present the results obtained using the proposed ensemble methodology for different cases. First, we compare the results produced using our methodology with those reported in Alcaraz et al. [1]. Next, we assess the effectiveness of our methodology using a comprehensive dataset of OpenMP parallel regions on CPUs. Finally, we show that the methodology can also be applied to classify GPU kernels.

5.1 Comparison with PCA and Correlation-Based Methodology

We begin by comparing the results obtained using our methodology to those reported in [1]. Employing the same benchmark, STREAM [12], and the same hardware, a Xeon E5645 processor. STREAM comprises four patterns -Copy, Scale, Sum, and Triad- each exhibiting distinct memory access patterns and operation counts.

The HwPCs were collected using the preset events available in Performance Application Programming Interface (PAPI) [14]. In this architecture, the set of preset HwPCs included the following event types and the corresponding number of counters:

- **Branches**: 7
- **Cache L1**: 8
- **Cache L2**: 16
- **Cache L3**: 10
- **TLB**: 3
- **Cycles**: 3
- **Operations**: 3
- **Instructions**: 8

As a result of applying the PCA and correlation-based methodology, the number of HwPCs neccesary to classify these patterns was reduced from 58 to 20.

Using the proposed ensemble methodology, the number of HwPCs is reduced from 58 to 5, which includes a diverse set of event types:

- **Instructions**: `PAPI_SR_INS`
- **Operations**: `PAPI_FP_OPS` and `PAPI_DP_OPS`
- **Branches**: `PAPI_BR_TKN`
- **Cache L1**: `PAPI_L1_ICH`

Four of them (`PAPI_DP_OPS`, `PAPI_FP_OPS`, `PAPI_BR_TKN`, `PAPI_SR_INS`) are also present in the reduced set from [1], and the remaining one, `PAPI_L1_ICH`, is highly correlated with `PAPI_BR_MSP` (0.9781) from this set.

Table 2. Evaluation results for Xeon E5645.

Results/Methodology	PCA & Correlation	Ensemble 1	Ensemble 2
Reduced Set	20	5	5
Accuracy	99.98%	93.83%	99.20%
K Fold Accuracy	–	85.94%	96.10%

In [1], the reduced set was later used to train an Artificial Neural Network (ANN), which was utilized to predict the parallel code region based on the values of HwPCs. The training set consisted of 54 problem sizes (432000 entries), while the remaining 2 sizes were reserved for testing (16000 entries). Table 2 shows that the ANN accurately classified 99.98% of the test set in this case.

Using the proposed methodology with the same train/test split, the model (Ensemble 1 in the table) accurately classified 93.83% of the test set. Our stratified 5 fold cross validation, achieving an average cross validated score of 85.94%.

However, applying a random stratified train/validation/test split common in ML, 70% of data for training, 15% for validation, and 15% for testing, the model (Ensemble 2 in the table) was able to classify accurately 99.20% of the test set. We also performed a stratified 5 fold cross validation, achieving an average cross validated score of 96.10%.

Therefore, the proposed methodology achieves comparable results using only 5 automatically determined HwPCs, minimizing event multiplexing and thereby enhancing measurement precision.

5.2 Comprehensive Dataset of OpenMP Regions

Our methodology was further evaluated on a different architecture, a 32-core Xeon E5-4620, using a more comprehensive dataset consisting of 18 kernels. This dataset included 4 kernels from the STREAM benchmark [12], 12 benchmarks from the PolyBench suite [21,22], which comprises synthetic benchmarks representing common computational kernels in scientific and engineering applications. Finally, we included Collatz conjecture and a Friendly number calculator.

The HwPCs were collected using the preset events available in PAPI for this architecture. These events were separated into compatible groups, while taking into account the maximum number of events that can be read at the same time in one processor and if they can be accessed simultaneously [1]. We measured each group of HwPCs for multiple executions of the code regions, with different configuration combinations (number of threads, affinity policy, scheduling policy, and chunk size). The problem sizes were computed using the methodology presented in [2], which focuses on stressing the different memory levels available in the architecture. For each of the HwPC groups, problem sizes, and configurations, we conducted 100 executions for statistical significance. Table 3a shows the summary of the configurations for the executions of each region.

In this architecture, the set of preset HwPCs included the following event types and the corresponding number of counters:

- **Branches**: 6
- **Cache L1**: 5
- **Cache L2**: 14
- **Cache L3**: 9
- **TLB**: 2
- **Cycles**: 3
- **Operations**: 3
- **Instructions**: 8

Using the proposed ensemble methodology, the number of HwPCs is reduced from 50 to 6, which, again, include a diverse set of event types:

- **Instructions**: `PAPI_SR_INS` and `PAPI_FDV_INS`
- **Operations**: `PAPI_FP_OPS` and `PAPI_DP_OPS`
- **Branches**: `PAPI_BR_NTK` and `PAPI_BR_MSP`

As mentioned in Sect. 2, the number of kernels (18) and the large number of tested configurations (more than 12M) precludes us of using the PCA/Correlation based method of the previous subsection on the comprehensive dataset.

