Ranking Commercial Machines through Data Transposition

Beau Piccart[‡] Andy Georges[†] Hendrik Blockeel[‡] Lieven Eeckhout[†]

[‡]Department of Computer Science, K.U.Leuven, Belgium
[†]Department of Electronics and Information Systems, Ghent University, Belgium

Abstract

The performance numbers reported by benchmarking consortia and corporations provide little or no insight into the performance of applications of interest that are not part of the benchmark suite. This paper describes data transposition, a novel methodology for addressing this ubiquitous benchmarking problem. Data transposition predicts the performance for an application of interest on a target machine based on its performance similarities with the industry-standard benchmarks on a limited number of predictive machines. The key idea of data transposition is to exploit machine similarity rather than workload similarity as done in prior work, i.e., data transposition identifies a predictive machine that is most similar to the target machine of interest for predicting performance for the application of interest.

We demonstrate the accuracy and effectiveness of data transposition using the SPEC CPU2006 benchmarks and a set of 117 commercial machines. We report that the machine ranking obtained through data transposition correlates well with the machine ranking obtained using measured performance numbers (average correlation coefficient of 0.93). Not only does data transposition improve average correlation, we also demonstrate that data transposition is more robust towards outlier benchmarks, i.e., the worst-case correlation coefficient improves from 0.59 by prior art to 0.71. More concretely, using data transposition to predict the top-1 machine for an application of interest leads to the best performing machine for most workloads (average deficiency of 1.2% and max deficiency of 24.8% for one benchmark), whereas prior work leads to deficiencies over 100% for some workloads.

1 Introduction

Current practice in benchmarking commercial machines is to run industry-standard benchmarks and report their performance numbers. This practice is adopted by various benchmarking consortia and corporations such as EEMBC[1] for embedded systems, TPC[2] for database systems, and SPEC[3] for high-performance computer systems. The information obtained from these benchmarking experiments provides valuable information for comparing existing commercial machines across a broad range of applications. For example, SPEC provides performance results for various benchmarks from several application domains such as compute-intensive workloads, Java workloads, graphics, web servers, mail servers, network file systems, etc.

Although these benchmarking efforts enable users to compare computer system performance across vendors for different types of workloads, they do not provide insight with respect to which computer system performs best for a given application of interest that is not part of the benchmark suite. In particular, it is unclear which performance numbers to base a purchasing decision on, i.e., it is unclear which benchmark is most similar to an application of interest. This is a ubiquitous and long-standing problem in benchmarking that affects various markets of the computer industry. For example, a phone company needs to decide which processor to include in its next-generation cell phone, however, its software may be very different from what the EEMBC benchmark suite provides. In addition, the phone company most likely will not be willing to distribute its proprietary software to third-party hardware vendors. Similarly, an Internet-service provider or a supercomputer host needs to decide which processors to provide in the data center, however, the software that will be run may be very different from what SPEC provides.

Prior work in this area by Hoste et al. [4] approached this problem by identifying a benchmark or a number of benchmarks in the benchmark suite that are most similar to the application of interest, see Figure 1(a). An inherent problem with this approach is that the application of interest may exhibit execution characteristics that are very different from the benchmarks included in the benchmark suite (i.e., the application of interest may be an outlier with respect

[1] http://www.eembc.org
[2] http://www.tpc.org
[3] http://www.spec.org

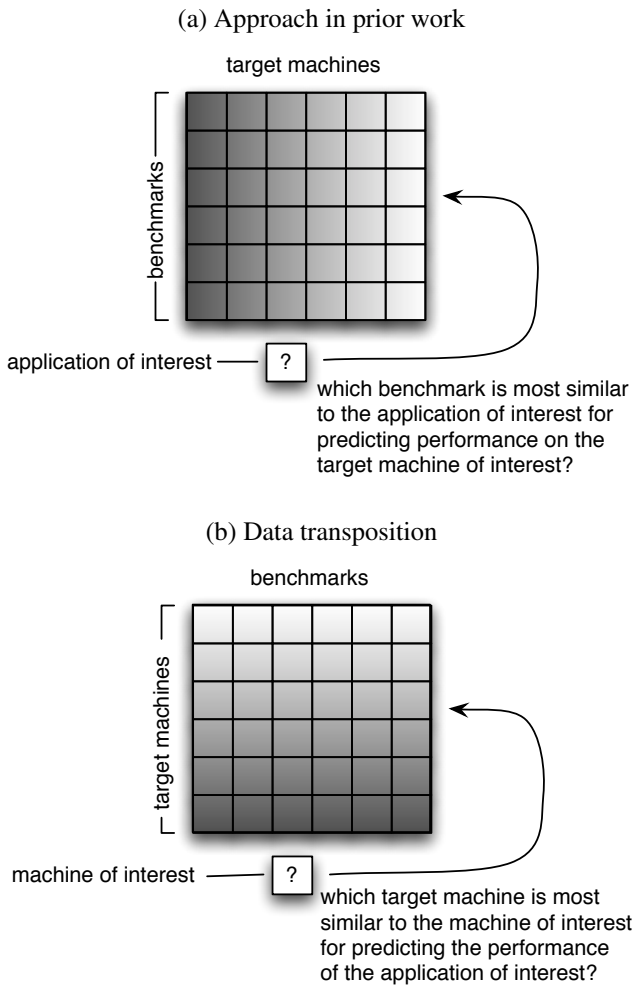

Figure 1: High-level conceptual comparison between (a) the approach in the work of Hoste et al. [4] and (b) data transposition.

to the benchmark suite), hence, it is unclear how useful the benchmark suite is for making an accurate performance prediction.

