2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)

2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) | 978-1-6654-1455-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/PDP52278.2021.00019

Building representative and balanced datasets of OpenMP parallel regions

Jordi Alcaraz*, Steven Sleder†, Ali TehraniJamsaz†, Anna Sikora*, Ali Jannesari†, Joan Sorribes* and Eduardo Cesar*
*Universitat Autònoma de Barcelona, Cerdanyola, Spain
Email: {jordi.alcaraz, anna.sikora, joan.sorribes, eduardo.cesar}@uab.cat
†Iowa State University, Ames, IA, USA
Email: {ssleder, tehrani, jannesari}@iastate.edu

***Abstract*—Incorporating machine learning into automatic performance analysis and tuning tools is a promising path to tackle the increasing heterogeneity of current HPC applications. However, this introduces the need for generating balanced and representative datasets of parallel applications' executions. This work proposes a methodology for building datasets of OpenMP parallel code regions patterns. It allows for determining whether a given code region covers a unique part of the pattern input space not covered by the patterns already included in the dataset. The proposed methodology uses hardware performance counters to represent the execution of the region, which is referred to as the region signature for a given number of cores. Then, a complete representation of the region is built by joining the signatures for every different thread configuration in the system. Next, correlation analysis is performed between this representation and the representation of all the patterns already in the training set. Finally, if this correlation is below a given threshold, the region is considered to cover a unique part of the pattern input space and is subsequently added to the dataset. For validating this methodology, an example dataset, obtained from well known benchmarks, has been used to train a carefully designed neural network model to demonstrate that it is able to classify different patterns of OpenMP parallel regions.**

***Index Terms*—hardware performance counters, machine learning, artificial neural networks, parallel applications, OpenMP**

I. Introduction

The increasing heterogeneity and complexity of current HPC systems escalates the difficulty of performance analysis and optimization of parallel applications. This is further compounded as both novel problems appear and the number of significant tuning parameters increases. Logically, this fact also affects the premises on which performance analysis and tuning tools are built. We claim that these kind of tools should incorporate new strategies, such as machine learning (ML), to further adapt to the characteristics of current HPC systems.

In this work, we limit the scope of the research to applications parallelized using OpenMP as the mainstream multi-threaded programming model for the HPC community.

To be able to incorporate ML techniques into performance analysis and tuning tools, we must be able first to identify the features that allow for the characterization of parallel applications. The work presented in [1] showed that a signature built with the values of a subset of hardware performance counters (HWCs) measured using PAPI [2] can be used to characterize a set of OpenMP parallel regions. This solves the problem of identifying the features for OpenMP regions.

However, there is also the problem of building representative and balanced datasets for training purposes. In other words, how to determine if a given parallel region pattern shall be included in a training set or not. It is worth noticing that generation of balanced datasets is considered a crucial problem in machine learning a data mining [3] because in the presence of significant imbalances the accuracy cannot be a representative of the true performance of a classifier [4].

The objective of this work is to present an approach for systematically deciding whether a given OpenMP parallel region pattern covers a unique portion of the input space that is not currently considered covered by in a training set. The input space is the N-dimensional space defined by the valid combinations of N hardware performance counters values. In this space a pattern is an OpenMP fragment of code that abstracts different regions with a similar behaviour.

This methodology assumes that the set of hardware counters that must be measured to build the region signature have been already determined, and that there is a properly built training set. Then, given a parallel region kernel, (i) the values of its signature must be gathered and processed for building the kernel representation, and (ii) a non-linear correlation analysis is performed between this representation and the representation of the patterns in the dataset to decide if the kernel covers a new part of the input space and should become a new pattern in the collection.

One way to validate this methodology is to use the dataset in an ML technique and show that it is able to classify parallel regions accordingly to the patterns included in the set. However, the ML technique must be applied while taking steps to assure the quality of its results. In this work, we chose to use an Artificial Neural Network (ANN) as the validation ML technique for classifying OpenMP parallel region patterns.

The results obtained using this ANN demonstrate that the proposed methodology accurately indicates if a pattern covers a new part of the input space and shall be included or not in a training set. In addition, it can also be used to demonstrate

This work has the support of the Ministerio de Economía, Industria y Competitividad MINECO-SPAIN under contract TIN2017-84553-C2-1-R and by the Generalitat de Catalunya GenCat-DIUiE (GRR) with the project 2017-SGR-313.

978-1-6654-1455-5/21/$31.00 ©2021 IEEE
DOI 10.1109/PDP52278.2021.00019

the impact of using balanced training sets. Moreover, this ANN is important because it can be the first step towards the generation of tuning models using ML.

This work addresses the following research questions: i) Can a representative and balanced dataset of OpenMP regions be systematically built? and ii) What is the impact of using a balanced dataset in the accuracy of a classification model generated by an ANN?

Consequently, its contributions can be summarized in i) a methodology based in hardware performance counters and correlation for systematically building representative and balanced OpenMP parallel regions datasets, and ii) an ANN for classifying OpenMP parallel regions that illustrates the importance of using representative and balanced datasets.

