A Statistical Approach for the Analysis of the Relation Between Low-Level Performance Information, the Code, and the Environment

Nayda G. Santiago
Michigan State University
ECE Department
2120 Engineering Building
East Lansing, MI 48824, USA
santia11@msu.edu

Diane T. Rover
Iowa State University
Department of ECE
3227 Coover Hall
Ames, IA 50011, USA
drover@iastate.edu

Domingo Rodríguez
Univ. of Puerto Rico at Mayagüez
ECE Department
P. O. Box 9042
Mayagüez, PR 00681-9042, USA
domingo@ece.uprm.edu

Abstract

This paper presents a methodology for aiding a scientific programmer to evaluate the performance of parallel programs on advanced architectures. It applies well-defined design of experiments methods to the identification of relations among different levels in the process of mapping computational operations to high-performance computing systems. Statistical analysis is used for studying different factors that affect the mapping process of scientific computing algorithms to advanced architectures. In particular, a case study on the numerical solution of finite element methods for the analysis of conformal antennas for electromagnetic radiation applications was used to test the proposed methodology. The use of statistics for identification of relationships among factors has formalized the solution of the problem and this novel approach allows unbiased conclusions about results. Subset selection based on principal components was used to determine the subset of metrics required to explain the behavior of the system.

1. Introduction

Performance data analysis is integral to the process of tuning parallel applications to advanced architectures. The traditional approach for performance tuning is through the process of data collection, analysis, and code optimization. In this approach the application programmer needs to understand instrumentation, learn the appropriate tools, and interpret data and its relation to the code, in order to optimize the code or system configuration, accordingly. This is illustrated in Figure 1. This method is complex and prone to wrong interpretations [10]. Also, transformations applied to source code are hard to map to performance data [8]. We propose an alternative method that minimizes ambiguity when determining which factors to consider during a tuning process of a parallel application.

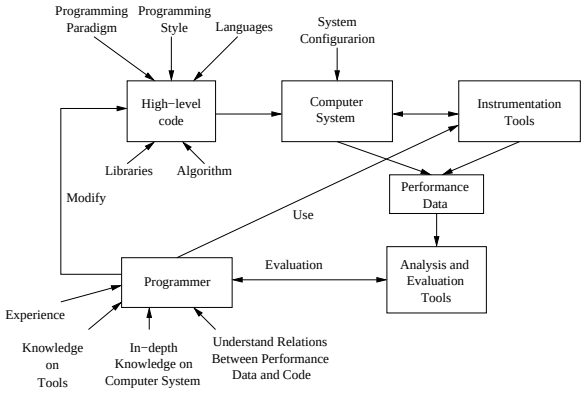

Figure 1. Analysis flow for tuning an application.

Some tools, such as Paradyn [7], take an automatic approach to determine whether there is a performance bottleneck and where to locate it. Most other tools take a different approach in what is called in statistics, *exploratory data analysis* (EDA). In this type of analysis, the calculation of simple statistics and graphical summaries provide the user an understanding of what information the data is conveying. No *a priori* knowledge about the data is used.

Complementary to EDA there is a method called *confirmatory data analysis* (CDA) where formal statistics is used to confirm or reject a hypothesis about the population under study. These methods have been used for a long time in areas such as biostatistics, economics, pattern recognition, and operational research. Coffin and Saltzman applied these traditional techniques to evaluate and compare optimization algorithms used in operational research [2]. This analysis allowed them to draw statistically sound conclusions about the algorithms. Sun *et al.* applied design of

experiments and ANOVA to evaluate memory hierarchies and understand their performance [12].

Certain combinations of factors such as programming style, language, compiler options, and algorithms will produce better performance results than others. In this work, we are presenting a methodology for obtaining information about how these factors affect performance for a specific application. This methodology is based on a combination of confirmatory data analysis statistics and exploratory data analysis and obtains sound conclusions about the effects of factors on the performance obtained.

We combined the use of design of experiments, analysis of variance (ANOVA), correlation, and subset feature selection, applied to performance data, to explain the behavior of the system and provide insight to the user on the relationship between high-level abstractions to low-level performance information. Figure 2 depicts this methodology. In this paper, we describe the analysis steps in detail. However, we expect most of the details to be hidden from the programmer as support for automation is developed and incorporated.