In this paper, we propose a very different technique, called *data transposition*. The key idea is to transpose the data matrix and solve the dual problem of finding the machine that is most similar to the target machine for predicting performance for an application of interest, see Figure 1(b). In other words, instead of finding the benchmark that is most similar to the application of interest for predicting performance on a target machine of interest, data transposition aims at finding the predictive machine that is most similar to the target machine of interest for predicting performance for the application of interest. While both approaches basically solve the same problem, we demonstrate that data transposition leads to more accurate performance predictions, the fundamental reason being that data transposition enables capturing outlier workload behavior better. Intuitively speaking, an application of interest that exhibits outlier behavior on one machine is also likely to exhibit outlier behavior on another machine. An empirical model then extrapolates outlier behavior across machines.

Data transposition assumes a (potentially large) set of target machines, to which the user has no access but for which benchmarking results are available for a (limited) set of benchmarks, e.g., performance numbers published by a benchmarking consortium such as SPEC. Further, a limited number of so-called predictive machines are assumed to be available to the user on which both the benchmarks and the application of interest can be run. Data transposition then predicts the performance of the application of interest on each of the target machines. It does so based on the published performance numbers for the target machines and the benchmarks along with a limited number of measurements that need to be done on the predictive machines using both the benchmarks and the application of interest; the method does not require executing the application of interest on the target machine. As part of its methodology, data transposition builds empirical models (i.e., a linear regression model and a neural network model in this paper) that predict performance on a target machine based on the performance on a (limited number of) predictive machine(s). The methodology then uses the empirical model to predict the performance on each of the target machines based on its performance on the predictive machines.

Our experimental evaluation using SPEC CPU2006 and performance numbers for 117 commercial machines demonstrates the method's accuracy. Data transposition predicts the ranking of the commercial machines with a correlation coefficient of 0.93 compared to the ranking obtained with measured performance numbers, whereas prior art achieves a ranking of 0.86. The top-1 machine according to data transposition yields a 1.2% performance deficiency on average (24.8% max) compared to the real top-1 machine for the given the application of interest. Prior work in this area by Hoste et al. [4] is accurate as well for most benchmarks, except for outlier workloads for which we observe deficiencies over 100%. Furthermore, we demonstrate the method's ease of use: we find that only a few predictive machines are sufficient for making accurate performance predictions.

This paper is organized as follows. We briefly describe prior work in this area in the next section. In Section 3, we then present data transposition and we elaborate on how it advances beyond prior work. We discuss potential applications in Section 4. Section 5 discusses our experimental setup, and we presents the results on the accuracy of data transposition in Section 6. Finally, we discuss related work in Section 7 and conclude in Section 8.

2 Prior Work

The problem that motivates this work can be summarized as follows. Assume we have an application of interest for which we want to rank a set of commercial machines and predict the best machine or the top-n best performing machines. We therefore rely on an existing performance database that is comprised of performance numbers for a number of benchmarks and machines. The approach taken by prior work was to exploit the similarity between the application of interest and the industry-standard benchmarks across these machines, so that an informed estimate can be made for the performance of the application of interest across the target machines.

In particular, Hoste et al. [4] use performance scores of a standardized benchmark suite on the target machines of interest, and in addition, they measure a set of microarchitecture-independent characteristics for the application of interest which they relate to the benchmarks in the standardized benchmark suite. These microarchitecture-independent characteristics capture the inherent program behavior that is unbiased towards a particular microarchitecture. They rely on the notion of similarity between the application of interest and the benchmarks (in terms of their microarchitecture-independent characteristics) to predict the performance of the application of interest. The key issue in this approach is to determine how differences in microarchitecture-independent characteristics translate into performance differences. They use a genetic algorithm to learn this relationship across a variety of machines. In short, Hoste et al. use the standardized benchmarks as proxies for the application of interest based on behavioral similarity.

The framework proposed in this paper is dual to Hoste et al.'s approach. Whereas they identify the benchmark(s) most similar to the application of interest to predict performance on a target machine, our approach transposes the problem and identifies the predictive machine most similar to the target machine to predict target machine performance for the application of interest. The intuition behind Hoste et al.'s approach is that workloads exhibiting similar inherent program behavior are likely to yield similar performance across a range of machines. This approach is effective for applications of interest that show similarity to the benchmarks in the benchmark suite, however, for applications of interest that are dissimilar to any of the benchmarks, so-called outliers, the method is unlikely to yield accurate performance predictions. Data transposition on the other hand overcomes this inefficiency by building on the notion of machine similarity: an application of interest that is dissimilar to any of the benchmarks and that may yield different performance on a particular machine, is also likely to yield different performance on other machines. In other words, an application of interest exhibiting outlier performance on a predictive machine is likely to exhibit outlier performance on the target machines.

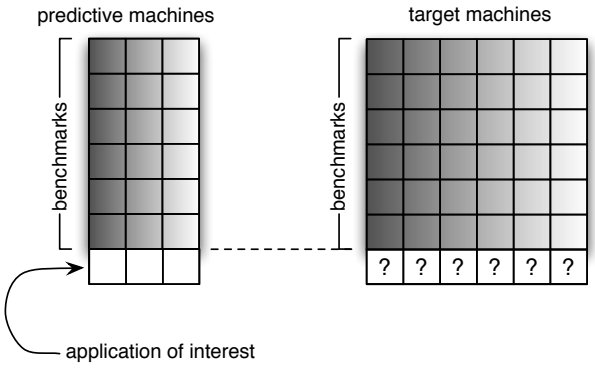

Figure 2: Problem statement and terminology.

In addition to the observation that data transposition leads to more accurate predictions for applications of interest that are outliers compared to the benchmarks in the benchmark suite, it does not require time-consuming profiling runs for collecting microarchitecture-independent program characteristics as in the Hoste et al. approach. Through data transposition, a limited number of real hardware runs on the predictive machines is sufficient for making accurate predictions on the target machines. This makes data transposition both faster and more practical.

3 Data Transposition

Before describing data transposition, we first introduce some terminology and definitions.

3.1 Data set and definitions

Data transposition starts from the two data sets as shown in Figure 2. In both data sets, the rows represent the benchmarks and the columns represent the machines.

