Benchmark Design for Robust Profile-Directed Optimization

Paul Berube, José Nelson Amaral
Dept. of Computing Science, University of Alberta
Edmonton, Alberta, T6G 2E8, Canada

Abstract

Profile-guided code transformations specialize program code according to the profile provided by execution on training data. Consequently, the performance of the code generated usind this feedback is sensitive to the selection of training data. Used in this fashion, the principle behind profile-guided optimization techniques is the same as off-line learning commonly used in the field of machine learning. However, scant use of proper validation techniques for profile-guided optimizations have appeared in the literature. Given the broad use of SPEC benchmarks in the computer architecture and optimizing compiler communities, SPEC is in a position to influence the proper evaluation and validation of profile-guided optimizations. Thus, we propose an evaluation methodology appropriate for profile-guided optimization based on cross-validation. Cross-validation is a methodology from machine learning that takes input sensitivity into account, and provides a measure of the generalizability of results.

1. Introduction

The SPEC CPU benchmark suite is likely the most significant performance evaluation tool for computer systems and compilers available today. SPEC CPU is the standard used for performance comparison across platforms. SPEC scores are used both as targets for development teams and as a quantification of performance advantage for sales teams. The SPEC CPU programs are a component of the testing framework for compiler teams, both to ensure functionality and to detect performance bugs or new performance-enhancing transformation opportunities. These industrial uses of the SPEC CPU are the primary concern of SPEC's industrial partners, and SPEC CPU fulfills this role very well.

Another significant group of consumers of SPEC CPU are academic researchers. While researchers seldom run the suite as prescribed by SPEC, and hence seldom publish reportable SPEC performance scores, the benchmark programs are well-known to the research community and provide a common framework for the discussion and comparison of research results. Furthermore, SPEC CPU is endorsed by industry as representative of a wide range of important applications, and comes complete with program input sets. Thus, using the benchmark suite minimizes research time for experimental design while providing an implicit suggestion of the general applicability of experimental results. Therefore, in a very significant way, the SPEC CPU benchmark suite guides compiler and architecture research.

Therefore, while the academic community is indebted to SPEC for the value provided by its benchmark suites, we also have an interest and responsibility to help ensure that SPEC CPU maintains the highest possible levels of scientific utility and integrity. Since profile-guided compiler optimization, also known as feedback-directed optimization (FDO) or as profile-directed feedback (PDF), has recently been disallowed from reported peak scores in CPU2006, this is an ideal time to reevaluate both the benchmark contents and the prescribed methodologies as they relate to PDF. Our concern does not lie in whether or not PDF is allowed when reporting SPEC performance scores. Instead, we want PDF to be evaluated in a scientifically sound manner, both in industry and in academia.

PDF has not achieved wide acceptance by compiler users. There are several reasons for this fact, but one significant barrier to acceptance is that the performance obtained without PDF meets or exceeds the requirements of most users. Thus, extra effort for training runs and recompilation does not provide these users with any additional utility. Also, PDF is not widely regarded as robust: it may not improve program performance (and may possibly hurt performance), while users worry about the representativeness of the training inputs they might select. Nonetheless, there is a class of performance-sensitive users who are willing to pay the overhead of a PDF compilation process in the hope of improving program performance.

On the other hand, research in PDF and other learning-compiler lines of investigation are vibrant, and will continue regardless of whether these techniques can be used to report peak scores [17, 13, 11]. These projects will most likely

continue to use the SPEC CPU benchmarks for performance evaluation.

Therefore, there is both industrial and academic demand for benchmarks with effective methodologies for the evaluation of PDF and related technologies. Robustness (or performance stability) is one of the key evaluation criteria for PDF. A benchmark suitable for PDF should asses performance along three dimensions:

1. the range of application domains
2. the typical program inputs
3. the input(s) selected for training

The current structure of the SPEC CPU suite is already designed to address the first dimension, since it is a classic concern independent of PDF. However, even in previous versions of the suite where PDF was allowed (*e.g.*, CPU2000), the second and third dimensions were neglected. In the absence of PDF, these two dimensions are not large concerns. However, when PDF is used, the program is specialized according to the training input, and input sensitivity issues should be considered. Unfortunately, the academic community has generally suffered from the same deficiencies as SPEC CPU.

Section 2 presents some background information pertinent to evaluation methodologies of PDF, including some basic information commonly taken for granted in the related field of machine learning (ML). Section 3 details our proposed evaluation methodology for PDF, while Section 4 discusses the practicality of the proposal. Section 5 offers some concluding remarks.