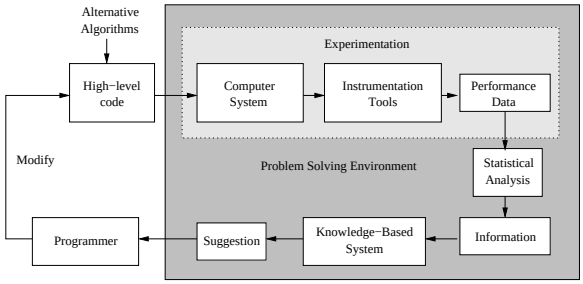

Figure 2. Proposed application tuning methodology.

Section 2 provides an overview of the methodology. In section 3, the evaluation method is demonstrated in a case study of an electromagnetics application for conformal antenna design. Sections 4 and 5 show the results and conclusions obtained.

2. Overview of Methodology

This work proposes a methodology for the analysis of performance data using a combination of CDA, EDA, and experimentation. EDA is characterized by utilizing no preliminary knowledge about the possible relations of variables under study and the use of statistics and graphical summaries to understand the information data is conveying. In CDA, formal statistical methods are used to confirm or reject a hypothesis about the population under study. Experimentation is used to collect unbiased data to confirm or reject the hypotheses.

There are four steps in the methodology. First, a *preliminary problem analysis* is done. Here we can visualize in general what is affecting performance and gather preliminary information. The second step is to *specify the experiment design* to collect enough unbiased information to be analyzed for establishing relationships. The third step is to *collect the data*. Finally, the last step is *data analysis*.

2.1. Preliminary Problem Analysis

A performance problem-solving process starts with the analysis of the problem specification. Information needs to be collected about the programmer's goal and both the performance problem and the application itself.

Once the application and performance goals are clear, the next step is to profile the code to identify possible functions to optimize. Analysis continues with the identification of possible factors affecting performance. These include environment factors, algorithms to solve those functions to optimize, and hardware specific factors. Once the factors are identified, a subset should be selected for the experiment, considering controllability, feasibility, practicability, and constraints.

2.2. Specification for the Experiment

The second step in the methodology is experiment specification. The theory of *design of experiments* allow us to take an objective approach in the experimentation process [9]. A well known model of the experimentation process is shown in Figure 3.

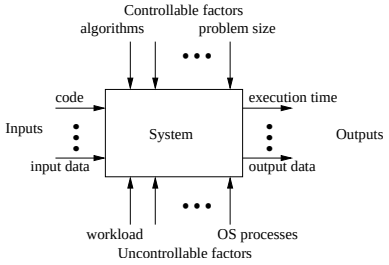

Figure 3. Model of an experiment.

Studying all possible factors and levels of these factors is an intractable problem. A level refers here to the different possible values of one factor considered in an experiment. In order to obtain the total number of experimental runs, it is necessary to calculate all possible assignment of factors when varying all at a time. Once a decision on the factors and levels is taken, the next step is to select the random order in which the experimental runs will be executed. Randomization is required to avoid the influence of uncontrollable factors in the outcome. We must also have at least two replicates of the experiment [9].

The effect of each factor is obtained through experimentation by the use of a factorial design. In this type of design, all combinations of all levels of all factors are tested, usually in a complete random order [5]. For practical considerations, in certain cases a completely random set of runs might not be easily implemented. A completely randomized run would imply that from run to run any factor may change. For most computer applications, this is impractical. For example, in our study, changing the problem size from experimental run to experimental run results in excessive experimentation time and limits our ability to automatically control experimentation. So a *split-split-plot* design was used. A *split-plot* design is a general case of a factorial design in which randomization is restricted. In this design, one factor is selected for a treatment. A treatment is a set of levels of controllable factors administered to an experimental run. The order in which the treatments will be applied to this factor is selected at random. Once this is fixed, a second factor is selected and, given the order for experimental runs selected for the first factor, randomization is done on the second factor. This could be repeated successively. When a third factor follows the same restrictions, this is called a *split-split plot* design [9]. A partial randomization of experiments causes a higher experimentation error so split-split plot is suggested only when a completely randomized design is not possible for practical reasons.

2.3. Data Collection

The data collection step is the only one determined particularly by the computer system, language, and tools used. This is due to the large variation of metrics available for different computer systems and at different levels. One group working towards standardization of performance metrics is the APART (Automatic Performance Analysis: Resources and Tools) group [4, 10]. Their work moves towards the formalization of the language and methods to present performance information and to identify the requirements for automatic performance analysis tools. APART workpackage 2 presents a set of metrics defined using ASL for determining some performance properties for shared memory, message passing, and high performance Fortran [4].