The data set on the left in Figure 2 comprises performance numbers for all the benchmarks as well as for the application of interest for the so-called *predictive machines*. These machines are assumed to be available to the user, i.e., the user can run the application of interest as well as the industry-standard benchmarks on these predictive machines to collect performance numbers. Typically, there are fewer predictive machines than *target machines*. The target machines are not available to the user, hence we only have performance numbers for the benchmarks on the target machines, and not for the application of interest.

The data set on the right in Figure 2 is provided by a benchmarking consortium; in fact, we use performance numbers from SPEC CPU2006 in our setup, as we will ex-

plain later. It comprises performance numbers for all benchmarks and all target machines. The goal now is twofold. First, we want to predict the performance of the application of interest on each of the target machines. Second, we want to both rank these machines and identify the best performing target machine(s) for the application of interest.

3.2 Models for performance prediction

We explore two flavors of empirical models for data transposition, namely linear regression and neural networks.

3.2.1 Linear regression

Linear regression builds a linear regression model for each target machine with each predictive machine, see also Figure 3. The regression model that yields the best fit across the predictive machines for a given target machine is retained; this model is subsequently used to predict the performance of the application of interest on that particular target machine. Put differently, the performance for that target machine correlates best with the performance of the chosen predictive machine. Figure 3 illustrates the methodology through an example: three regression models — since there are three predictive machines — are built for target machine #3. In this example, we predict the performance for target machine #3 with the regression model obtained using predictive machine #1 — because predictive machine #1 yields the most accurate linear model for target machine #3. This procedure is repeated for all target machines, which enables us to rank the target machines based on the predicted performance numbers. This ranking provides the relative ordering of target machines for the application of interest; the top-1 machine is predicted to yield the highest relative performance.

3.2.2 Neural networks

Neural networks can also be used for performance prediction through data transposition. Neural networks have the advantage over linear regression models that they can model non-linear relationships. Figure 4 illustrates how this is done. The input to the neural network is the performance of the benchmark applications, and the output is the predicted performance for the application of interest, on the target machine. We consider a multi-level perceptron in this work. We train a neural network using the set of predictive machines. Training the neural network involves inputting the performance numbers of the benchmarks on the predictive machines, and expecting the performance for the application of interest at the output. The training algorithm then learns the neural network to predict the performance for the application of interest based on the performance numbers for the benchmarks. Model training is done using performance data on the predictive machines only. Once the model is trained, it is used to predict performance for the application of interest on each of the target machines. Intuitively speaking, the neural network learns how the performance of the application of interest relates to the other benchmarks. The implicit assumption is that this relationship is similar on the target machines as it is on the predictive machines.

4 Potential Applications

We envision several potential applications for data transposition.

Guiding purchasing decisions When purchasing a new computer system, the customer has to rely on published performance numbers as provided by benchmarking consortia and corporations such as SPEC, EEMBC and TPC. These numbers however only quantify performance for a number of a standardized benchmarks. As a result, it is unclear to which benchmark(s) the application of interest is most similar, and by consequence it is unclear which performance number(s) to base a purchasing decision on. Typically, these decisions are driven by average performance figures across the entire benchmark suite, or they are typically based on presumed similarities across applications from the same application domain. Data transposition provides a methodology for ranking machines, which enables making better purchasing decisions, as we will demonstrate later in the evaluation section of this paper.

Performance prediction of unavailable hardware Prototype hardware or expensive hardware may be hard to obtain for experimentation and measurement. Data transposition provides a solution to performance evaluation on unavailable hardware, i.e., by comparing the performance for the application(s) of interest against a benchmark suite (which is to be run only once on the expensive prototype hardware), useful performance predictions and assessments can be obtained.

Fast design space exploration Simulation-based processor design space exploration is extremely time consuming. Cycle-accurate simulators typically incur a slowdown compared to real hardware of at least 5 orders of magnitude. Hence, simulating one minute of real execution time takes at least two months of simulation time for evaluating a single microarchitecture design point. Obviously, exploring and refining a microarchitecture design at these speeds is infeasible. Data transposition may help speedup design space