2. Background

Data inputs are a key component of benchmarks. Evaluation inputs must be carefully selected to represent typical program usage while meeting benchmark requirements such as dynamic memory footprint size, CPU load, and running time. The use of PDF for benchmark programs enhances both the importance and selection difficulty of inputs. In particular, two intuitively similar inputs may not induce similar code transformations in a compiler, or may not present similar performance responses to compiler transformations [4, 5].

Training input selection presents several additional challenges beyond those presented by the selection of evaluation inputs. Training inputs must represent typical program use, without requiring long running times. Furthermore, in large applications, a single training-sized input cannot cover all the important use cases. Therefore, it can be difficult to provide a single representative training input.

Furthermore, there is disagreement in the benchmarking community over what is meant by a "representative training input." Some benchmark authors advocate that an effective way to ensure that the training input is representative of the evaluation input is to include a portion of the evaluation input in the training input. Thus, part of what the compiler sees during training is a perfect predictor of at least some portion of the evaluation input. However, others in the benchmarking community argue that this technique for training input creation makes the training and evaluation inputs too similar [18]. When training inputs are repeated in the evaluation set, over-fitting (see Section 2.1) is rewarded, which is counter the goals of an effective evaluation. Is the purpose of PDF to maximize program performance on a particular input, or to enhance program performance on the range of inputs likely to be seen in practice? If PDF is meant to improve performance in practice rather than to maximize benchmark scores, then training and evaluation workloads should not overlap. Nonetheless, this practice has been used for SPEC CPU benchmarks, such as `gap` from CPU2000 and `hmmer` from CPU2006. Previously, there have been no precise guidelines or rules for input selection for SPEC CPU benchmarks, which has left individual benchmark authors to decide for themselves how these inputs should be devised.

2.1 Lessons from Machine Learning

A training run to generate a program profile for a compiler is very similar to a training run in machine learning. The profile essentially provides a sample data point (of program execution) to which the compiler attempts to fit the program in order to maximize expected program performance. The field of machine learning has developed excellent techniques to robustly evaluate the performance of such offline learning systems. It is time for the compiler and benchmarking communities move from our current evaluation methodologies to the well-established methodologies accepted by the ML community.

The artificial intelligence community has had a strong interest in learning techniques for several decades. One of the earliest goals of mechanical computing was to create a machine with human capabilities. This goal has been most apparent in the field of computerized game playing. The strongest computer players of many well-researched games use machine learning techniques. Furthermore, machine learning techniques have important industrial applications [12, 15]. These advances in machine learning are supported and facilitated by well-developed and well tested evaluation methodologies spanning from theoretical algorithm analysis to practical experimental evaluation.

Machine learning teaches that there are two key ideas when evaluating a learning system. Overfitting the training data must be avoided, and the statistical significance of results must be determined [8].

Underfitting is the problem where the learning system does not learn as much as possible from the training data, and consequently fails to achieve maximum performance on the evaluation data. Underfitting is not a significant problem, but rather indicates that opportunities for improvement exist.

Conversely, overfitting is the case where the learning system matches the training data too closely. Consequently, small variations from the training data in the evaluation data cause large errors by the system. Overfitting can be detected by comparing the performance of the system when evaluated using the training data versus distinct evaluation data. Excellent performance on the training data, but poor performance on the evaluation data suggests that overfitting has occurred. Overfitting is a significant problem that must be avoided. The upshot of the overfitting-underfitting spectrum is that an evaluation must be careful to ensure that the set of training inputs is fully distinct from the set of evaluation inputs. This way, if overfitting does happen, it is detectable and does not inflate the evaluation results. Otherwise, evaluation could report significant performance gains that are not achievable in practice.

Overfitting is particularly problematic when the number of training inputs is low. With few training examples, it is difficult to separate the peculiarities that make inputs non-identical from the commonalities between inputs that should be exploited. If a single training input is used, this distinction is impossible. Since discovering and exploiting the commonalities between inputs is a fundamental task of many learning systems, including PDF compilers, it is imperative that multiple inputs are used during the training process.

The statistical significance of results is very important when evaluating complex systems. System performance is not independent of the data used, and will vary according to the inputs used for training and evaluation. Changing the data may change the maximum possible level of performance. For example, the entropy in data used for a compression algorithm will change how much the data can be compressed, and consequently how significantly the performance of the compression routines impact whole-program. According to the central limit theorem, random samples from a population with finite variance are approximately normally distributed. Performance measures should have finite variance, and are thus approximately normally distributed around their mean. Consequently, the sample standard deviation or confidence intervals can be calculated from the measurements to express the expected spread of performance measures from the mean.