During this step, we identify which metrics are measurable for the paradigms and systems being used. Specifically, we identify the instrumentation tools that are available and the metrics that are measurable at the operating system, application, and hardware levels. Then from these, for a given paradigm, we select the APART-recommended set of metrics. Important metrics suggested by the application programmer should also be selected. Once a set of performance metrics is selected, instrumentation is activated to collect the data. Code is compiled and linked as needed, and performance data are collected during execution.

2.4. Data Analysis

After data collection, analysis begins, and the metric data are first formatted to support the statistical techniques. For one experiment, a matrix format is used. Each element of the matrix is either an average or absolute metric value. An average value, M_{avg}, is computed as the sum of all metric sample values divided by the number of samples, where the samples of the metric values are taken during execution time only. For example, *page faults per second* might be measured as an average. An absolute value, M_{abs}, is a metric whose value is obtained as a total at the end of execution time only. *Total execution time* is an example of an absolute metric. One experiment consists of R experimental runs in a predefined random order. This random order determines the precision obtained in the results. Let P denote the number of performance metrics measured during an experimental run. Let r denote the experimental run where $0 \leq r \leq R-1$ and p is the metric identification number where $0 \leq p \leq P-1$. This results in the following data format for one experiment:

$$X = \begin{bmatrix} M_k(0,0) & M_k(0,1) & \cdots & M_k(0,P-1) \\ M_k(1,0) & M_k(1,1) & \cdots & M_k(1,P-1) \\ \vdots & \vdots & \ddots & \vdots \\ M_k(R-1,0) & M_k(R-1,1) & \cdots & M_k(R-1,P-1) \end{bmatrix}$$

where $M_k(r,p)$ denotes average or absolute metric value for experimental run r and metric p, and k is either avg or abs. Each column of this performance data matrix contains the measurement of one performance metric over a set of experimental runs and each row contains information about one experimental run. Several statistical techniques may be applied to this matrix.

2.4.1. Correlation Matrix.

The correlation coefficient is a measure of the linear association between two variables. The correlation matrix is a two-dimensional array of correlations where all correlation coefficients are organized systematically. A sample autocorrelation matrix is computed using the formula $S = \frac{1}{R-1} D \widetilde{X}^T \widetilde{X} D$ where $\widetilde{X} = X - 1\overline{x}^T$ and 1 is a $1 \times p$ unit vector and $\overline{x}$ is a row vector containing the means of the columns of X. R is the number of experiments [11]. D denotes a diagonal matrix containing the inverse of the standard deviation of each metric. The value of each element (i,j) in the correlation matrix contains the correlation coefficient between metric i and metric j.

2.4.2. ANOVA.

Analysis of variance (ANOVA) is a statistical procedure for the analysis of the response of an experiment. We are using ANOVA to determine whether there is influence of any

of the factors on the result obtained for each performance metric. In ANOVA, the goal is to determine if there is an effect of different treatments on a population. In hypothesis testing, the hypothesis assumed to be true is called the *null hypothesis* and the contradictory hypothesis is called *alternate hypothesis*. The null hypothesis tested by ANOVA is that no factor will influence the solution and that there is no interaction between any factors. The probability of error by selecting an alternate hypothesis when the null hypothesis is true is called *type I error* and is denoted by α (also called *alpha value*). Once the alpha level for the test is selected, a set of test statistics are computed and a conclusion on whether the null hypothesis is probable or not is reached. In our case, ANOVA at α level 0.05 will be used to establish relationships among factors and performance metrics.

2.4.3. Multidimensional Data Analysis.

The multidimensional nature of the output performance metrics prompts us to identify mechanisms for data reduction and subset selection. Subset selection refers to the selection of the most independent columns of the matrix to explain the variability of results. Vélez and Jiménez show that the number of columns required for subset selection will be the same number of components that we should retain when performing principal component analysis (PCA) on the data to preserve the variability of the multidimensional data [13].

Now the question to answer is how many principal components should be retained to account for most of the variation in the data? There are three commonly used methods in multivariate analysis:

Scree test [6]. The eigenvalues of the correlation matrix of the data set are sorted in descendent order and plotted. The point where the curve flattens is selected as the cutoff point, and this is the number of principal components to select.

Cumulative Percentage of Total Variation [3]. The eigenvalues of covariance matrix of the data are computed. Each eigenvalue contributes to a percentage of the total variance. Those eigenvalues whose eigenvectors explain most of the variance are selected. A threshold of typically 95% of the total variance is used.

