Optimizing Program Performance via Similarity, Using a Feature-agnostic Approach

Rosario Cammarota, Laleh Aghababaie Beni
Alexandru Nicolau, and Alexander V. Veidenbaum

Department of Computer Science, University of California Irvine, Irvine, USA
{rosario.c,laghabab,nicolau,alexv}@ics.uci.edu

Abstract. This work proposes a new technique for performance evaluation to predict performance of parallel programs across diverse and complex systems. In this work the term system is comprehensive of the hardware organization, the development and execution environment.
The proposed technique considers the collection of completion times for some pairs (*program*, *system*) and constructs an empirical model that learns to predict performance of unknown pairs (*program*, *system*). This approach is feature-agnostic because it does not involve previous knowledge of program and/or system characteristics (features) to predict performance.
Experimental results conducted with a large number of serial and parallel benchmark suites, including SPEC CPU2006, SPEC OMP2012, and systems show that the proposed technique is equally applicable to be employed in several compelling performance evaluation studies, including characterization, comparison and tuning of hardware configurations, compilers, run-time environments or any combination thereof.

Keywords: Program Characterization, Feature-agnostic, Cluster Analysis, Empirical Performance Modeling, Program Optimization

1 Introduction

Over the past decade there has been an exponential growth in computer performance [1] that quickly led to more sophisticate and diverse software and computing platforms (e.g., heterogeneous multi-core platforms [2], parallel browsers [3]). The cost of software development and hardware design too increases and creates the need for evaluating performance of proposed software and system changes before the actual implementation and deployment begin.

However, given the increasing complexity of modern micro-architectures [2], software [4, 5], development and execution environments [6], performance of a program on new systems (specularly, performance that a system delivers to new programs) is difficult to predict. Constructing a comprehensive model that includes all the possible aspects featuring software and computing platform is practically limited by the cost of feature retrieval compared with the performance goal to reach. For example, while having negligible run-time overhead, collecting a large number of hardware performance counters is not possible at

once - it requires multiple runs of a software as only a limited number of hardware performance counters can be collected in one run, e.g., up to eight in modern Intel microprocessors [7]. Instrumentation based retrieval and simulation also incur in significant run-time overhead [8].

As a result, prior research (*a*) mainly focuses on the characterization and empirical performance modeling of serial programs and (*b*) limits its attention to a number of program/system features that are tied to a specific performance study (e.g., system procurement [9, 10], tuning of compiler heuristics [11, 12], task-to-core allocation tuning for heterogeneous and homogeneous multi-cores [13, 14], selection of the number of cores and parallel scheduling algorithm [15], design space exploration [16–18]) and do not generalize to other performance studies. These techniques can be categorized as feature-aware because the features chosen to characterize the specific performance study are extracted from either system and program properties, e.g., hardware performance counters, or program-inherent properties, e.g., instructions mix, working set size [19], or system design properties, e.g., number, type, size and organization of hardware components [16, 18, 20].

In this work we present a new feature-agnostic technique for performance modeling, prediction and, more important, we present the application of the proposed technique to several cases of serial and parallel program performance optimization. The type of characterization presented in this work differs from the characterizations introduced in previous research. In fact, in this work only the knowledge of completion times for some pairs (*program*, *system*) is leveraged to predict performance for unknown pairs (*program*, *system*). The characterization of programs and systems is embedded in series of known performance of a programs on certain systems and known performance that some systems give to some programs. Because of the above, the proposed technique is said to be feature-agnostic because it does not involve a knowledge of programs (e.g., the number of instructions of a certain type, the memory footprint) and systems (e.g., the memory hierarchy organization, the operating frequency) characteristics.

Using the information available, first a technique based on micro-array visualization [21] is proposed to highlight the presence of coherent patterns of similarity between programs and systems. Programs exhibiting similar series of performance on different systems are clustered together using hierarchical clustering [22]. Likewise, systems exhibiting similar series of performance on different programs are clustered together. A heat-map is constructed to provide a joint visualization of coherent patterns, i.e., areas on the map on which (*program*, *system*) performance are nearly-equal. The presence of such patterns highlights opportunities to predict program performance by similarity. For example, two (or more) programs that attain nearly-equal performance on a number of systems are deemed to be similar under a certain similarity metric. These programs are also likely to attain nearly-identical performance on a new system. Hence, the knowledge of performance for some of these programs on a

new system is predictive of performance of a similar, unseen programs on this system.

Performance modeling and prediction happens as follows: two prediction models are constructed and their outcomes are combined to predict performance for unknown pairs (*program*, *system*). The first model aims to predict performance that a new system provides to a program, given the available performance that other systems provided to the program. The second model aims to predict performance of a program on a new system given the available performance of other programs on the system. Performance prediction for unknown pairs (*program*, *system*) is obtained as the weighted average of the outcomes of the two predictors above. The importance of combining the two predictions above is to improve prediction accuracy compared with the accuracy of the individual model. To construct prediction models, we use, evaluate and compare machine learning algorithms for regression, e.g., Linear Regression (`LR`) [23] and Support Vector Regression (`SVR`) [24].