If the standard deviation is small compared to the measured gain, then the measured gain is meaningful, and will very likely be observed in any additional sample points (*i.e.*, other data inputs) that were not tested during evaluation. On the other hand, if the standard deviation (or confidence interval) is large compared to the mean, then the true mean may be significantly different than the measured mean, and the measured mean may not accurately represents the gain expected on other inputs. In other words, the gain observed during evaulation may simply be noise. For example, consider an evaluation that yields a mean speedup of 1.05 over a baseline. Two times the standard deviations is approximately a 95% confidence interval. Thus, if the standard deviation is 0.005, it can be stated with high confidence that the evaluated technique provided a 5% improvement, $\pm$ 1%. Alternatively, if the same evaluation produced a standard deviation of 0.05, then the technique provides a 5% improvement $\pm$ 10%. In this second case, the 5% improvement is not statistically significant, and it is uncertain that the technique improves performance. Thus, calculating standard deviations or confidence intervals provides a good measure of the robustness of the measured performance across the untested inputs that could be encountered in the field, provided that the sample of inputs used for evaluation is representative of the inputs encountered in the field.

2.2 Cross Validation

An appropriate evaluation methodology must account for the problem of overfitting the training data and the need for statistical confidence for performance measurements. A common evaluation techniques used in ML that addresses these concerns is cross-validation [8]. In its simplest form, cross-validation takes a set of data (inputs in the PDF case), splits them into two groups, and uses one group for training and the other for evaluation. More sophisticated cross-validation divides the data in various ways to produce more permutations of training and evaluation data sets. Nonetheless, an essential property of cross-validation is that training and evaluation are repeated several times, using different non-overlapping input sets for training and evaluation. The fact that in every case, the training and evaluation sets are non-overlapping eliminates the problem of overfitting inflating performance results. Performing multiple training and evaluation runs reduces the impact of particularly poor or favorable training or evaluation sets. Finally, the existence of multiple evaluation measurements allows statistical confidence to be calculated.

3. Proposed Benchmark Methodology

PDF is a different compilation technique than traditional non-PDF compilation, with different performance issues and consequently different requirements for a performance-evaluation methodology. Therefore, instead of allowing PDF in reported peak scores, we propose that an optional separate PDF score be reported, along with base and peak.

The proposed PDF evaluation methodology is based on cross-validation, and incorporates the requirements of this technique with the constraints and concerns of performance evaluation for high-performance systems.

3.1 PDF Workload

Cross-validation requires a workload of inputs. We call this set of inputs the PDF workload, W. The intuitive guideline for the PDF workload is that the inputs in the workload should be as varied as possible and should attempt to cover or sample the space of program inputs, biased toward typical program inputs but not selecting these types of inputs exclusively. Rather than attempting to find one input that is representative of program use, the workload as a whole will be representative of program usage. For example, one input could correspond to a particular use-case of the program, and another input could correspond to a different use-case. These two inputs can be completely unrelated, and neither need try to represent all common uses of the program. Program behaviors that are common independent of input characteristics will be universally represented, while input-dependent program behaviors will be represented in proportion to their occurrence in the workload. Consequently, the more important a particular program behavior is to overall program performance, the more frequently it will be represented in the workload.

The inputs selected for the PDF workload should be independent. Input independence is difficult to define precisely, but the data in two inputs should not overlap or have a common origin. For example, a particular chess position should not occur in two input files, images should not be subsections from a common source, or a matrix of data should not be sampled from another matrix used in a different input. Randomly generated data should be avoided because it is both unrepresentative of real data, and multiple data randomly generated in the same way will likely have very similar characteristics. If the program accepts multiple data formats, W should contain examples from as many of these formats as practical. If the data has implicit or explicit dimensionality, different elements of W should avoid repeating the same dimensionality.

Each input in W may potentially be used for both training and evaluation (but only in different evaluation contexts). Therefore, these inputs must run long enough to provide meaningful performance measures, but also be small enough that training runs (possibly an order of magnitude slower than an soptimized non-training run) complete in an acceptable time. Furthermore, these constraints should continue to be met as machine performance continues to increase during the lifetime of the benchmark. Therefore, we suggest that inputs in W should have a (non-training) running time of approximately 2 min.