This work is organized as follows. Section II summarizes the basic techniques applied in our methodology. Afterwards, Section III, presents the proposed methodology for determining if a given parallel region pattern shall be included in the training set. Next, Section IV shows how the proposed methodology can be used to build and validate a dataset using a set of kernels extracted from different well-known benchmarks. Related works are discussed in Section V. Finally, Section VI concludes this paper and discusses potential future work.

II. Background

This section introduces the mechanisms and mathematical tools used in this work for developing and validating the proposed methodology.

A. Correlation Coefficient

Correlation is a statistical technique that shows the relationship, or lack thereof, between independent variables. As such, it can be a powerful tool to determine whether or not different code regions correspond to similar patterns.

Of the several methods to perform correlation analysis, among the best known are Pearson's correlation coefficient, Spearman's rank correlation and Kendall's tau [5].

Pearson correlation coefficient leverages covariance to calculate the statistical and linear relationship between two datasets. Spearman rank correlation is a non-parametric method which measures the monotonic relationship between two datasets without making assumptions as to the frequency distribution. The last approach, Kendall's tau, is also a non-parametric method used to assess and test correlation between non-interval scaled data.

Pearson correlation coefficient is assumed to efficiently characterize relationships between independent variables which follow a normal distribution [5] [6], but not in the case of non-normal distributions. This assumption may not hold in the case of hardware performance counters and, consequently, this method has been discarded from the set of potential correlation analysis methods for this work.

B. Pattern recognition with Artificial Neural Networks

Applying an Artificial Neural Network (ANN) in this work provides two benefits. The first of these is to demonstrate the importance of using a representative and balanced dataset with a current and commonly-used ML approach. The second builds upon the results in [1] in which an ANN was shown to be capable of learning the relationship between performance counters and pattern labels. This result can be adapted to examine pattern correlations and therefore validate a dataset. A model trained on a constructed dataset when provided a testing set of discarded patterns should, with high accuracy, predict them as the patterns which they were found to correlate with when the dataset was built.

Towards this, we have used a Fully-Connected Feedfoward Neural Network (FNN), a class of neural network which, in addition to their current popularity, are able to arbitrarily approximate any function under certain conditions [7]. This network architecture also features several modifications which help to increase classification accuracy while maintaining a relatively low training overhead.

III. Methodology

This section introduces the methodology to determine whether a given OpenMP parallel region pattern covers a portion of the feature space that is not represented by the training set so far. Finding an effective solution to this issue is significant in ensuring a balanced and representative dataset. Intuitively, including one or more patterns which are nearly identical will not provide any further information and are a source of imbalance which reduces the quality of the dataset.

Figure 1 shows the general scheme of the methodology we propose. The *pattern collection* represents the current, potentially empty, training dataset, which we assume to be balanced. We consider that if the signature of a *candidate kernel* is not highly correlated with any pattern in the *pattern collection*, it covers a new part of the input space and should become a new pattern in the collection.

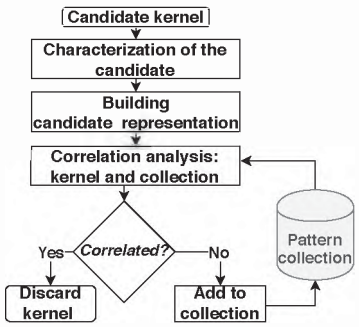

Fig. 1. Methodology for determining whether a pattern should be included or not in the training set.

In the case that we locate an OpenMP kernel which appears to be an attractive candidate for inclusion in the dataset, we use the following steps to verify whether or not it covers a new portion of the input space:

- **Characterization of the candidate kernel**.The kernel is executed to obtain its signature for each possible combination of number of threads and affinity.

- **Building candidate kernel representation**. After the kernel's data is available, the signatures are joined creating a representation of the *candidate kernel* in the form of a vector that will be used in the correlation analysis.
- **Correlation analysis**. Correlation analysis is performed between the vector and the vectors of each pattern considered to be representative, which are in the *pattern collection*, to determine the extent of their similarity.

A. Characterization of the candidate kernel

The first step for assessing a candidate kernel consists of generating its signatures. In order to obtain the kernel's signature for the OpenMP parallel region it must be executed multiple times (for ensuring statistical significance) for all possible combination of the following parameters:

- *Number of threads*. The kernel is executed using different thread configurations available in the system, from the minimum number of cores (serial execution) to the maximum number of cores available in the system.
- *Thread affinity*. OpenMP offers two methods of assigning threads to cores: close affinity (assigning threads to contiguous cores) and spread affinity (assigning threads to cores in a round-robin fashion). Given the impact of thread assignment in the memory footprint, specially in NUMA systems, the kernel is executed using both options for every number of cores.

In addition, the size of the problem to execute the candidate kernel should be chosen to ensure that the overhead introduced by OpenMP is negligible.