As in the previous case, we tested the results of the proposed ensemble methodology by training a model ensemble using a random stratified train/validation/test split (70% of the data for training, 15% for validation, and the remaining 15% for testing). The obtained accuracy on the test set was 96.95%, as shown in Table 3b. We also performed a stratified 5 fold cross validation, achieving an average cross validated score of 96.82%.

Once more, the proposed methodology achieves very good results using only 6 automatically determined HwPCs, minimizing event multiplexing and thereby

Table 3. Execution parameters and evaluation results for Xeon E5-4620.

(a) Execution parameters.

Execs./System	Xeon E5-4620
HwPC Sets	12
Threads	32
Problem Sizes	29
OpenMP Pars.	11
Reps.	100
Total	12,249,600

(b) Evaluation results.

Results/Method.	Ensembles
Reduced Set	6
Accuracy	96.95%
K Fold Accuracy	96.82%

enhancing measurement precision. Moreover, for a dataset including a significantly greater amount of regions and configurations, the methodology is still sharply reducing the number of HwPCs.

5.3 Applying the Methodology for Characterizing GPU Kernels

The last experiments have the objective of showing that the proposed methodology can be also applied for reducing the number of HwPCs necessary to characterize GPU kernels. HwPC data was collected on a set of benchmarks, either exhaustively or through random sampling, using the Kernel Tuning Toolkit (KTT) [15] on 2 different GPUs, the GeForce GTX 680 (Kepler) and the GeForce GTX 1070 (Pascal). For each configuration, data comprising tuning parameter values, execution time, and HwPC values was obtained on five kernels analyzed by Petrovič et al. in [16]: Convolution, Couloumb Sum, Nbody, Transposition, and GEMM, with the inclusion of the Reduction kernel for the Pascal GPU.

Using the ensemble methodology with the Kepler dataset (5 kernels), the number of HwPCs was reduced from 35 to 3, while for the Pascal dataset (6 kernels) it was reduced from 167 to 4. The resulting HwPCs are the following:

GTX 680 (Kepler):

- `inst_fp_32`
- `l2_write_transactions`
- `inst_executed`

GTX 1080 (Pascal):

- `inst_per_warp`
- `gld_requested_throughput`
- `gst_requested_throughput`
- `dram_read_throughput`

Given the smaller size of the datasets compared to the CPU ones, we opted for larger holdout datasets. Instead of the traditional 15/15 split for validation and testing, we implemented a random stratified train/validation/test split, allocating 70% of the data for training, 3% for validation, and 27% for testing. We believe that using 27% of the dataset for testing provides a more robust assessment of the effectiveness of our methodology in this context. Table 4 shows that the trained model achieves 99.94% accuracy on the Kepler dataset and 99.81%

accuracy on the Pascal dataset. We also performed a stratified 5 fold cross validation, achieving an average cross validated score of 99.99% on the Kepler dataset, and 99.43% on the Pascal dataset. It is highly probable that the cause of such sharp reduction in the number of HwPCs is the small number of kernels in the dataset. Nevertheless, the obtained results are also very promising for GPUs.

Table 4. Counters' reduction and evaluation results for GeForce GTX 680 (Kepler) and GeForce GTX 1080 (Pascal).

Results/System	GeForce GTX 680	GeForce GTX 1080
Full Set	35	167
Reduced Set	3	4
Accuracy	99.94%	99.81%
K Fold Accuracy	99.99%	99.43%

6 Related Work

The presented methodology involves the application of ML model ensembles to characterize a code region through a reduced set of HwPCs. There are several studies that make use of statistical techniques, and also pursue the objective of characterizing code regions.

On the one hand, there are approaches that involve the characterization of code regions via a representative dataset composed of multiple kernels and configurations. READEX [11] authors use Periscope Tuning Framework [13] and a representative dataset to record timing related data. This performance data is then used to discover relevant code regions in the application. Akash et al. [8] were able to determine parallel loop patterns using HwPCs and code graph representations of the code regions. On the other hand, other approaches focus on a specific application, exclusively collecting its data. In [18] the authors utilize CERE to characterize code regions, using clustering techniques to identify parallel regions with similar characteristics. The applications under consideration are decomposed into smaller code segments, *codelets*, which are then associated with corresponding OpenMP parallel regions. Artemis [20] uses collected execution data from instrumented parallel regions, and user provided features and execution policies for the code region. Extra-P [5] uses HwPCs and other metrics, such as the number of bytes an MPI function sends or receives, to generate a parallel profile of an application. In all the mentioned tools, there is a selection of which HwPCs should be used to characterize code regions. However, to the best of our knowledge, none of them uses a systematic methodology to determine the set of the most relevant HwPCs, as the one proposed in this work.