3.2 Training and Evaluation

The PDF workload will be randomly split into three non-overlapping partitions, each containing an equal number of inputs. Let A, B, and C be the names of the three partitions. These partitions may be specified as part of the benchmark design. A rule of thumb suggests that at least 5 independent measurements are required to make statistical significance tests worthwhile. More than 30 evaluations would typically be considered a large sample size, which enhances the reliability of statistical measures. However, benchmark running time constraints and benchmark author resources place practical limitations on how many inputs can be collected and used. In order to facilitate the cross-validation methodology presented here, the number of inputs should be a multiple of 3. To enable meaningful statistical measures of confidence, a minimum of 9 independent inputs are required, since $\frac{2}{3} \times 9$ provides 6 independent performance measures.

One partition (say A) is selected as the training partition. The compiler will perform a training run using one or more of the inputs in A. If the compiler cannot use profile information from all inputs in A (*e.g.*, can only train on a single input), then the subset of inputs that the compiler uses for training are selected from A in order, according to an input list supplied with the benchmark. Let the program created by training on A be denoted by P_A. Note that the compilation of P_A may use all the settings and flags used to compile for peak. P_A is run on the inputs in $B \cup C$. This training/evaluation cycle is repeated, running P_B on $A \cup C$, and again running P_C on $A \cup B$.

3.3 Reporting PDF Performance

PDF performance will be reported as a mean speedup compared to the program used to report peak performance.

Let T_i^N be the mean running time of an odd number (at least 3) of executions of P_N on input $i \in W$, $N \in \{A, B, C\}$. Likewise, let T_i^P be the same measure using the program compiled for peak. For each input i, exactly two of $\{T_i^A, T_i^B, T_i^C\}$ exist (T_i^N does not exist if $i \in N$, since the same input is never used for both training and evaluation). Denote the set of all such performance measurements by $M = \{T_i^N | \forall N \in \{A, B, C\}, \forall i \in W, i \notin N\}$.

The $PDFPeak$ score is the average speedup compared to peak observed in all PDF-optimized evaluation runs:

$$PDFPeak = \frac{1}{|M|} \sum_{j_i \in M} \frac{T_i^P}{j_i}$$

where i is an input, such that T_i^P and j_i correspond to evaluations using the input i.

In addition to reporting $PDFPeak$, the standard devia-

tion of the speedups must also be reported, using the standard bias-adjusted sample standard deviation equation:

$$\sigma_{PDFPeak} = \sqrt{\frac{1}{|M|-1}\sum_{j_i \in M}(j_i - PDFPeak)^2}$$

Clearly, compiler designers will strive for a high $PDFPeak$ score. In order to achieve this goal, PDF must improve the performance of the program over a large number of unseen inputs. Furthermore, the training phase does not guarantee that selecting a single input from the training set will provide a particularly representative training run. However, using more than one input during training (in particular, all of the training set) likely provides a more representative sample of program execution than any single training input could. Therefore, in order for a PDF compiler to achieve a high $PDFPeak$ score, it must be both robust in the face of the possibility of a poorly chosen training input, as well as in the face of potentially varying behavior in evaluation inputs. Of course, a good way to avoid problems with an unfortunate training input is to use multiple training inputs. The capability to use multiple training inputs is already present in several commercial compilers, including the IBM XL compiler [10] and the Intel C++ Compiler [9].

Additionally, the standard deviation is a critical metric, because it provides confidence for the $PDFPeak$ speedup value. If $(PDFPeak - 2 \times \sigma_{PDFPeak}) > 1$ there is more than a 95% probability that PDF does in fact improve performance compared to peak. Even if $PDFPeak$ is much larger than peak, we cannot be confident that PDF improves performance for the program if $\sigma_{PDFPeak}$ is large compared to that difference. In essence, the standard deviation gives a measure of the robustness of PDF in improving performance, which is a key factor that must be considered when PDF is evaluated.

3.4 Relationship Between PDF and Ref

Traditionally, SPEC CPU benchmarks have included a ref input or workload of inputs used for performance evaluation. In cases where ref is a workload, it is likely appropriate to let ref and W be the same set of inputs. This unification of ref and W not only reduces the input selection burden on benchmark authors, but also reduces the computational overhead required to get the running time measurements required to compute $PDFPeak$. Furthermore, even in the case where PDF is not used, evaluation on a workload of inputs provides statistical confidence measures that evaluation on a single, large, input cannot.