Finally, with the objective of increasing the accuracy of the kernel's signatures, multiple executions are required to obtain all the counters' values, where each execution measures one set of counters. Consequently, the total number of executions of the candidate kernel can be computed using (1).

$$n\_executions = n\_times * counter\_sets * 2 * n\_cores \quad (1)$$

B. Building the candidate kernel representation

Figure 2 summarizes the second step of our methodology, the aggregation of the collected data in order to form the candidate kernel's representation.

To create this representation, the execution data for each set of hardware performance counters is joined and sorted by the number of threads and their affinity, and then by their repetition number. The candidate kernel representation is built by removing all the information except for the hardware counter values expressed as the ratio counter/cycle. The result of these steps is a sorted vector of signatures for all executions of the candidate kernel.

C. Correlation analysis

The last step of the methodology consists of performing a correlation analysis between the candidate kernel representation and the representation of all patterns in the current *pattern collection*.

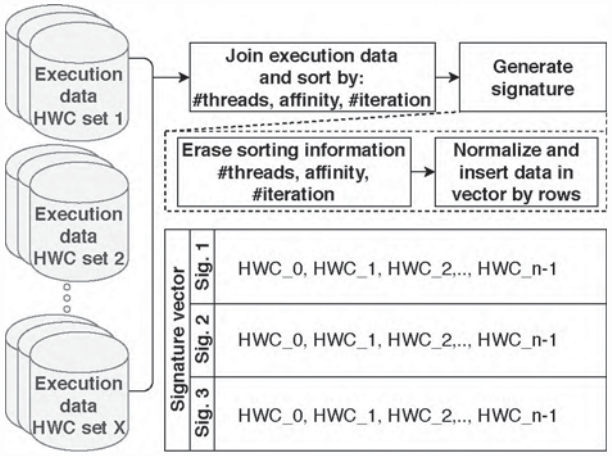

Fig. 2. Methodology to build the candidate kernel representation.

Spearman's rank correlation is based on deviation between two series of data and is more sensitive to data errors and discrepancies, while Kendall's tau is based on concordance (and discordance) of data pairs and is effective for detecting trends. In almost all cases, both of them will lead to the same inferences and so the verification of this serves to increase the robustness of our method.

Consequently, Spearman's rank correlation and Kendall's tau are applied to establish whether the candidate kernel covers a new region of the input space or not. If both methods indicate that the candidate kernel is not highly correlated to any patterns in the current collection and it logically represents a different region, the candidate kernel will become a pattern that will be added to the collection.

Next, we must decide which coefficient thresholds will be used in order to determine if the correlation between the candidate kernel and one of the patterns in the collection can be regarded as high enough to warrant the candidate pattern exclusion. These thresholds may vary depending on the problem at hand, but according to [5] and [8] a correlation coefficient above 0.7 can be considered very high.

Patterns that are not highly correlated would likely require different performance tuning strategies, which indicates that they cover different parts of the input space. Therefore, we adopt a stricter criterion and consider that two patterns are highly correlated if their Spearman's rank exceeds 0.9 (90%); and Kendall's tau 0.8 (80%). Empirically, this combination also worked well for our problem as similar patterns reported correlation values of Spearman's r>0.9 and Kendall's tau>0.8.

The *pattern collection* stores for each pattern the hardware counter values (signatures) of multiple executions using multiple problem (data structures) sizes. These sizes are chosen by examining the available memory levels in the system, and taking into account the number of data structures (vectors or matrices) used by each pattern. As most processors have 4 memory levels (L1, L2, L3 caches, and RAM), four groups of problem sizes are used in order to stress each memory level.

It can be observed that, while the representation of the candidate kernel has been computed on a problem size that has been chosen for minimizing OpenMP overhead (see Section III-A), the signatures of patterns in the *pattern collection* have been computed using several problem sizes chosen in order to stress the available memory levels. Therefore, the correspondence between the problem size for the candidate kernel and the problem size of the patterns in the collection is not a priori known and, as a consequence, the correlation analysis must be performed for every signature of each of the patterns in the collection.

Figure 3 displays an overview of how this process is performed. First, a correlation matrix is built and initialized to 0. This matrix will have one row for each pattern in the collection and four columns to store the Spearman's rank, the Kendall's tau, statistical significance of Spearman's rank (P value), and the problem size.

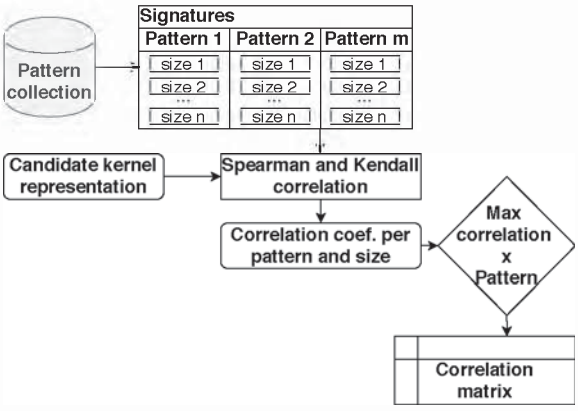

Fig. 3. Process for obtaining the correlation of the candidate kernel against the *pattern collection*.