Program optimization occurs in the form of selecting the combination of algorithmic optimization, compilation, execution environment and hardware design that maximizes performance, i.e., minimizes completion time, of a program of interest. A procedure based on k-fold cross-validation [25] is proposed to validate the proposed prediction technique.

To the best of our knowledge this is the first work that considers a feature-agnostic, practical technique for program and system characterization, performance modeling and prediction. In particular, this work makes the following contributions: (*i*) It provides a new, practical and generally applicable technique for cross-system performance modeling and prediction. Performance prediction relies on a fairly general and simple to retrieve program characterization, i.e., performance of some pairs (*program*, *system*) - which usually is available in industry and research settings; (*ii*) The application of the proposed technique to several practical performance studies is shown, e.g., the cases of system selection, compiler and run-time settings selection are illustrated; (*iii*) It is also shown that for the sake of several performance studies, the problem of program characterization and specifically that of feature selection can be moved to the problem of selecting appropriate simpler programs, e.g., programs from different application domains. Performance modeling and prediction happens by similarity between series of completion time of a program on different systems and of a system on different program; (*iv*) It is shown that the proposed characterization is not only applicable across different systems, but also holds for both serial and parallel programs.

The rest of the paper is organized as follows: Basic concepts and definitions that are adopted in this work are introduced in Section 2. Section 3 discusses previous research; The characterization and performance modeling technique is described in Section 4. Section 4 also discusses on the evaluation methodology developed to validate the proposed performance modeling technique; The experimental evaluation and results are presented and discussed Section 5. Key highlights and future developments are discussed in Section 6.

2 Basic Concepts and Definitions

In this work a property of a program, e.g., instruction mix, working set size, and/or of a system, e.g., memory hierarchy configuration, average number of cycles to execute each instruction in the instruction set architecture (`ISA`), is a *feature* of the program and/or the system. A set of features is a *signature*. This work focuses on signatures whose features assume real number values, techniques to analyze patterns (similarity) occurring between features, empirical performance modeling that associate performance to features.

A signature is a n-dimensional vector of real numbers $\mathbf{s} = [f_1, f_2, \cdots, f_n]$ whose elements can be homogeneous (they measure the same quantity, e.g., a series of energy consumptions or completion time) or heterogeneous (they measure different quantities, e.g., number of cache misses, cost of communication). A feature is agnostic of program and/or system properties if its value cannot be directly associated to any property of a program or of a system. Examples of such type of features are total completion time, total energy consumption etc. Otherwise a feature is aware of program and/or system characteristics. Techniques to model program and system performance for program optimization are *feature-agnostic* when the signature adopted by the specific technique is composed of features that only latently account for program and/or system properties. Otherwise the approach is said to be *feature-aware*.

Given a set of signatures, a similarity measure [26] provides a way of measuring the degree of similarity between two programs (or systems). The comparison of two signatures under a certain similarity measure makes program characterization (i.e., discovery of groups of signatures with similar properties) possible. More in general, the use of cluster analysis (as explained in the next Section) makes possible to partition a set of signatures into smaller groups of signatures with similar features [22]. Because the type of characterization proposed in this work is feature-agnostic and the elements of a signature are completion times, the similarity measure adopted in this work is the Euclidean distance.

3 Related Work

For a specific performance study, deciding whether to adopt a feature-aware or -agnostic characterization has pros and cons. Feature-aware characterizations appear in many common practices in program characterization and tuning [19, 27, 28], algorithmic optimization [29], various intelligent forms of profile-guided optimization [11, 12] and design space exploration [16, 18]. While feature-aware characterizations provide information to interpret the behavior of a program and/or system, feature extraction comes at the cost of iterating multiple program runs or program/compiler instrumentation and/or simulation. In addition to the above, the amount of features amongst which one can choose is enormous. As a result, only a small subset of features are usually considered at once and the selection of features is usually subjective and/or time/cost constrained. While statistical analysis techniques can be employed to select essential features from a group of features, the selection of the initial group of features remains subjective.

On the contrary, feature-agnostic characterizations have been used to analyze the coverage of programs within a benchmark suite [30], for hardware design rating [9, 31] and compiler aided program optimization [32]. However, in prior work feature-agnostic characterizations have been employed in conjunction with serial programs and only one of the following signatures was used at a time: program performance across systems (*program signature*); performance of multiple programs on a system (*system signature*). Differently from previous research, not only this work covers serial program, but also considers and focuses on parallel program optimization. Furthermore, this work combines the use of program and system signature, which enables (*i*) the construction of very accurate performance models and (*ii*) the application of the proposed technique to different program optimization scenarios, as illustrated in Section 5.