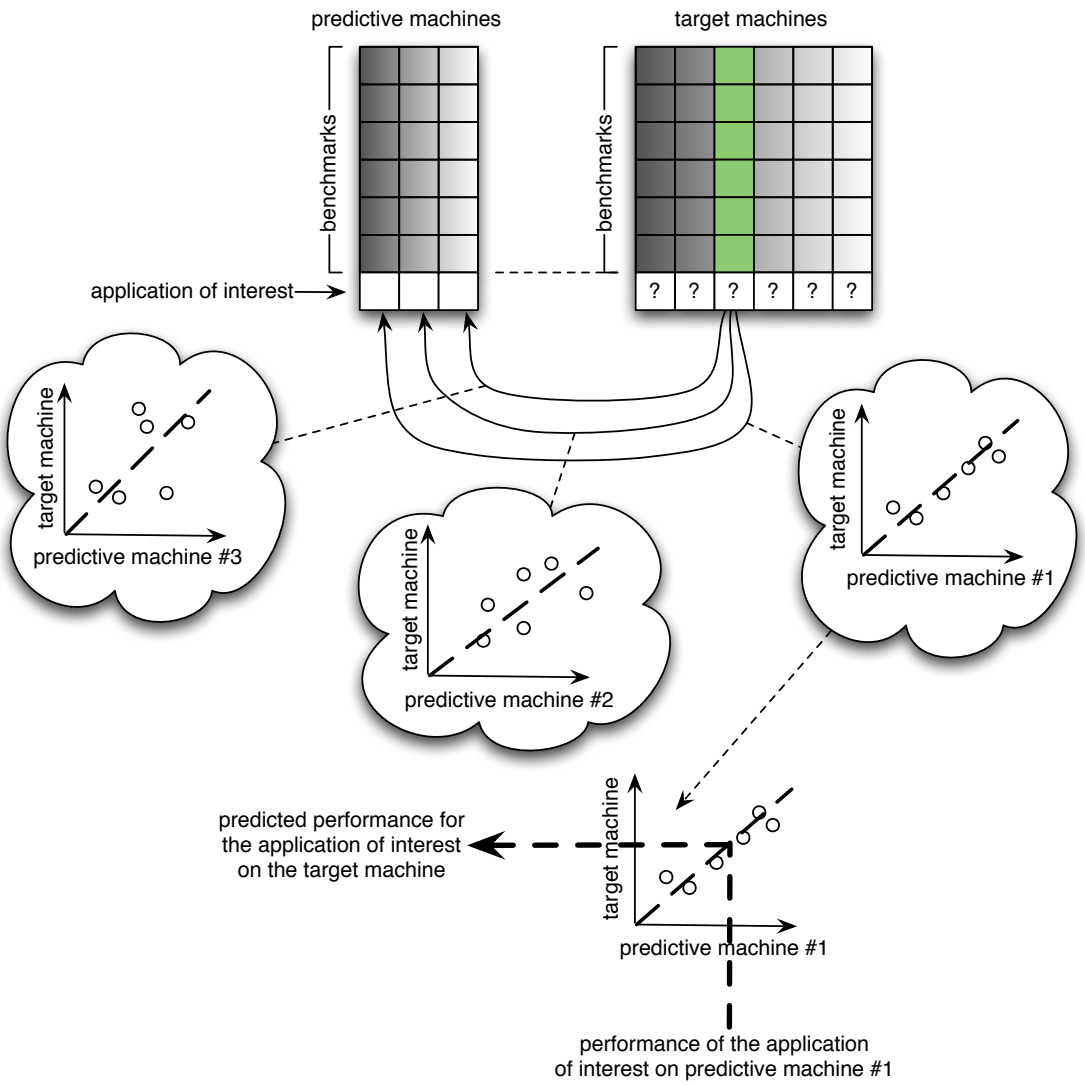

Figure 3: Performance prediction through data transposition using linear regression.

exploration: simulating a number of representative benchmarks in detail on the slow simulator is sufficient to predict the performance of other benchmarks and applications.

Task scheduling on heterogeneous systems Research into heterogeneous computer systems is gaining importance at different levels in the computing range. For example, the composition of computing nodes in a grid or data center may be heterogeneous due to upgrades or by design; heterogeneity may also be considered to increase the energy efficiency of a multi-core processor design [8]; or, heterogeneity may emerge because of chip technology process variability in a homogeneous multi-core processor [13]. An important question in heterogeneous system design is how to schedule the applications for maximizing overall system performance. Data transposition may be an enabler to drive the scheduling algorithm on heterogeneous systems by providing performance predictions for each of the computing nodes; the scheduling algorithm can then use these performance predictions to yield better schedules.

5 Experimental Setup

5.1 Benchmarks and platforms

Our data set contains reported performance numbers for a set of industry-standard benchmarks on a number of commercial machines. In particular, for this study, we use performance numbers reported for the SPEC CPU2006

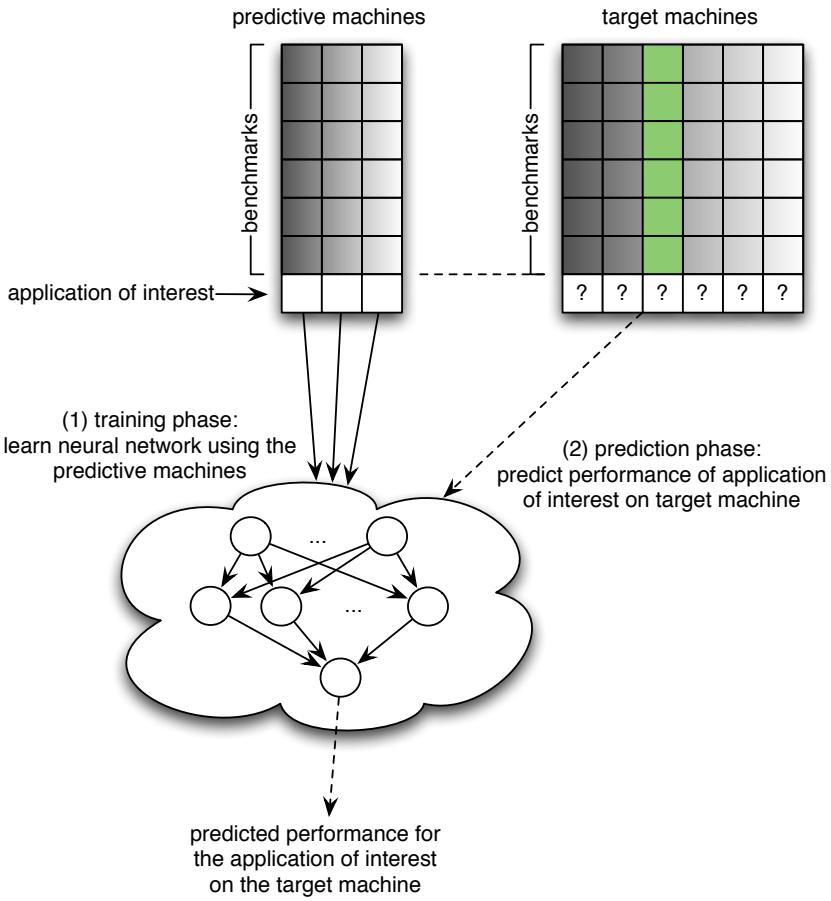

Figure 4: Performance prediction through data transposition using neural networks.

benchmark suite[4] which includes 29 integer and floating-point performance benchmarks. We use the *speed* ratios with base optimization, i.e., `SPECint_base2006` and `SPECfp_base2006`; these speedup numbers are relative to a reference SUN Ultra5_10 workstation with a 296MHz SPARC processor.