Next, Kendall's and Spearman's correlation analysis are applied between the candidate kernel representation and all problem sizes for each pattern in the *pattern collection*. This provides both the correlation coefficients and the Spearman's rank statistical significance. Finally, for each pattern, the information associated with the the maximum Spearman's rank (coefficient values, P value, and the corresponding problem size) is stored in the correlation matrix.

Finally, if the candidate kernel is highly correlated with at least one pattern in the collection, i.e., there is at least one row in the correlation matrix with a Spearman's $r>0.9$, Kendall's $tau>0.8$, and a P value<0.05, the kernel is discarded as it does not covers a new portion of the input space. Otherwise, the candidate kernel is considered to cover a new portion of the input space and it is added to the collection as a new pattern. For doing so, the kernel must be executed for all the problem sizes included in the *pattern collection* to obtain the corresponding signatures.

IV. Experimentation

In this section the methodology we propose is applied to incrementally construct a dataset of OpenMP parallel patterns and the ANN described bellow is leveraged to validate the set and demonstrate the importance of using a balanced dataset. Figure 4 summarizes the workflow followed in this experimentation:

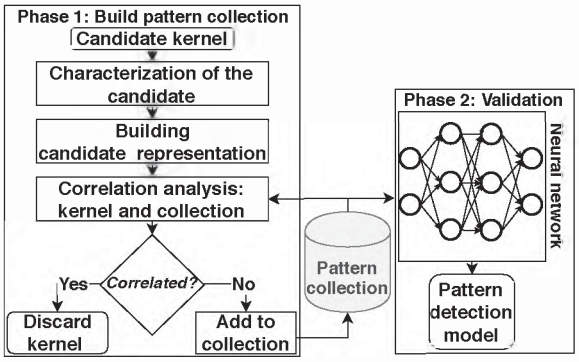

Fig. 4. Summary of the experimentation workflow.

- Phase 1: Building the *pattern collection* applying the steps described in Section III.
- Phase 2: Validating the collection and illustrate the importance of using a balanced dataset using the ANN.

The candidate kernels for building the training set are extracted from two different well-known benchmarks:

- STREAM (Sustainable Memory Bandwidth in High Performance Computers) [9] is a synthetic benchmark composed of simple vector kernels to measure sustainable memory bandwidth.
- POLYBENCH [10] [11] is a collection of benchmarks with multiple kernels. Its version 4 includes 23 different benchmarks divided in different categories (datamining, linear algebra, medley and stencils).

The ANN used for this experimentation and hardware is a Fully-connected, Feed-forward Neural Network and its architecture can be seen in Table I.

TABLE I
Artificial Neural Network Architecture

Layer	Neurons	Inputs	Activation	Weight Constraint	Dropout
Input	N/A	18	N/A	N/A	0%
Hidden 1	18	18*18	SELU	Clip [-10.0, 10.0]	10%
Hidden 2	16	18*16	SELU	Clip [-10.0, 10.0]	10%
Output	8	16*8	Softmax	Clip [-10.0, 10.0]	0%

SELU activations were selected for the hidden layers of the network, which provide several benefits, some are: self-normalizing, cannot die as Rectified Linear Units do, and do not produce vanishing or exploding gradients [12]. The output layer of the network utilizes a Softmax function paired with Categorical Cross-Entropy, allowing the network to perform classification for multiple classes. Therefore, the network outputs a vector of probabilities of a given instance in the dataset belonging to each pattern.

The potential for the network to overfit is counteracted with a probabilistic dropout and constraining the networks' weights [13]. During iterations of training, each neuron in the hidden layers of the network is temporarily removed with a probability of 0.1. Due to the low number of neurons in each network layer, increasing the dropout probability past 0.1 could prevent the model from converging. A constraint was applied to the weights incident to the hidden and output layers by clipping them to the range [-10.0, 10.0]. This helps to regularize the weights and prevents only a small number of them from dominating the network.

The Adam optimizer [14] was selected for training network by stochastic gradient descent with learning rate $\alpha = 0.001$ and exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The selection of these hyperparameters is less significant as the regularization provided by the use of SELUs allows for much higher learning rates and decays while maintaining a relatively smooth convergence rate [12].

All the experiments were conducted on a DELL T7500 with two XEON E5645 processors. Each processor features six multi-threaded cores and four memory levels: three levels of cache (32KB L1 and 256KB L2 for each core, and 12MB L3 shared between all the cores) and 96GB of RAM. In addition, [1] shows that OpenMP parallel regions in this platform can be characterized by the values of 18 hardware counters.

To characterize a candidate kernel on this platform all the combinations (9000 according to (1)) of the following parameters must be executed:

- *Number of threads*: from 1 to 12 threads.
- *Thread affinity*: close and spread.
- *Number of executions*: 75 to attain statistical significance.
- *Number of counter sets*: 5 to cover the 18 counters.