4 Learning to optimize programs by similarity

This section presents a unified approach to program and system characterization and a technique to construct effective performance models based on such characterization. The performance characterization and modeling technique presented in this work assumes the knowledge of performance values (*completion times*) for some pairs (*program*, *system*). These values are organized in a $m \times n$ matrix $\mathcal{M}$ such that each element $m_{i,j}$ of $\mathcal{M}$ represents performance of the program i on the system j.

Rows in the matrix $\mathcal{M}$ can be interpreted as program signatures, i.e., $\pi_i = [m_{i,1}, m_{i,2}, \cdots, m_{i,n}]$. Likewise, columns in the matrix $\mathcal{M}$ can be interpreted as system signatures $\sigma_j = [m_{1,j}, m_{2,j}, \cdots, m_{m,j}]$. Therefore, a program signature is the series of performance values that a program attains on different system configurations, whereas a system signature is the series of performance values that a system configuration delivers to a set of programs.

The matrix $\mathcal{M}$ is in general sparse because performance for several entries corresponding to pairs (*program*, *system*) may be missing. Two cases are considered in this work: (*i*) The first case in that $\mathcal{M}$ is dense, i.e., all the entries are known. In this case we present a technique to visualize patterns of similarity between programs and systems; (*ii*) The second case in that the sparsity of $\mathcal{M}$ is limited to just one program of interest, whose performance is unknown on a number of systems. Hence, the program signature for this program is as follows: $\imath_i = [m_{i,1}, m_{i,2}, \cdots, ?, ?, ?, \cdots, ?]$, where the question marks indicate unknown elements in the signature.

The performance modeling and prediction technique presented in this work aims to predicts performance of the program of interest on the unknown systems by similarity, i.e., with respect to performance attained by similar programs on the unknown systems. In particular, in this work, predicting by similarity means that performance prediction is based on a combination of the following two aspects in the matrix $\mathcal{M}$: (*i*) Performance that the program of interest attained on known systems; (*ii*) and Performance that other programs attained on a new system.

4.1 Program and system similarity

The Euclidean distance is adopted in this work as similarity measure [26]. The Euclidean distance is an appropriate similarity measure as the goal of this study is to discover similarity patterns where programs attain nearly-equal performance on different system configurations and/or system configurations deliver nearly-equal performance to different programs.

For either program or system signatures, similarity analysis on a set of signatures is assessed using average-linkage hierarchical clustering [22, 26]. Hierarchical clustering organizes rows (program signatures) of $\mathcal{M}$ into a tree using the following procedure: First, for any set of m signatures, an upper-diagonal (similarity) matrix is computed by using the Euclidean distance between the signatures; Second, the similarity matrix is visited to identify the highest similarity value, i.e., the lowest distance that connects the most similar pair of signatures. A node is created from joining these two signature, and a signature expression profile is computed for the node by averaging observation for the joined elements; Third, the similarity matrix is updated with this new node replacing the two joined elements. This process is repeated $m - 1$ times until only a single element remains. Likewise, the process above is repeated for the columns (system signatures) in $\mathcal{M}$.

Given the similarity trees, the rows and columns in $\mathcal{M}$ are re-organized. Rows are permuted such that similar program signatures are adjacent. Likewise, columns are permuted such that similar system configurations are adjacent. A clustergram visualization [21] associates a heat-map to the permuted version of $\mathcal{M}$. The heat-map is composed of rectangular tiles arranged in a matrix shape where the position of each tile corresponds to the position of an element in the permuted $\mathcal{M}$. Each tile is associated to a color corresponding to the value of the element in the permuted relatively to the average value of the elements $\mathcal{M}$. Values corresponding or close to the average value are colored in black or dark shades of either green or red. Values in the matrix above the average are colored with shades of red - the higher the value is above the average, the lighter red is associated to the corresponding tile. Values in the matrix below the average are colored with shades of green - the lower the value is below the average, the lighter green is associated to the corresponding tile. Finally, the similarity trees are appended to the margins of the heat-map to compose the clustergram - refer to Figure 1. The usefulness of such a representation is that coherent patterns are represented by patches of the same gradient of colors on the heat-map. The formation of such patches is induced by the similarity structure in the signatures and may indicate a functional relationship among system signatures and programs.

4.2 Performance Modeling

In this work performance modeling relies on known machine learning algorithms for regression to predict performance for unknown pairs $(program,\ system)$.

Model Components - Let us assume for a moment that the signature of a program of interest contains only a single missing entry,

$$\pi_i = [m_{i,1}, m_{i,2}, \cdots, m_{i,j-1}, ?, m_{i,j+1}, \cdots, m_{i,n}]$$

The first step of the proposed modeling technique is to construct a model to predict performance of the system at column j for the program i from the knowledge of performance of the program i on the other systems. Hence, from $\mathcal{M}$, a regression model is constructed to learn the following map $\Sigma : \mathbf{f} \rightarrow p_i$, where $\mathbf{f}$ is the feature vector of performance of a certain program on the systems $1, 2, \cdots, j-1, j+1, \cdots, n$ and its outcome is performance that a program i attains on the system j, i.e., p_j.