We selected 117 commercial machines out of the 1K+ machines that are available on the SPEC website as of Dec 2009. These 117 machines were chosen such that they are as diverse as possible in terms of their (micro)architecture, instruction-set architecture, technology node, etc., see Table 1. For each processor family, we pick a number of machines by CPU nickname — different CPU nicknames reflect differences in microarchitecture, chip technology, cache sizes, bus speed, etc. For each nickname we include three machines. For example, our dataset includes 9 AMD Opteron K10 machines in total, see also the first line in Table 1: 3 machines with a Barcelona CPU, 3 with an Istanbul CPU, and 3 machines with a Shanghai CPU.

[4] `http://www.spec.org/cpu2006/`

As we have outlined in Section 3, we require the data set to be split into two groups: a set of predictive machines versus a set of target machines. Because the selection of predictive machines may have significant impact on the overall accuracy, we will consider different sets of predictive machines throughout our experiments. In all of our experiments, we use a cross-validation setup to allow for a fair evaluation of our methodology. This means there is no overlap between the set of predictive versus target machines: for a given set of predictive machines — a processor family in this study — we remove those machine types from the set of target machines, see also Figure 5.

Additionally, we also consider a leave-one-out methodology with respect to the benchmarks. This means we pick a single benchmark out of the benchmark suite — this is our application of interest — and we build a prediction model using the remaining 28 benchmarks, see also Figure 5. Once the model is built, we compare the predicted performance for the application of interest against its measured performance on each of the target machines.

Processor family	*CPU nickname*
AMD Opteron (K10)	Barcelona, Istanbul, Shanghai
AMD Opteron (K8)	Santa Rosa, Troy
AMD Phenom	Agena, Deneb
AMD Turion	Trinidad
IBM POWER 5	POWER5+
IBM POWER 6	POWER6
Intel Core 2	Allendale, Conroe, Kentsfield, Merom-2M, Penryn-3M, Wolfdale, Yorkfield
Intel Core Duo	Yonah
Intel Core i7	Bloomfield XE
Intel Itanium	Montecito
Intel Pentium D	Presler
Intel Pentium Dual-Core	Allendale
Intel Pentium M	Dothan
Intel Xeon	Bloomfield, Clovertown, Conroe, Dunnington, Gainestown, Harpertown, Kentsfield, Lynnfield, Tigerton, Tulsa, Wolfdale-DP, Woodcrest, Yorkfield
SPARC64 VI	Olympus-C
SPARC64 VII	Jupiter
UltraSPARC III	Cheetah+

Table 1: The machines considered in this study sorted by processor family. Our selection contains 3 machines of each CPU nickname.

6 Evaluation

We evaluate performance prediction through data transposition in three different settings: (i) targeting a processor family based on performance numbers for other processor families, (ii) targeting newer machines based on a predictive set of older machines, and (iii) limiting the set of predictive machines. Throughout the evaluation we refer to the prior work proposed by Hoste et al. [4] as *GA-kNN*, as it involves a genetic algorithm (*GA*) to learn how to weight microarchitecture-independent workload differences to performance differences and then derives a performance prediction based on the k nearest neighbors (*NN*) in the workload space; we assume 10 neighbors in our setup ($k = 10$). We use the WEKA[5] v3 Multilayer Perceptron implementation with default settings as the neural network. We refer to the data transposition approach using the ^{T} superscript, and we refer to the linear regression approach as NN^T (it selects the best fitting predictive machine or 'nearest neighbor' for making a prediction) and the neural network approach as MLP^T (as it uses a multi-level perceptron).

[5] http://www.cs.waikato.ac.nz/ml/weka/

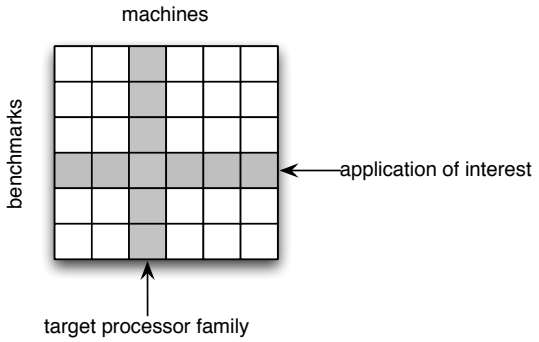

Figure 5: Cross-validation and leave-one-out setup: we eliminate the target machine and application of interest from the dataset when training the empirical performance models.

6.1 Metrics

We use three different metrics to quantify the accuracy of data transposition: (i) target machine ranking, (ii) the top-1 performance prediction error, and (iii) the average performance prediction error. We now briefly discuss each of these metrics.

Ranking measures the ability of the method to predict the relative ranking of the target machines. This is done by first predicting the performance for the application of interest on each target machine. We then rank the machines according to the predicted performance for the application of interest. The predicted ranking is then compared to the actual ranking using the Spearman rank correlation coefficient. The correlation coefficient ranges between 0 and 1; a correlation coefficient of one means a perfectly predicted ranking.

Once a predicted ranking has been obtained, we compare the speedup of its top machine with the speedup of the actual top machine, yielding the top-1 error. In Section 4, we discussed various applications, including the case for guiding purchase decisions. The top-1 error indicates what the loss in performance would be if a purchase is following the performance prediction.

The third metric is the average prediction error across all target machines and benchmarks.

6.2 Predicting another processor family

In the first experiment, we consider a single processor family as the set of target machines, and we use the machines from the other families as predictive machines. Following the cross-validation approach outlined in Section 5,

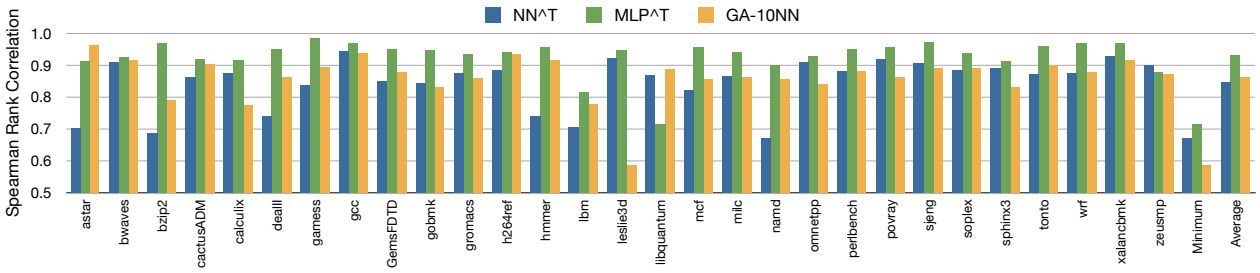

Figure 6: Spearman rank correlation coefficient for data transposition (NN^T and MLP^T) versus prior work (*GA-kNN*).