To fully characterize a pattern on this platform, all these combinations are repeated using 43 different *problem sizes* to stress all the memory levels.

A. Building the pattern collection

The first group of candidate kernels is extracted from the STREAM benchmark. It contains four simple OpenMP kernels designed to represent the behavior of vector style applications:

- *Copy*. A simple copy between two vectors: $c[i] = a[i]$.
- *Add*. Addition of two vectors: $c[i] = a[i] + b[i]$.
- *Scale*. Scalar multiplication: $c[i] = scalar * a[i]$.
- *Triad*. Mix of Add and Scale: $c[i] = scalar * a[i] + b[i]$.

As the *pattern collection* is initially empty, it is initialized using the Copy pattern and then the methodology is applied for the remaining candidates. Consequently, Copy must be executed for all combinations of the parameters and problem size values (9000*43 executions), while Add, Scale and Triad must be initially executed only for a significant problem size (9000 executions). MATE [15] [16] is used to acquire the hardware performance counters values for each of these executions and compute the signatures for each kernel.

Once the *pattern collection* has been initialized with all the Copy signatures, and one significant signature has been computed for Add, Scale and Triad, we perform correlation analysis. Table II shows that the values of Spearman's rank and Kendall's tau between the three candidate kernels (Add, Scale and Triad) and the pattern in the collection (Copy) are below the conditions (r>0.9 and tau>0.8) established in the methodology for considering them to be covering the same region of the input space. However, Table II also shows that there is a very strong correlation between Add, Scale and Triad, which is clearly above the threshold. This relationship is observed in Figure 5, which shows that none of the candidate kernels are correlated with the Copy pattern, but also that they are strongly correlated with each other.

TABLE II
SPEARMAN(S) AND KENDALL(K) MAXIMUM CORRELATION COEFFICIENTS FOR STREAM.

	Copy		**Triad**		**Add**		**Scale**	
	S	**K**	**S**	**K**	**S**	**K**	**S**	**K**
Copy			0.82	0.78	0.86	0.82	0.85	0.83
Triad	0.82	0.78			0.99	0.95	0.99	0.93

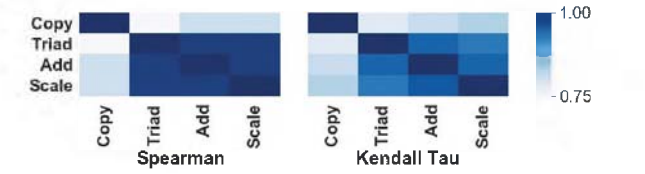

Fig. 5. Spearman and Kendall maximum correlation for STREAM kernels.

As a result, only one of the candidates shall be included in the *pattern collection*. The chosen pattern is Triad because it is the result of the composition between Add and Scale, i.e., it is logically the most general one.

Consequently, after applying the methodology to the kernels extracted from STREAM, the *pattern collection* will contain the following patterns:

- Copy. Pattern abstracting accesses (reads and/or writes) to consecutive memory positions.
- One dimensional group. Pattern abstracting operations involving only one-dimensional vectors.

Next, we extend the *pattern collection* using POLYBENCH, from which we have extracted 29 new candidate kernels. POLYBENCH did not have a parallel version, but we realized that it was simple to implement an OpenMP parallel version of several of the benchmarks stored in the directories *blas*, *kernels* and *stencils*. After executing all the parallel kernels of POLYBENCH and generating their candidate kernel representations (signature), we incrementally detected new patterns and added them to the *pattern collection*. These new detected patterns are described as follows:

- *Reduction*. Pattern abstracting reduction operations, such as adding all the elements of a vector: $total = \sum c[i]$.
- *Stride*. Pattern abstracting accessing elements with a stride: $c[stride \cdot i] = a[stride \cdot i]$.

- *Rows stride*. Pattern abstracting operations involving column-wise traversal of a matrix: $c[i \cdot N][j] = a[i \cdot N][j]$.
- *Matrix x Vector*. Pattern abstracting different matrix-vector operations, such as the matrix-vector multiplication: $A = B \times v$.
- *Matrix x Matrix*. Pattern abstracting different matrix-matrix operations, such as the matrix-matrix multiplication: $C = A \times B$.
- *Stencil*. Pattern abstracting stencil operations, such as: $A[i][j] = A[i-1][j]+A[i+1][j]+A[i][j-1]+A[i][j+1]$

Figure 6 graphically shows the correlation of the kernels extracted from POLYBENCH and the 8 detected patterns. It can be clearly seen that all these kernels are highly correlated to at least one of the patterns in the collection.

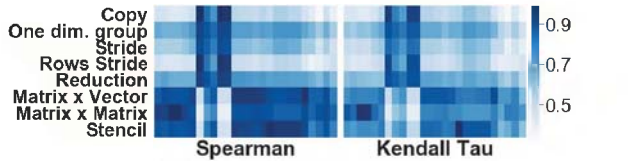

Fig. 6. Spearman rank and Kendall's tau correlation between POLYBENCH discarded kernels and the *pattern collection*.