The missing entry in π_i is also missing in the system signature of the system i, i.e.,

$$\sigma_j = [m_{1,j}, m_{2,j}, \cdots, m_{i-1,j}, ?, m_{i+1,j}, \cdots, m_{m,j}]$$

Hence, the second step of the proposed modeling technique is to construct a model able to predict performance of the program at the row i from the knowledge of performance of other programs in the column j. Hence, from $\mathcal{M}$, another regression model is constructed to learn the following map $\Pi : \mathbf{g} \rightarrow p_j$, where $\mathbf{g}$ is the vector feature of performance of that a certain system delivers to the programs $1, 2, \cdots, i-1, i+1, \cdots, m$ and its outcome is performance that the system j with deliver to a program i, i.e., p_i.

Combined Model - A machine learning algorithm $\mathcal{A}$ is trained in the two cases above to construct two models for the maps Σ and Π. These models are referred as $\hat{F}$ and $\hat{G}$. Prediction for the missing entry in position (i, j) in the dataset $\mathcal{M}$ is performed by combining the following quantities

$$\hat{p_j} = \hat{\Sigma}\left([m_{i,1}, m_{i,2}, \cdots, m_{i,j-1}, m_{i,j+1}, \cdots, m_{i,n}]\right)$$

and

$$\hat{p_i} = \hat{\Pi}\left([m_{1,j}, m_{2,j}, \cdots, m_{i-1,j}, m_{i+1,j}, \cdots, m_{m,j}]\right)$$

It is clear now that $\hat{p_j}$ and $\hat{p_i}$ attempt to predict the same quantity, but from two different perspectives. The combined prediction is obtained using a weighted average of the quantities above, i.e.,

$$\hat{m}_{i,j} = w_j \times \hat{p_j} + w_i \times \hat{p_i}$$

In this work, the Equal Weight Average (`EWA`), i.e., the arithmetic average of the two predictions is taken. Other averaging techniques are possible and a survey of such techniques is in [33].

In this work it is always assumed that the number of missing entries is much smaller than the number of entries in the dataset $\mathcal{M}$ - several entries are available from the history of previous tests. This assumption is likely to be satisfied in practice because testing for performance of a software for different system configurations is a daily routine in industry. Hence, in the case of missing entries, the procedure above is repeated for each missing entry.

Validation Metrics - For an incomplete program signature, i.e.,

$$\pi_i = [m_{i,1}, m_{i,2}, \cdots, ?, ?, ?, \cdots, ?]$$

the modeling technique illustrated in the previous section constructs the following signature $\hat{\pi}_i = [m_{i,1}, m_{i,2}, \cdots, \hat{m}_{i,n-k}, \hat{m}_{i,n-k+1}, \hat{m}_{i,n-k+2}, \cdots \hat{m}_{i,n}]$, where $\hat{m}_{i,j}$ represent performance predictions for the program of interest on the systems $n-k, n-k+1, \cdots, n$, and $k+1$ is the number of unknown performance values.

Given the true performance values that the program of interest attains on the systems $n-k, n-k+1, \cdots, n$, i.e., $m_{i,n-k}, m_{i,n-k+1}, m_{i,n-k+2}, \cdots m_{i,n}$, and the predicted values, i.e., $\hat{m}_{i,n-k}, \hat{m}_{i,n-k+1}, \hat{m}_{i,n-k+2}, \cdots \hat{m}_{i,n}$ the evaluation of the prediction accuracy and the quality of the predicted values is determined in terms of Minimum Absolute Error (`MAE`) - which is defined as the sum of the pairwise absolute differences of components of π_i and $\hat{\pi}_i$ divided by the number of systems n. This metric measures the accuracy in terms of error and error magnitude in the predictions.

In addition, the discrepancy ϵ between the performance delivered to the program of interest by the optimal system and the average performance delivered by the predicted optimal systems in R repetitions of predictions for k missing systems is evaluated. This metric is important to assess the goodness of the proposed modeling technique at selecting an arbitrary system configurations - ultimately, program optimization happens via the selection of the system configuration that minimizes completion time.

Model validation procedure - Given a dense dataset $\mathcal{M}$, the following procedure based on cross-validation [25] is proposed. For each program in the set of programs the dataset is split such as a number of entries $1 \leq k \leq n-1$ is chosen at random and taken out from $\mathcal{M}$. For each missing entry a model constructed as described in Section 4.2 is build and performance for the missing entry is predicted. The random selection of k entries is repeated for a large number of times R compared to the number of entries, such that, a good coverage of the prediction ability of the proposed technique for arbitrarily missing subset of k values is evaluate. Hence, for each k, the average `MAE` and ϵ are reported.