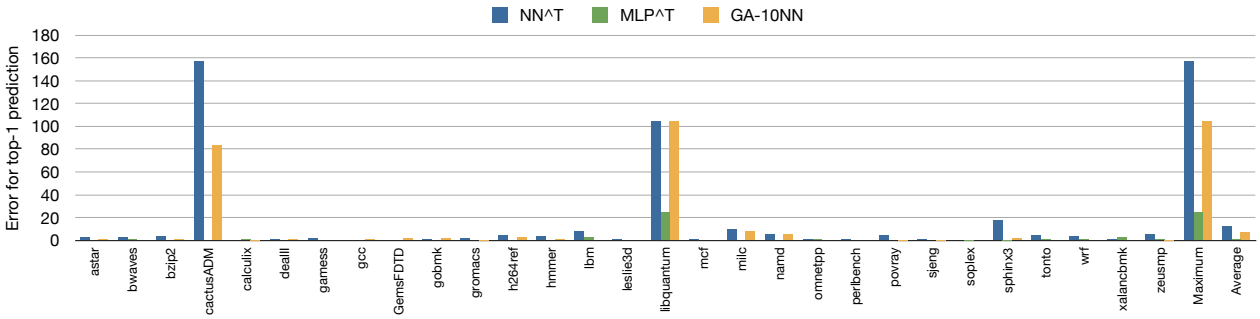

Figure 7: Top-1 prediction error for data transposition (NN^T and MLP^T) versus prior work (*GA-kNN*).

	NN^T	MLP^T	*GA-kNN*
Rank correlation	0.85 (**0.67**)	**0.93** (**0.71**)	0.86 (0.59)
Top-1 error	11.9 (156.7)	**1.21** (**24.8**)	7.30 (104)
Mean error	**4.04** (**31.81**)	**1.59** (**19.4**)	6.25 (51.34)

Table 2: Performance comparison for the different methods using processor-family cross-validation. Numbers in bold represent cases where data transposition outperforms the previously proposed *GA-kNN* method. Average numbers are presented; the numbers between brackets give the worst case.

we have 17 predictive/target pairs on top of the leave-one-out cross-validation at the benchmark level. The results are summarized in Table 2; we aggregated the results across both the target machines and the benchmarks. Average numbers are reported as well as worst-case results; the worst case numbers reports the worst-case result across all target machines and benchmarks, and are shown between brackets. The MLP^T approach beats the other approaches on all three metrics. Compared to *GA-kNN*, the NN^T approach has a slightly lower average rank correlation (0.85 vs. 0.86), but does significantly better in the worst case (0.67 vs. 0.59). We obtain a similar result for the average error, where NN^T outperforms *GA-kNN* (4.04 vs. 6.25). These results indicate that approaches based on data transposition are able to outperform the state-of-the-art when predicting the performance of machines with an unseen architecture.

Figures 6 and 7 show the same data on a per-benchmark basis for the rank correlation coefficient and the top-1 prediction error, respectively. These results show that data transposition is better capable of accurately predicting performance of outlier benchmarks compared to the previously proposed *GA-kNN* approach [4]. In particular, the leslie3d benchmark is an outlier benchmark compared to the other benchmarks in the benchmark suite, and the *GA-kNN* method has difficulty making an accurate prediction (i.e., rank correlation coefficient of 0.59). Data transposition on the other hand improves the predicted ranking to 0.92. Similarly, for the cactusADM and libquantum benchmarks, data transposition using neural networks is the most accurate approach for predicting the top-1 performing machine, see also Figure 7; data transposition using linear regression and the prior state-of-the-art approach are highly inaccurate. These benchmarks are outliers with higher-than-average SPEC scores, and yields the highest performance on an Intel Xeon Gainestown system. The namd and hmmer are outliers at the opposite side of the spectrum: these benchmarks have lower-than-average SPEC scores, and yield the highest performance on Intel Montecito processor systems. Both data transposition and the prior work are accurate at estimating performance for these benchmarks.

Data transposition using neural networks is more accurate than using linear regression. This is apparent from both Figures 6 and 7. The reason is that neural networks can model non-linear relationships, as mentioned before. Both the average and minimum rank correlation coefficients are higher for the neural networks compared to linear regression. Also, data transposition using neural networks is very accurate when it comes to predicting the top-1 machine. Whereas *GA-kNN* and data transposition through linear regression yields prediction errors that are higher than 100%, data transposition using neural networks brings the error down to 25% at most for one of the benchmarks; for the other benchmarks, data transposition using neural networks predicts the top-1 machine with (near) perfect accuracy.

6.3 Predicting future machines

The previous experiment did not take into account the release date of the different machines and aimed at predicting the performance of a processor family based on other processor families, irrespective of their release date. A relevant case in practice might be to predict the performance for a future processor family. To this end, we now limit the target machines to those released in 2009, using machines that were released before 2009 only as the predictive set. We distinguish three possibilities for the predictive set: the machines released in 2008, 2007 and pre-2007. This distinction allows us to see how far into the future a set of predictive machines can reliably predict performance. The results are summarized in Table 3, and we briefly discuss them now.