However, in other cases it may be most appropriate for ref to be a small number of, or a single, large input(s) that is/are not suitable for inclusion in W. In these cases, two options are available. The simplest option is to leave the measures of base/peak and $PDFPeak$ completely independent. However, a potentially more informative approach is to let the ref input(s) be a new partition of inputs in W that is never used for training, but only for evaluation. In this way, the performance of PDF on these large (and presumably more important) inputs is taken into account. If this method is used, it may be beneficial to reformulate the equation for $PDFPeak$ to use a weighted mean instead of an unweighted mean, to give more significant to the ref input(s).

4. Practicallity Considerations

A cross-validation approach to PDF evaluation provides performance measurements that are more reliable than traditional approaches. However, this improved evaluation does incur a cost, and it is important that the cost is not prohibitive for any stakeholders.

4.1 Compiler Users

Ultimately, the performance results produced using benchmarks are intended to help end-users of computer systems estimate the performance of various systems for their application workloads. If multiple training inputs are required to reliably obtain the level of performance indicated by the proposed evaluation methodology, end users should also adopt a multi-input training methodology. Despite the immediate impression that such a training methodology would increase the burden on PDF users, training on multiple inputs should not be a significant issue for the class of performance-sensitive users who are interested in PDF. Such users typically maintain sets of inputs to use for regression testing, both for program correctness and program performance. These input sets will include the important use cases of the program. Instead of requiring the user to develop a specific training input that attempts to cover all those use cases (in order to be representative), the user is freed to simply train on all the use cases they have already identified. Therefore, while the training time may be extended, the effort required to facilitate a representative training regiment is significantly reduced.

4.2 Benchmark Users

When properly incorporated into a benchmark suite, cross-validation should be nearly invisible to a benchmark user. The multiple training runs, evaluations, and performance measure calculations should all be performed automatically by the benchmark framework. The only task for the user should be to specify the correct arguments to the compiler to enable the creation and use of program profiles, and possibly to specify the maximum number of inputs to

use for training, if this is a limitation in the PDF implemented by the compiler.

The proposed training and evaluation methodology is not significantly more expensive in terms of computation time than the traditional methodology. Assuming that the PDF workload is unified with the ref workload, the baseline measurements for speedup comparison are taken care of by a standard non-PDF run of the benchmark. Training requires instrumented runs on at most $|W|$ training-sized inputs, but only three additional program compilations. Evaluation requires twice the number of runs used to compute the peak score.

4.3 Benchmark Authors

Benchmark authors face the largest burden from the proposed methodology. Each benchmark author must select the inputs used with the program. Authors have two options when selecting the training set. They may use their expert knowledge to carefully consider a number of inputs and select those that expose different program behaviors or present distinct program use cases. Alternatively, they may select a large collection of inputs, and then use one or more of the existing input similarity clustering techniques from the literature to determine redundancy in the original collection [7, 14].

The ease with which many inputs can be gathered or generated is an important consideration when proposing the collection of a workload of inputs. Based on the documentation of SPEC CPU 2006 benchmark programs [16], there should be little difficulty obtaining inputs for integer-style benchmark programs:

400.perlbench, 433.gcc, 483.xalancbmk There are large repositories on the web of C, C++, and Perl programs. XML documents should be easily obtainable. The reference workloads for these programs already consist of several inputs.

401.bzip2 Any file can serve as a valid input for a compression program. The bzip2 reference workload consists of several input.

429.mcf, 471.omnetpp, 473.astar The algorithms implemented by these programs work on the provided topology. Commodity flow graph, network topologies and path-finding maps may not be plentifully available in the particular formats required by these programs, but the use of format conversion scripts or other input generators should allow for the collection of many inputs with moderate effort from the authors.

445.gobmk, 458.sjeng Both go and chess board positions are easily generated. Furthermore, given that programs for both games compete at events such as the Computer Olympiad [1] and the World Chess Championship [2], collections of well-known board positions in standard formats should be available.

456.hmmer Many online, publicly accessible databases for protein sequences and related information exist, such as the Swiss-Prot [3] database.

462.libquantum Number factoring requires only an integer as input, and an optional base for modular exponentiation.

464.h264ref Video streams are plentiful on the Internet. Furthermore, various video characteristics impact encoders, such as the amount of action in a scene, grayscale ("black and white" movies) or color, and animated or live-action.

A scan of the SPEC 2006 floating-point programs presents modeling and simulation tools, equation solvers, a rendering engine and a speech-recognition program. Changing system parameters, data sizes, material properties, and/or the scenario presented by the input files should lead to the creation of collections of inputs, in the absence of real-world input sets. However, profile-guided optimizations are typically most beneficial to integer programs, and have far less impact on scientific codes.