5 Experiments

This section evaluates the proposed technique and illustrates its application to program optimization. Program optimization is the selection of a system configuration that makes program performance satisfactory (or, ideally, minimizes completion time). Even though a variation in program performance is driven by program attribute changes, e.g., a change in compiler settings, in the context of the proposed technique attribute changes represent latent information that is hidden in $\mathcal{M}$ and summarized by series of performance values.

Dataset name	n. of programs x n. of systems
CINT2006	13 x 4308
CFP2006	18 x 4250
OMP2001	10 x 401
OMP2012	10 x15
NAS-ICC	10 x 40

Table 1. Dataset descriptions.

5.1 Datasets

The datasets considered in this Section and their sizes - the number of pairs $(program, system)$ - are illustrated in Table 1. The first four datasets, i.e., `CINT2006` [34], `CFP2006` [34], `OMP2001` [35], `OMP2012` [36], are obtained from the past decade - from January 2001 to December 2012 - of SPEC benchmark results - that is publicly available from SPEC website (http://www.spec.org). Programs in the SPEC benchmarks synthesize real life applications from different application domains - i.e., that are developed with different goals and software development constraints. In particular, programs in `SPEC CPU2006` are intended to exercise system's processor, memory subsystem and compiler. Programs in `SPEC OMP2001` and `SPEC OMP2012` benchmarks are intended to measure performance of shared memory multi-processor systems and heavily exercise the memory subsystem using parallel programs that are compliant with the OpenMP v2.x and OpenMP v3.x (http://www.openmp.org) specifications respectively.

In terms of system configurations, the variety of micro-architectures (e.g., UltraSparcIII, Intel Itanium, R12000 processors), and compilers (e.g., Sun, HP), is richer in the case of shared-memory multiprocessors evaluation than it is for single processor (on which most of the previous modeling techniques focus), where most benchmark records refer to Intel architectures and compilers ($\approx 91\%$ of the total records). Vice versa, UNIX/Linux operating system (Linux $\approx 48\%$, Sun $\approx 24\%$) is the choice for multi-processors evaluation. Linux, Windows and Solaris appear in single processor evaluations.

The last dataset concerns performance of the OpenMP version of the Nasa Parallel Benchmarks [37] subject to changes in compiler settings - inlining settings and AVX vectorization [38] in the `Intel ICC` compiler - and number of threads. The target architecture is Intel Sandy Bridge.

Each dataset in Table 1 is log-transformed, i.e., the natural logarithm of each entry in the dataset is taken.

5.2 Similarity analysis

This section illustrates clustergrams for some of the datasets introduced in the previous Section. While a detailed analysis of all the families (clusters) that are formed using the clustering procedure illustrated in Section 4.1 is out of the scope of this paper, a comparative analysis of small clusters (each cluster contains ≈ 16 systems) from the red and the green areas of the clustergrams is briefly conducted.

A first cluster is extracted from the red patch (that indicates completion times above the average) on the right hand side of Figure 1. System configurations in

this cluster are Intel-based and belong to the families Intel Pentium, Core and Xeon M. A second cluster is extracted from the green patch (completion times below the average) toward the right hand side of Figure 1. System configurations in this cluster are Intel-based and belong to the families Xeon E and X. Clusters group together system configurations based on micro-architectures in the same entry level. In the case of `CFP2006` the clustergram can be roughly divided in six areas - refer to Figure 2. Similarly to the case of `CINT2006`, a comparative analysis of small clusters (each cluster contains ≈ 20 systems) from the red and the green areas is briefly conducted. A first cluster is extracted from the red patch toward the right hand side of Figure 2. System configurations in this cluster are Intel-based and belong to the families Xeon E and X. A second cluster is extracted from the green patch in the center of Figure 2. System configurations in this cluster belong to the families Core-i3E/EV2 and Pentium G. Even in this case clusters group together system configurations based on micro-architectures in the same entry level, however, the comparison of these clusters among `CINT2006` and `CFP2006` shows that system configurations based on architectures whose performance are above the average for `SPEC CINT`, correspond to system configurations whose performance is below the average for `SPEC CFP`. At a larger granularity - larger clusters - differences between clusters become noticeable according to compilers' type, version and settings as well as system library levels - e.g., included as a part of different Linux distributions.

The clustergram for `SPEC OMP2001` is roughly divided in three areas of system configurations. As in the case of `SPEC CPU`, two small clusters (each cluster composed of ≈ 20 systems) from the red and the green areas are briefly analyzed. System configurations in a red cluster are based on micro-architecture families powered by Intel Xeon X. System configurations in the green cluster belong to families powered by Intel Itanium 2, R12000 and Ultra Sparc III. Limited to the records retrieved from SPEC website, the information from the two clusters above indicates that system configurations based on Intel Itanium 2 and HP compilers deliver performance above the average.