Summarizing, the resulting dataset obtained from the STREAM and POLYBENCH benchmarks is composed of the following patterns: *Copy*, *One dimensional group*, *Stride*, *Rows stride*, *Reduction*, *Matrix x Vector*, *Matrix x Matrix*, *Stencil*.

B. Validating the pattern collection

In order to validate the collection obtained in the previous subsection, which also will validate the proposed methodology, we have used this collection to train the ANN for producing a pattern classification model.

The pattern dataset is composed of 822504 signatures representing the 8 detected patterns, it is divided into a training set composed of 658003 signatures and a test set composed of 164501 signatures. The network has been trained for 24 epochs using batches with 100 signatures and obtaining a final loss of 0.0301 and accuracy of 0.9893.

Next, a validation set has been built using the signatures of the kernels considered and discarded in the previous section and several new kernels extracted from NAS parallel benchmarks (NPB) [17], which is a set of 5 kernels (IS, EP, CG, MG, and FT) and 3 pseudo-applications (BT, SP, and LU) commonly used to study the performance of parallel supercomputers.

Specifically, we used the following ten OpenMP parallel kernels extracted from the NPB:

- *Add_BT* and *rhs_norm_BT*. These kernels correspond to the *add* and *rhs_norm* BT's functions, respectively.
- *normztox_CG*, *norm_temps_CG*, *rhorr_CG*, *z_alpha_p_CG*, *pr_beta_p_CG*, and *qAp_CG*. Which are regions that have been extracted from different CG's functions.
- *l2norm_LU*. Which corresponds to the *l2norm* LU's function.
- *ssor_LU*. Which is a region extracted from the LU's *ssor* function.

Table III shows the very high accuracy of the trained classification model on the signatures of the kernels extracted from the STREAM and POLYBENCH benchmarks that do not have been chosen for building any of the pattern representations (discarded kernels).

TABLE III
ACCURACY OF THE ANN ON THE KERNELS THAT DO NOT REPRESENT ANY PATTERN IN THE COLLECTION

	Number of kernels	Accuracy
Copy	3	0.93
1D group	3	0.9
Stride	3	0.91
Rows stride	0	-
Reduction	0	-
Matrix x Vector	5	0.9
Matrix x Matrix	5	0.87
Stencil	12	0.99

Table IV shows the results produced by the pattern classification model for the 10 kernels extracted from the NPB. In this case, we are proceeding the other way around because there is no previous correlation analysis that tells us to which pattern is correlated each of these kernels. Consequently, to validate the classification done by the model, we have computed the correlation coefficients of each kernel signature to the patterns in the dataset. Figure 7 shows both correlation coefficients (Spearman's rank and Kendall's tau) between the NPB candidate kernels and the patterns in the dataset. It is clearly seen that the plotted results align with the classification given by the model. This results demonstrates our claim that the ANN can also be used to detect candidate patterns not included yet in the dataset.

TABLE IV
CLASSIFICATION OF THE KERNELS EXTRACTED FROM THE NPB USING THE TRAINED ANN

NAS kernel	Predicted Pattern
Add_BT	One dimensional group (97%)
l2norm_LU	Reduction (88%)
norm_temps_CG	Reduction (100%)
normztox_CG	One dimensional group (100%)
pr_beta_p_CG	One dimensional group (99%)
qAp_CG	Reduction (94%)
rhorr_CG	Reduction (100%)
rhs_norm_BT	Reduction (88%)
ssor_LU	One dimensional group (98%)
z_alpha_p_CG	One dimensional group (84%)

Finally, we have devised an experiment for showing the importance of using a balanced training dataset. It consists of

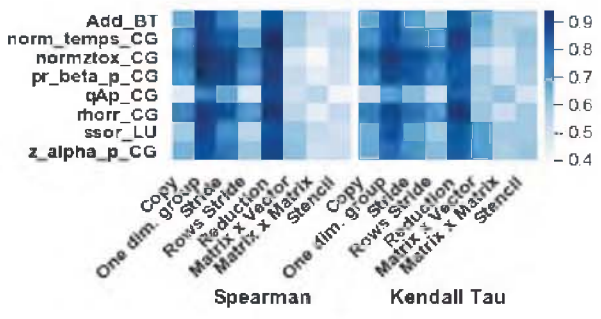

Fig. 7. Correlation coefficients between NPB extracted kernels and the *pattern collection*.

training an ANN model multiple times using an unbalanced dataset, validating the resulting model using some of the kernels extracted from the NPB, and comparing the results to the ones obtained with the balanced dataset.

The unbalanced dataset includes several kernels of the *One dimensional group*, *Stride* and *Stencil* patterns as independent classes and does not includes kernels for the *Matrix x Vector* and *Matrix x Matrix* ones. We use the following kernels:

- Copy.
- Add, Scale, Triad, Stride 2, and Stride 4 (from the One dimensional group).
- Reduction.
- Stride 16 and Stride 64 (from Stride).
- 2PStencil and 2D4PStencil (from Stencil).