7 Conclusions and Future Work

In this paper, we introduced a novel method of reducing the HwPCs that characterize a parallel region. The proposed methodology, based on Machine Learning ensembles aimed to be automatic, and independent of hardware architecture. The ensemble methodology has been tested on CPUs and GPUs. The CPU tests were performed on 2 different sets corresponding to different architectures. During evaluation, we were able to significantly reduce the HwPCs required for characterization of a parallel code region. The reduced set only contained 5 and 6 HwPCs, and the ensemble methodology was able to correctly classify 99.20% and 96.95% of the parallel regions, respectively. To validate the architecture independence of the methodology, we evaluated it on 2 different GPU architectures with a small set of CUDA kernels. Although the collected databasets were not as large as in the CPU case, it is clearly demonstrated that the ensemble method drastically reduces the number of HwPCs necessary to characterize a CUDA kernel. The reduced set contains 3 and 4 HwPCs classifying kernels with 99.94 and 99.81% of accuracy, respectively. These results are significant, since the sheer amount of available HwPCs on GPUs yields impossible any manual effort. Currently, we are using ensembles of ML regression models to determine a reduced set of HwPCs for optimizing the values of tuning parameters.

References

1. Alcaraz, J., Sikora, A., César, E.: Hardware counters' space reduction for code region characterization. In: Yahyapour, R. (ed.) Euro-Par 2019. LNCS, vol. 11725, pp. 74–86. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29400-7_6
2. Alcaraz, J., et al.: Predicting number of threads using balanced datasets for openmp regions. PDP Spec. Issue 2021 Comput. **105**(5), 999–1017 (2023)
3. Arik, S.Ö., Pfister, T.: Tabnet: attentive interpretable tabular learning. arXiv preprint arXiv:1908.07442 (2019)
4. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
5. Calotoiu, A., Hoefler, T., Poke, M., Wolf, F.: Using automated performance modeling to find scalability bugs in complex codes. In: Proceedings of the ACM/IEEE Conference on SC13, Denver, pp. 1–12. ACM (2013)
6. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 16), pp. 785–794. ACM, New York (2016)
7. Cox, D.R.: The regression analysis of binary sequences. J. Roy. Stat. Soc.: Ser. B (Methodol.) **20**(2), 215–232 (1958)
8. Dutta, A., Alcaraz, J., TehraniJamsaz, A., Sikora, A., Cesar, E., Jannesari, A.: Pattern-based autotuning of openmp loops using graph neural networks. In: 2022 IEEE/ACM International Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S), pp. 26–31 (2022)
9. Filipovič, J., Hozzová, J.A.N., Oľha, J., Petrovič, F.: Using hardware performance counters to speed up autotuning convergence on gpus. J. Parall. Distrib. Comput. **160**, 16–35 (2022)
10. Hoerl, A.E., Kennard, R.W.: Ridge regression: biased estimation for nonorthogonal problems. Technometrics **12**(1), 55–67 (1970)

11. Kjeldsberg, P.G., Gocht, A., Gerndt, M., Riha, L., Schuchart, J., Mian, U.S.: Readex: linking two ends of the computing continuum to improve energy-efficiency in dynamic applications. In: Design, Automation Test in Europe Conference Exhibition, 2017, pp. 109–114 (2017)
12. McCalpin, J.: Memory bandwidth and machine balance in high performance computers. In: IEEE Technical Committee on Computer Architecture Newsletter, pp. 19–25 (1995)
13. Miceli, R., et al.: Autotune: a plugin-driven approach to the automatic tuning of parallel applications. In: Proceedings of the 11th International Workshop on the State-of-the-Art in Scientific and Parallel Computing (PARA 2012), vol. 7782, 328–342 (2013)
14. Mucci, P., Moore, S., Deane, C., Ho, G.: Papi: a portable interface to hardware performance counters (1999)
15. Petrovič, F., Filipovič, J.: Kernel tuning toolkit. SoftwareX **22**, 101385 (2023)
16. Petrovič, F., et al.: A benchmark set of highly-efficient CUDA and OpenCL kernels and its dynamic autotuning with kernel tuning toolkit. Futur. Gener. Comput. Syst. **108**, 161–177 (2020)
17. Polikar, R.: Ensemble based systems in decision making. IEEE Circuits Syst. Mag. **6**(3), 21–45 (2006)
18. Popov, M., Akel, C., Chatelain, Y., Jalby, W., de Oliveira Castro, P.: Piecewise holistic autotuning of parallel programs with CERE. Concurr. Comput. Pract. Exp. **29** (2017)
19. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. Roy. Stat. Soc.: Ser. B (Methodol.) **58**(1), 267–288 (1996)
20. Wood, C., et al.: Artemis: automatic runtime tuning of parallel execution parameters using machine learning. In: Chamberlain, B.L., Varbanescu, A.-L., Ltaief, H., Luszczek, P. (eds.) ISC High Performance 2021. LNCS, vol. 12728, pp. 453–472. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78713-4_24
21. Yuki, T., Pouchet, L.N.: Polybench 4.0 (2015). https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/
22. Yuki, T.: Understanding PolyBench/C 3.2 kernels. In: Rajopadhye, S., Verdoolaege, S. (eds.) Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques, Vienna (2014)
23. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. J. Roy. Statist. Soc. Ser. B (Statist. Methodol.) **67**(2), 301–320 (2005)