5.3 Program Optimization via Similarity

The application of the proposed technique to program optimization concerns the utilization of performance predictions to predict system configurations that minimize the completion time of a given program of interest. System selection corresponds to hardware and compiler settings selection in the cases of `SPEC CPU2006`; it corresponds to hardware, compiler and run-time environment settings in the cases of `SPEC OMP2001`, `SPEC OMP2012` and `NPB`. We use and compare Linear Regression and Support Vector Regression [24] to construct the model components Π and Σ. The combined model is indicated as (Π, Σ). Predictions results are averaged over 500 repetitions of randomly picking k system configurations from a program signature, for each program. Experimental results compare the proposed technique against a random selection of one out of the predicted system configurations.

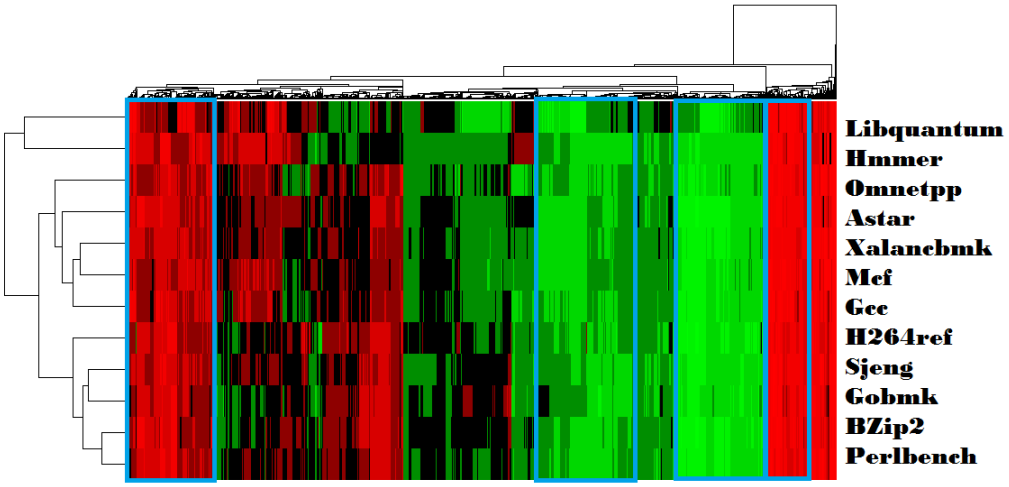

Fig. 1. Feature-agnostic Characterization of `CINT2006`

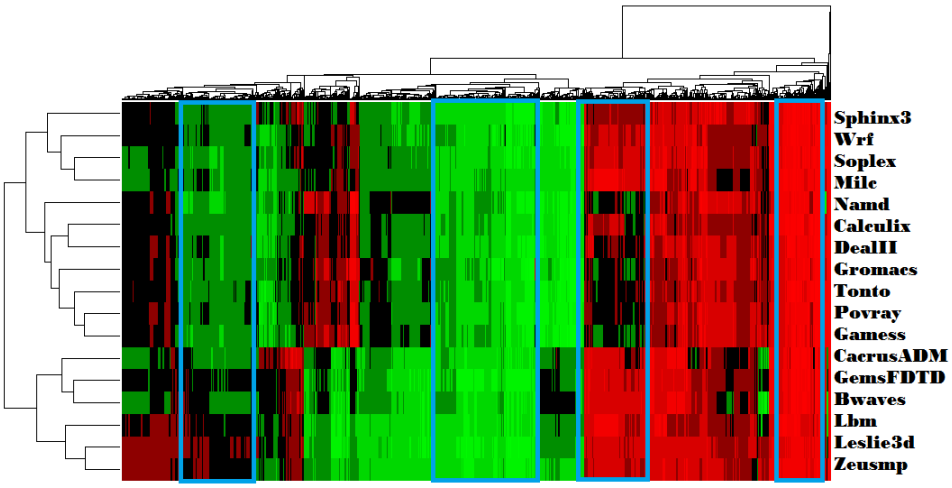

Fig. 2. Feature-agnostic Characterization of `CFP2006`

Table 2 illustrates `MAE` and ϵ for `SPEC CINT2006`. Table 3 illustrates `MAE` and ϵ for `SPEC CFP2006`. In both cases the hybrid model reduces the discrepancy of one order of magnitude - this enforces the concept that both the information about programs and systems in a feature-agnostic characterization are important to construct effective predictors. System configuration predictions with the hybrid model $(\Pi,\ \Sigma)_{SVR}$ are almost perfect, i.e., performance prediction is less than 1% from optimal performance.