Predicting one year into the future, (i.e., using the 2008 machines to predict the 2009 machines), works best using the proposed data transposition approaches. On all three metrics, data transposition outperforms the previous state-of-the-art *GA-kNN*. We obtain an increase in the Spearman rank correlation from 0.87 to 0.93 and a reduction in top-1 and mean error from 6.84 and 10.75 to 2.17 and 4.38, respectively. NN^T does somewhat better than MLP^T, which seems to suggest that MLP^T is more sensitive to the training data than NN^T.

If we go back one more year (i.e., using the 2007 machines to predict performance of the 2009 machines), *GA-kNN* achieves a better ranking. Note however that this method does not rely on data from these predictive machines, and takes only the target machines and the benchmark characteristics into account. Even though the predicted ranking correlates better with the actual ranking under *GA-kNN*, data transposition does better on the mean error score. This is the case even for machines predating 2007. The reason why *GA-kNN* performs relatively better than data transposition when predicting further in the future is that *GA-kNN* bases its prediction on a microarchitecture-independent characterization of the workloads only, which is independent of time. Data transposition on the other hand makes a prediction based on historic performance numbers from older machines..

From this we conclude that data transposition (both NN^T and MLP^T) are more accurate when predicting into the near future than *GA-kNN*. However, when predicting in the distant future (two years and more ahead), then data transposition using linear regression (NN^T) is more accurate than using neural networks. Even though *GA-kNN* achieves a better ranking when predicting in the distant future, both the top-1 and mean error are smaller for data transposition using linear regression (NN^T).

6.4 Limited number of predictive machines

We now consider the case where we need to make a performance prediction based on a limited number of predictive machines. The reason is that we need to run benchmarking experiments on the predictive machines for data transposition to work, as explained previously. Limiting the number of predictive machines increases the practicality of method. To evaluate how well data transposition works considering a limited number of predictive machines, we set up the following experiment. The target machines all have been released in 2009, whereas the predictive machine are a subset of the machines released in 2008. We use three subset sizes: 10, 5 and 3.

The results of this experiment are summarized in Table 4. As expected, prediction accuracy decreases with a limited number of predictive machines, however the effect is relatively small. The MLP^T method is most robust to a decrease in the number of predictive machines: even with three predictive machines only does MLP^T outperform *GA-kNN*. NN^T suffers more from a limited number of predictive machines. *GA-kNN* is able to rank the machines better than NN^T when we use 5 or less predictive machines, but performs worse compared to NN^T based on the mean error and the top-1-error. These results demonstrate that data transposition (using neural networks) is accurate even when having access to a limited set of predictive machines only, which makes the method practical to use.

6.5 Selecting predictive machines

Now that we know that a limited number of predictive machines is sufficient, the question is how to select these predictive machines. An easy-to-implement approach is to select predictive machines randomly. Another approach may be to select predictive machines such that they maximize the coverage relative to the target machines. In other words, by choosing a diverse set of predictive machines one may maximize the likelihood of finding a similar (close-

	(a) MLP^T			(b) NN^T		
	2008	2007	older	2008	2007	older
Rank correlation	**0.93** (**0.71**)	0.80 (0)	0.77 (0.49)	**0.92** (**0.76**)	0.82 (0.37)	0.74 (0.31)
Top-1 error	**3.78** (**50**)	9.23 (119)	6.84 (**43**)	**2.17** (**43**)	**4.31** (**92**)	**2.07** (**29.3**)
Mean error	**5.50** (**65.61**)	**8.10** (**70.79**)	**8.36** (**64.89**)	**4.38** (**35.16**)	**9.22** (**82.13**)	**9.22** (**53.34**)

Table 3: Performance comparison for the different methods predicting the performance for machines from 2009 using older machines. Numbers in bold outperform the previously proposed *GA-kNN* method; the numbers between brackets report the worst case result.

	(a) MLP^T			(b) NN^T		
Subset size	10	5	3	10	5	3
Rank correlation	**0.9**	**0.89**	**0.89**	0.87	0.81	0.81
Top-1 error	**6.17**	**2.79**	**3.04**	**2.17**	**5.49**	**5.49**
Mean error	**5.53**	**4.93**	**5.16**	**5.17**	**6.00**	**6.05**

Table 4: Performance comparison for the different methods predicting the performance for machines from 2009 using a small subset from the 2008 machines. The numbers in bold indicate cases where data transposition outperforms the previously proposed *GA-kNN* method.

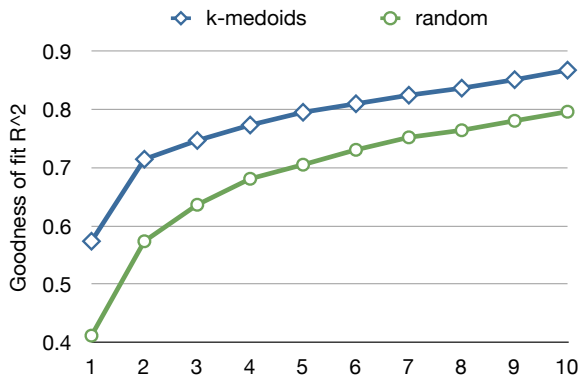

Figure 8: Comparing random selection versus k-medoid clustering for selecting predictive machines through MLP^T. For the random case, 50 random selections were averaged.

enough) (set of) predictive machine(s) for each target machine. This may improve the accuracy of the performance prediction, and hence, it could be viewed of as the best possible approach. We implement this approach by choosing the predictive machines through k-medoid clustering, which randomly selects k cluster centers (predictive machines), initially, and groups all the remaining machines to their closest cluster. Assigning machines to clusters changes the centroids of the clusters, hence, different machines emerge as cluster centers (i.e., a new set of predictive machines is constructed), after which new cluster centers need to be determined, etc. This process is iterated until convergence or steady-state, i.e., all machines are assigned to a cluster and cluster membership does not change across iterations of the algorithm. The cluster centers then are the predictive machines. This results in a diverse set of machines, for example an Intel Core 2, Pentium D Presler, Xeon Gainestown and a SPARC 64 VII when selecting 4 predictive machines. Figure 8 shows the goodness of fit for random selection versus k-mediod clustering as a function of the number of predictive machines for MLP^T. The key observation from this graph is that k-medoid clustering outperforms random selection by over a factor two, e.g., two predictive machines selected by k-medoid clustering achieve a better fit (0.714) than five randomly selected machines (0.705).