Table V shows the most frequent results produced by the ANN models trained using the unbalanced dataset. It can be seen that for the considered kernels that, in contrast with the previous results, the results using the unbalanced dataset are flipping between different possibilities, which for some cases belong to completely different patterns.

TABLE V
PREDICTION GIVEN BY THE ANN MODEL TRAINED WITH THE UNBALANCED SET.

Kernel	Prediction Unbalanced Set
Add_BT	Add (65%), Scale (35%)
norm_temps_CG	Reduction (36%), 2D4PStencil (64%)
pr_beta_p_CG	2PStencil (31%), Triad (68%)
rhorr_CG	Reduction (46%), 2D4PStencil (54%)
rhs_norm_BT	Reduction (77%), 2D4PStencil (15%)
ssor_LU	Triad (32%), 2PStencil (62%)

V. Related Work

Our approach identifies which kernels among different OpenMP regions should be included to reduce redundancy in a dataset. Similar works appear in the literature which focus on classification of parallel regions by leveraging analytical models or machine learning to identify parameters such as the number of threads or scheduling. However, the problem of building a balanced and representative training set is seldom addressed, which could jeopardize the obtained results.

DiscoPoP [18] is a tool designed to locate opportunities for parallelism in sequential programs and give suggestions to the user as to their potential implementations. DiscoPoP identifies small regions of code called computational units (CU), which follow the read-compute-write pattern, and arranges them in a data dependency graph. The dependencies identified by DiscoPoP are used in order to find 4 types of patterns [19] named *DOALL*, *Reduction*, *Task parallelism* and *Pipeline*. For each of them an OpenMP construct is proposed for producing the parallelized version of the input program. In [20] dynamic features generated by DiscoPoP are used in order to train classifiers to classify loops in a sequential program as parallelizable or not. The work in [21] presents a similar approach which uses program dependence graphs and other features to train an ML model that is used to predict the probability of a code regions being parallelizable.

Kernel Tuning Toolkit (KTT) is an autotuning tool for OpenCL and CUDA kernels [22]. KTT uses an automatic search of the tuning space to determine a configuration for one or more kernels with shared tuning parameters. A benchmark redundancy reduction approach is presented in [23]. The main idea is to extract non-overlapping sections of code from benchmarks, characterize them using a set of static and dynamic features, and use clustering to select the subset of representative sections for accelerating system selection.

The approach in [24] focuses on using active-learning instead of empirical evaluation for performance tuning. It uses a particular implementation of regression trees called dynamic trees in order to find an appropriate selection of parameter values inside a defined parameter space. In [25], a compiler based approach is taken to predict the number of threads and scheduling policy of OpenMP applications. This approach uses ANN to predict scalability and Support Vector Machines (SVMs) to select the scheduling policy. Source code, data and runtime features are used to train the two models.

APARF [26] is an adaptative runtime framework to enhance peformance of OpenMP task-based programs. APARF makes use of PAPI to obtain hardware performance events and trains an ANN to find which scheduling scheme should be used to obtain the optimal performance in unseen programs with an average accuracy of 93%. In [27], profiling data is used to extract information about a region and SVMs are used to decide whether a candidate loop should be parallelized or not.

VI. Conclusions and Future Work

We have developed and validated a methodology that allows for building a balanced and representative dataset to be used for training an ML approach. This methodology is focused on determining whether a given OpenMP parallel region pattern covers a portion of the feature space that is not represented by the training set so far. In this proposal, an OpenMP pattern is represented by the signatures of a representative kernel, calculated for various sizes of problems chosen to stress the levels of memory available on the system, and correlation is

used for deciding if a given kernel could be representative of a pattern covering a new portion of the feature space.

The use of the proposed methodology has been illustrated through its application to the set of kernels extracted from two well-known benchmarks (STREAM and POLYBENCH) for a particular architecture. This process has allowed for the construction of a preliminary dataset comprised of the following 8 patterns labels: *Copy*, *One dimensional group*, *Stride*, *Rows stride*, *Reduction*, *Matrix x Vector*, *Matrix x Matrix*, and *Stencil*.

The resulting dataset has been used to train an ANN as a pattern classification model and the results obtained in the validation of this model has been compared to the ones obtained training multiple times the same ANN using an unbalanced dataset. This comparison answers the research questions addressed in this work, demonstrating i) that the proposed methodology allows for systematically generating a balanced dataset, and ii) the significant impact of using a balanced dataset on the classification accuracy.

Our next step will be to extend the methodology for using datasets to train a robust ANN detecting performance problems. Upon reaching this goal, it is conceivable that an ANN could be applied in a Reinforcement Learning context to automatically tune performance parameters such as the optimal number of threads, affinity or scheduling policy to be used in a given region. In conclusion, the dataset and the methodology to construct it described in this work are a step towards an autonomous system for addressing performance issues for OpenMP regions.