	MAE					ϵ			
Missing Item (k)	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$	RND	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$
1	1.00	0.43	0.36	0.20	2.52	0.0470	0.0086	0.0076	0.0008
2	1.06	0.53	0.34	0.20	1.82	0.0887	0.0179	0.0139	0.0015
3	1.05	0.54	0.35	0.20	2.05	0.1207	0.0307	0.0232	0.0020
4	1.05	0.57	0.35	0.20	2.19	0.1796	0.0411	0.0216	0.0027
5	0.98	0.53	0.35	0.20	2.19	0.1923	0.0538	0.0271	0.0030
6	1.13	0.54	0.35	0.20	2.04	0.2224	0.0640	0.0377	0.0036

Table 2. MAE and ϵ for CINT2006

	MAE					ϵ			
Missing Item (k)	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$	RND	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$
1	1.43	0.75	0.26	0.15	1.40	0.1330	0.0594	0.0463	0.0006
2	1.60	0.82	0.26	0.15	1.36	0.2211	0.1101	0.0521	0.0018
3	1.72	0.87	0.25	0.15	1.28	0.3099	0.1574	0.0076	0.0031
4	1.87	0.95	0.26	0.15	1.26	0.3341	0.2024	0.0541	0.0033
5	2.05	1.11	0.25	0.15	1.41	0.4061	0.2434	0.0604	0.0044
6	2.02	1.07	0.25	0.15	1.27	0.3446	0.1557	0.0171	0.0054

Table 3. MAE and ϵ for CFP2006

	MAE					ϵ			
Missing Item (k)	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$	RND	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$
1	1.38	0.76	0.37	0.25	2.19	0.0446	0.0012	0.0020	0.0006
2	1.39	0.76	0.37	0.25	2.19	0.0843	0.0027	0.0047	0.0009
3	1.41	0.74	0.37	0.25	2.19	0.1303	0.0040	0.0073	0.0017
4	1.42	0.75	0.36	0.25	2.19	0.1647	0.0070	0.0101	0.0033
5	1.42	0.77	0.36	0.25	2.21	0.2214	0.0077	0.0115	0.0026
6	1.43	0.77	0.36	0.25	2.20	0.2596	0.0102	0.0155	0.0028

Table 4. MAE and ϵ for OMP2001

	MAE					ϵ			
Missing Item (k)	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$	RND	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$
1	0.16	0.44	0.18	0.28	0.82	0.0000	0.0269	0.0000	0.0000
2	0.17	1.76	0.18	0.30	1.54	0.0000	0.1038	0.0000	0.0099
3	0.17	1.63	0.18	0.31	1.52	0.0000	0.1023	0.0000	0.0161
4	0.17	0.61	0.18	0.35	1.28	0.0000	0.1265	0.0000	0.0391
5	0.18	8.77	0.19	0.37	1.75	0.0002	0.1240	0.0000	0.0453
6	0.19	0.59	0.21	0.46	1.74	0.0015	0.0935	0.0000	0.0416

Table 5. MAE and ϵ for OMP2012

Table 4 illustrates MAE and ϵ for SPEC OMP2001. Table 5 illustrates MAE and ϵ for SPEC OMP2012. As in the case of serial benchmarks, hybrid models reduce the discrepancy of one order of magnitude. However, both hybrid and model based on programs signatures are suitable for the purposed of system selection. System configuration predictions are almost perfect, i.e., performance prediction is less than 1% from optimal performance.

Table 6 illustrates and compare average MAE and ϵ, where the average is taken across programs.

Experiments for the last dataset concern the selection of combinations of compiler and run-time settings that, in addition to the baseline optimization level O3, enable/disable vectorization, i.e., -xavx, assigns an inlining-level

	MAE					ϵ			
Missing Item (k)	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$	RND	Π_{LM}	$(\Pi,\ \Sigma)_{LM}$	Π_{SVR}	$(\Pi,\ \Sigma)_{SVR}$
1	4.68	3.34	0.79	0.40	3.75	0.12	0.09	0.05	0.04
2	3.84	1.87	0.79	0.40	4.30	0.16	0.12	0.06	0.04
3	4.66	1.77	1.21	0.62	3.42	0.17	0.12	0.06	0.04
4	6.57	1.78	0.81	0.41	3.38	0.16	0.13	0.06	0.04
5	2.83	1.59	0.79	0.41	3.32	0.16	0.13	0.06	0.05
6	3.33	1.36	0.96	0.49	3.41	0.15	0.12	0.06	0.05

Table 6. MAE and ϵ for NAS-ICC

and number of parallel threads, i.e., to configure the OpenMP run-time environment. The compiler and the architecture under attention are the Intel ICC v13.1 compiler and Intel Ivy Bridge Core i7-3632QM respectively. In this case, the hybrid model $(\Pi,\ \Sigma)_{SVR}$ provides superior performance in terms of predicting compiler and run-time settings with an average prediction error that is at most 5% from the optimal selection.

6 Conclusion

This work proposed a new feature-agnostinc technique for program and system characterization, performance modeling, prediction and its application to program optimization. The characterization approach to performance modeling considers the collection of completion times for some pairs $(program,\ system)$. Thereby, the proposed technique is practical, because it does not require feature selection, neither does it incur in overheads due to feature retrieval. Because of the above, the proposed technique is suitable to be applied in industry settings to reduce engineering effort for optimizing software. Such an effort involves to find rapid solutions to performance studies, involving the discovery of complex systems settings, that the proposed technique can effectively address.