In short, the burden of collecting a set of inputs for cross-validation does not appear to be significant in most cases, as represented by SPEC CPU 2006.

5. Conclusion

PDF and learning compiler techniques are a vibrant field of research that offer significant performance gains over uninformed compilation. However, these techniques require a more robust evaluation methodology than traditionally used for the performance evaluation of computer systems. Fortunately, the field of machine learning has already studied both the issues and complications of evaluating a learning system, and developed methodologies to evaluate these systems despite these complications. We propose a cross-validation approach to PDF evaluation that avoids the problem of overfitting the training data while providing statistical confidence measures for the performance results.

However, one large problem remains open. How should the sets of inputs used for performance evaluation be selected? In many cases, expert knowledge can be leveraged to select representative workloads of inputs. However, in other cases, such as long-running database applications, program usage patterns can change over time in unpredictable ways. While the proposed evaluation methodology does consider a range of inputs, those inputs must be

selected *a priori*, and the methodology may not predict program performance if program usage drifts in ways unanticipated during benchmark construction. Ideally, benchmark performance should predict performance in the field. After all, users only care about the performance they observe in their program.

PDF is useful when program behavior changes very slowly. At the other extreme, JIT technologies impose a run-time overhead but dynamically respond to program behavior at run time. In between is a spectrum of behavior change rates that require approaches that are more dynamic than PDF, but less dynamic than JIT. Some work with limited scope has been done in this intermediate space, such as dynamic prefetching [6]. Future research must continue to explore points in the dynamic optimization spectrum. Consequently, the need for input selection methodologies and evaluation techniques that consider both input sensitivity and performance stability will expand.

Acknowledgments

This research is supported by fellowships and grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Informatics Circle of Research Excellence (iCORE), the Canadian Foundation for innovation (CFI), and Alberta Ingenuity. Thanks to Dan Lizotte for discussions and advice related to machine learning.

References

[1] 11th computer olympiad. http://www.cs.unimaas.nl/Olympiad2006/, June 2006.

[2] 14th world computer-chess championship. http://www.cs.unimaas.nl/wccc2006/, June 2006.

[3] Swiss-prot protein knowledgebase. http://www.expasy.org/sprot/, November 2006.

[4] P. Berube. *Aestimo*: A feedback-directed optimization evaluation tool. Master's thesis, University of Alberta, October 2005.

[5] P. Berube and J. N. Amaral. Aestimo: a feedback-directed optimization evaluation tool. In *2006 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 251 – 260, Austin, Texas, March 2006.

[6] A. Das, J. Lu, H. Chen, J. Kim, P.-C. Yew, W.-C. Hsu, and D.-Y. Chen. Performance of runtime optimization on blast. In *Proceedings of the international symposium on Code generation and optimization*, pages 86–96, Washington, DC, USA, 2005. IEEE Computer Society.

[7] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism*, 5:1–33, 2 2003.

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, chapter 7, pages 214–221. Springer Series in Statistics. Springer, 2003.

[9] Intel Corporation. Intel C++ compiler options. ftp://download.intel.com/support/performancetools/c/linux/v9/copts_cls.pdf, 2006.

[10] International Business Machines. Detailed descriptions of the XL fortran compiler options. http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.xlf101a.doc/xlfcr/opts-details.htm.

[11] P. Kulkarni, S. Hines, J. Hiser, D. Whalley, J. Davidson, and D. Jones. Fast searches for effective optimization phase sequences. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 165–198, 2004.

[12] T. M. Mitchell. Does machine learning really work? In *AI Magazine*, volume 3, pages 11–20. The American Association of Artificial Intelligence, Fall 1997.

[13] Z. Pan and R. Eigenmann. Rating compiler optimizations for automatic performance tuning. In *ACM/IEEE Conference on High Performance Networking and Computing (SC04)*, pages 14–23, November 2004.

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

[15] S. Russell and P. Norvig. *Artifi cial Intelligence: A Modern Approach*, chapter 1. Prentice Hall, 2 edition, 2003.

[16] Standard Performance Evaluation Corporation. SPEC CPU2006. http://www.spec.org/cpu2006/, August 2006.

[17] M. Stephenson and S. P. Amarasinghe. Predicting unroll factors using supervised classification. In *International Syumposium on Code Generation and Optimization*, pages 123–134, 2005.

[18] R. Weicker and K. Dixit. (osgcpu-10955) re: Your question to SPEC about input data selection for benchmarks. Personal email correspondences, July 2004.