REFERENCES

[1] J. Alcaraz, A. Sikora, and E. César, "Hardware counters' space reduction for code region characterization," in *Euro-Par 2019*, ser. Lecture Notes in Computer Science, R. Yahyapour, Ed., vol. 11725. Springer, 2019, pp. 74–86.

[2] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," in *Proceedings of the department of defense HPCMP users group conference*, vol. 710, 1999.

[3] G. H. Nguyen, A. Bouzerdoum, and S. L. Phung, "Learning pattern classification tasks with imbalanced data sets," in *Pattern Recognition*, P.-Y. Yin, Ed. Rijeka: IntechOpen, 2009, ch. 10. [Online]. Available: https://doi.org/10.5772/7544

[4] A. Nath and K. Subbiah, "The role of pertinently diversified and balanced training as well as testing data sets in achieving the true performance of classifiers in predicting the antifreeze proteins," *Neurocomputing*, vol. 272, pp. 294 – 305, 2018.

[5] L. Jäntschi and S.-D. Bolboaca, "Pearson versus spearman, kendall's tau correlation analysis on structure-activity relationships of biologic active compounds," *Leonardo Electronic Journal of Practices and Technologies*, vol. 6, pp. 76–98, 2005.

[6] A. J. Bishara and J. B. Hittner, "Testing the significance of a correlation with nonnormal data: comparison of pearson, spearman, transformation, and resampling approaches." *Psychological methods*, vol. 17, no. 3, p. 399, 2012.

[7] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[8] M. M. Mukaka, "A guide to appropriate use of correlation coefficient in medical research," *Malawi medical journal*, vol. 24, no. 3, pp. 69–71, 2012.

[9] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," *Link: www.cs.virginia.edu/stream/*, 1995.

[10] T. Yuki, "Understanding polybench/c 3.2 kernels," in *International workshop on Polyhedral Compilation Techniques (IMPACT)*, 2014, pp. 1–5.

[11] T. Yuki and L.-N. Pouchet, "Polybench 4.0," 2015, accessed: April 21 2020. [Online]. Available: https://www.cs.colostate.edu/AlphaZsvn/Development/trunk/mde/edu.csu.melange.alphaz.polybench/polybench-alpha-4.0/polybench.pdf

[12] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] A. Martínez, A. Sikora, E. César, and J. Sorribes, "Elastic: A large scale dynamic tuning environment," *Scientific Programming*, vol. 22, 1970.

[16] A. Sikora (Morajko), P. Caymes-Scutari, T. Margalef, and E. Luque, "Mate: Monitoring, analysis and tuning environment for parallel/distributed applications," *Concurrency and Computation: Practice and Experience*, vol. 19, pp. 1517 – 1531, 08 2007.

[17] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The nas parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[18] Z. Li, A. Jannesari, and F. Wolf, "Discovery of potential parallelism in sequential programs," in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 1004–1013.

[19] M. Norouzi, F. Wolf, and A. Jannesari, "Automatic construct selection and variable classification in openmp," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 330–341.

[20] D. Fried, Z. Li, A. Jannesari, and F. Wolf, "Predicting parallelization of sequential programs using supervised learning," in *2013 12th International Conference on Machine Learning and Applications*, vol. 2. IEEE, 2013, pp. 72–77.

[21] A. Maramzin, C. Vasiladiotis, R. C. Lozano, M. Cole, and B. Franke, ""it looks like you're writing a parallel loop": a machine learning based parallelization assistant," in *Proceedings of the 6th ACM SIGPLAN International Workshop on AI-Inspired and Empirical Methods for Software Engineering on Parallel Computing Systems*, 2019, pp. 1–10.

[22] J. Filipovič, F. Petrovič, and S. Benkner, "Autotuning of opencl kernels with global optimizations," in *Proceedings of the 1st Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems*, ser. ANDARE '17, New York, NY, USA, 2017.

[23] P. de Oliveira Castro, Y. Kashnikov, C. Akel, M. Popov, and W. Jalby, "Fine-grained benchmark subsetting for system selection," in *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '14, New York, NY, USA, 2014, p. 132–142.

[24] P. Balaprakash, R. Gramacy, and S. Wild, "Active-learning-based surrogate models for empirical performance tuning," *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, pp. 1–8, 09 2013.

[25] Z. Wang and M. F. O'Boyle, "Mapping parallelism to multi-cores: a machine learning based approach," in *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2009, pp. 75–84.

[26] A. Qawasmeh, A. M. Malik, and B. M. Chapman, "Adaptive openmp task scheduling using runtime apis and machine learning," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 889–895.

[27] G. Tournavitis, Z. Wang, B. Franke, and M. F. O'Boyle, "Towards a holistic approach to auto-parallelization: integrating profile-driven parallelism detection and machine-learning based mapping," *ACM Sigplan notices*, vol. 44, no. 6, pp. 177–187, 2009.