Experimental results show that the proposed modeling technique equally applicable to be employed in several compelling performance studies, including characterization, comparison and tuning of hardware configurations, compilers, run-time environments or any combination thereof and for both serial and parallel programs.

Acknowledgements

This work was partially supported by the NSF grant number CCF-1249449, the NSF Variability Expedition Grant number CCF-1029783 and a grant from Intel Corp.

References

1. G. E. Moore. Readings in computer architecture. chapter Cramming more components onto integrated circuits, pages 56–59. 2000.
2. S. Borkar and A. A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, 2011.

3. C. G. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik. Parallelizing the web browser. In *Proceedings of the First USENIX Workshop on Hot Topics in Parallelism*, 2009.
4. L. D. Paulson. Developers shift to dynamic programming languages. *Computer*, 40(2):12–15, 2007.
5. N. B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, 2010.
6. M. W. Hall, D. A. Padua, and K. Pingali. Compiler research: the next 50 years. *Commun. ACM*, 52(2):60–67, 2009.
7. D. Levinthal. Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors, 2009.
8. C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005.
9. B. Piccart, A. Georges, H. Blockeel, and L. Eeckhout. Ranking commercial machines through data transposition. In *Proceedings of the 2011 IEEE International Symposium on Workload Characterization*, 2011.
10. K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012.
11. J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. F. P. O'Boyle, and O. Temam. Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the International Symposium on Code Generation and Optimization*, 2007.
12. G. Fursin and O. Temam. Collective optimization: A practical collaborative approach. *ACM Trans. Archit. Code Optim.*, 7(4), December 2010.
13. D. Grewe and M. F. P. O'Boyle. A static task partitioning approach for heterogeneous systems using opencl. In *Proceedings of the 20th International Conference on Compiler Construction*, 2011.
14. R. W. Moore and B. R. Childers. Automatic generation of program affinity policies using machine learning. In *22nd International Conference on Compiler Construction 2013*, pages 184–203, 2013.
15. Y. Zhang and M. Voss. Runtime empirical selection of loop schedulers on hyperthreaded smps. In *Proceedings of The 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.
16. B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, 2006.
17. C. Dubach, T. M. Jones, and M. F. P O'Boyle. Microarchitectural design space exploration using an architecture-centric approach. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007.
18. C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.
19. K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere. Performance prediction based on inherent program similarity. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2006.

20. R. Meeuws, S. A. Ostadzadeh, C. Galuzzi, V. M. Sima, R. Nane, and K. Bertels. Quipu: A statistical model for predicting hardware resources. *ACM Trans. Reconfigurable Technol. Syst.*, 6(1), 2013.
21. M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. (PNAS)*, 1998.
22. R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28, 1958.
23. P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection.* John Wiley & Sons, Inc., 1987.
24. A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3), August 2004.
25. M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36(1):111–147, 1974.
26. M. F. Janowitz. *Ordinal and Relational Clustering.* World Scientific, 2010.
27. A. Phansalkar, A. Joshi, L. Eeckhout, and L.K. John. Measuring program similarity: Experiments with spec cpu benchmark suites. 2005.
28. J. Reinders. *VTune Performance Analyzer Essentials: Measurement and Tuning Techniques for Software Developers.* Engineer to Engineer Series. Intel Press, 2005.
29. C. Jung, S. Rus, Brian P. Railing, N. Clark, and S. Pande. Brainy: effective selection of data structures. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, pages 86–97, 2011.
30. J. J. Dujmovic. Universal benchmark suites. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1999.
31. J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, 2003.
32. E. Park, J. Cavazos, and M. A. Alvarez. Using graph-based program characterization for predictive modeling. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, pages 196–206, 2012.
33. C. W. J. Granger and R. Ramanathan. Improved methods of combining forecasts. *Journal of Forecasting*, 3, 1984.
34. J. L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4), September 2006.
35. V. Aslot and R. Eigenmann. Performance characteristics of the spec omp2001 benchmarks. *SIGARCH Comput. Archit. News*, 2001.
36. M. S. Muller, J. Baron, W. C. Brantley, H. Feng, D. Hackenberg, R. Henschel, G. Jost, D. Molka, C. Parrott, J. Robichaux, P. Shelepugin, M. Waveren, B. Whitney, and K. Kumaran. SPEC OMP2012 An Application Benchmark Suite for Parallel Systems Using OpenMP. In *OpenMP in a Heterogeneous World*, volume 7312 of *Lecture Notes in Computer Science*. 2012.
37. 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, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks: summary and preliminary results. In *Proceedings of the Conference on Supercomputing*, 1991.
38. N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel white paper*, 2008.