7 Other Related Work

We now discuss other related work beyond the prior work discussed in Section 2.

7.1 Empirical performance modeling

Empirical modeling leverages statistical inference and machine learning techniques such as regression modeling or neural networks to automatically learn a performance model from training data. Joseph et al. [6] apply linear regression to processor performance analysis: they build linear regression models that relate micro-architectural parameters (along with some of their interactions) to overall processor performance. Joseph et al. only use linear regression to test microarchitecture design parameters for significance, i.e., they do not use linear regression for predictive modeling.

Linear regression assumes that the response variable behaves linearly with its input variables. This assumption is often too restrictive. Lee and Brooks [9] advocate spline-based regression modeling in order to capture non-linearity. A spline function is a piecewise polynomial used in curve fitting. A spline function is partitioned in a number of intervals with different continuous polynomials.

An artificial neural network is an alternative approach for building an empirical model. Neural networks are machine learning models that automatically learn to predict (a) target(s) from a set of inputs. The target typically is performance and/or power or any other metric of interest, and the inputs typically are microarchitecture parameters. Neural networks could be viewed of as a generalized non-linear regression model. Several groups have explored the idea of using neural networks to build performance models, see for example Ipek et al. [5], Dubach et al. [1] and Joseph et al. [7]. Lee et al. [10] compare spline-based regression modeling against artificial neural networks and conclude that both approaches are equally accurate; regression modeling provides better statistical understanding while neural networks offer greater automation.

All of this prior work shares the commonality that it aims at predicting performance for a target machine for a given set of benchmarks. Their goal is to drive architecture design space exploration. We are addressing a related but very different problem: we aim at predicting performance for a target machine for a *novel* workload; this could be viewed as joint workload/architecture exploration.

7.2 Program similarity

Several researchers have proposed methods for quantifying program similarity. Saavedra and Smith [12] use the squared Euclidean distance computed in a benchmark space built up using dynamic program characteristics at the Fortran programming language level such as operation mix, number of function calls, number of address computations, etc. Yi et al. [14] use a Plackett-Burman design for classifying benchmarks based on how the benchmarks stress the same processor components to similar degrees. Eeckhout et al. [3] use principal component analysis to identify similarities across programs. The input given to the principal component analysis can be microarchitecture-dependent [2], microarchitecture-independent [11] or mixed of both [3]. The work described in this paper goes one step further and aims at exploiting program similiarity and dissimilarity to make performance predictions for new workloads.

8 Conclusion

The ubiquitous problem in benchmarking is to predict the performance of an application of interest on a set of target machines the user does not have access to. This paper presented data transposition, a novel method for doing so. It builds empirical models for predicting performance across machines using a standard set of benchmarks. By building such models for a limited number of predictive machines and a (potentially) large set of target machines, data transposition allows for predicting performance of an application of interest on a set of target machines by running it on the predictive machines only. The intuition behind data transposition is that if a workload is (dis)similar to a set of benchmarks on (a) predictive machine(s), it is likely to be proportionally (dis)similar on a target machine. An empirical model is trained to learn how workload (dis)similarity translates into performance differences on actual hardware.

Our experimental results demonstrate the method's accuracy: the ranking achieved through data transposition correlates well with the real ranking (average rank correlation coefficient of 0.93) across a set of 117 commercial machines using the SPEC CPU2006 benchmark suite. We also found that only a few predictive machines are sufficient for achieving accurate predictions. Further, the top-1 machine can be predicted with a 1.2% average prediction error (and 24.8% max error for one workload); in contrast, state-of-the-art method proposed by Hoste et al. [4] leads to errors above 100% for some workloads. A key benefit of data transposition is that it can better predict outlier workload performance.

Acknowledgment

We thank the reviewers for their constructive and insightful feedback. Andy Georges is supported through a postdoctoral fellowship with the Fund for Scientific Research–Flanders (FWO). Additional support is provided by the FWO projects G.0255.08 and G.0179.10, the UGent-BOF projects 01J14407 and 01Z04109, and the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement no. 259295.

References

[1] C. Dubach, T. M. Jones, and M. F. P. O'Boyle. Microarchitecture design space exploration using an architecture-centric approach. In *Proceedings of the IEEE/ACM Annual International Symposium on Microarchitecture (MICRO)*, pages 262–271, Dec. 2007.

[2] L. Eeckhout, A. Georges, and K. De Bosschere. How Java programs interact with virtual machines at the microarchi-

tectural level. In *Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Languages, Applications and Systems (OOPSLA)*, pages 169–186, Oct. 2003.

[3] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism*, 5, Feb. 2003. `http://www.jilp.org/vol5`.

[4] 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 2006 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 114–122, Sept. 2006.

[5] E. Ipek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 195–206, Oct. 2006.

[6] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 99–108, Feb. 2006.

[7] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. A predictive performance model for superscalar processors. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 161–170, Dec. 2006.

[8] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the ACM/IEEE Annual International Symposium on Microarchitecture (MICRO)*, pages 81–92, Dec. 2003.

[9] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 185–194, Oct. 2006.

[10] B. Lee, D. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pages 249–258, Mar. 207.

[11] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John. Measuring program similarity: Experiments with SPEC CPU benchmark suites. In *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 10–20, Mar. 2005.

[12] R. H. Saavedra and A. J. Smith. Analysis of benchmark characteristics and benchmark performance prediction. *ACM Transactions on Computer Systems*, 14(4):344–384, Nov. 1996.

[13] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 363–374, June 2008.

[14] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proceedings of the Ninth International Symposium on High Performance Computer Architecture (HPCA)*, pages 281–291, Feb. 2003.