Eigenvalues greater than 1 [6]. The eigenvalues of the correlation matrix of the data computed and those eigenvalues greater than one are selected.

Once the number of metrics needed to explain the variability of the data is known, a subset selection method is used to choose important metrics based on a cost function. We suggest using independence of metrics as the cost function to explain the variability of the data since it is related to the amount of information contained in the performance data matrix. In the subset selection method suggested by Vélez and Jiménez [13], the criterion of independence between columns is used as a measure for subset selection. Those features that are most independent and explain the highest correlation are selected based on principal component analysis and singular value decomposition (SVD).

3. Case study - Conformal Antenna Design

We have selected a case study of an application in the area of finite elements methods for conformal antenna analysis. This code implements an iterative solver whose kernel is a matrix-vector multiply of dense matrices and is representative of the types of workload in this area. We used experimental design techniques to determine how low-level performance information is affected by the code, problem size, and compiler options. This section introduces and demonstrates the methodology in the context of the conformal antenna design case study.

3.1. Preliminary Problem Analysis

The performance objective is to improve the execution time of the antenna analysis code. The code uses a biconjugate gradient iterative solver to find the solution. The goal is to parallelize the code and to reduce the execution time while keeping the memory requirements as low as possible due to the large matrices involved in the computation.

The original serial code was profiled and, not surprisingly, 84% of the time was spent in a dense matrix-vector multiplication routine and other routines were accounting for 3% or less of the total execution time each. Therefore, efforts concentrated in optimizing this dense matrix-vector multiplication routine. Several different dense matrix-vector multiplication routines were tested and problem sizes were changed by modifying the physical specifications of the antenna.

The experiments were done on a quad-processor Sun Enterprise 450 Server running Solaris 5.7. This server is a shared-memory, symmetric multiprocessor system (SMP). Each processor is an UltraSparc^{TM} II running at 400MHz with 2MB of local, high-speed external cache memory. We used OpenMP directives for code parallelization with the Forte Fortran HPC 6 Fortran compiler and Guidef77 3.9 parallelizing compiler.

3.2. Specification for the Experiment

The inputs to our system are the application code and data. The outputs are the matrices containing the different metrics to measure performance. Controllable factors in the experiment are problem size, algorithm, compiler options,

and sampling rate of the metrics. Among uncontrollable factors we consider environment variables and workload.

We investigated a comprehensive set of performance factors and determined that an observable, controllable and measurable set includes problem size, dense matrix-vector multiplication algorithm selection, and compiler options. The set of factors and levels in this experiment is shown in Table 1. Since the used compiler generates a different executable with each permutation of flags, the effect of permutations was also considered.

Table 1. Factors and levels in experiments.

Factors	Number of Levels
Problem Size	3
Compiler Options	13
Algorithms	2
Number of repetitions	3
Total number of experimental runs	234

Figure 4 shows a graphical description of a block of our split-split-plot design. A block refers to a replicate or repetition of the basic experiment. In this figure, a block in the design is divided into whole plots where the the problem size (1, 2, and 3) was selected at random. The subplot factor is the matrix multiplication algorithm (A and B). Then sub-subplots will contain the compiler options (a - m) that were tested randomly.

2	1	3
A b,l,a,...,h	B d,c,g,...,b	B a,i,d,...,c
B j,h,e,...,c	A k,b,g,...,a	A m,a,e,...,h

Figure 4. Example of one block for our split-split plot design.

Three replicates of the basic experiment were done. The number of iterations for obtaining the solution of the iterative solver has been fixed to remove the impact of reduced matrix conditioning.

3.3. Data Collection

The antenna code runs in two modes: model generation mode and solver mode. The first mode generates the matrices used in the computation and the second mode finds a solution for the antenna analysis. Running the code under the model generation mode, matrices for a given problem size are generated. Then one matrix-vector multiplication algorithm is selected and 13 experimental runs, each with a different compiler option, are set up for one batch of runs using the same problem size. Here the code is running in solver mode. A crontab file sequentially starts all experimental runs. Instrumentation and application code run simultaneously.

3.4. Data Analysis

Once we obtained the metrics, they were placed in the matrix form discussed in section 2.4 and its correlation matrix was computed. The most correlated metrics with execution time were identified.

Analysis of variance (ANOVA) at α level 0.05 was done for each set of metrics obtained. Then the methods discussed in section 2.4.3 were used to determine how large the set of important metrics should be. Since each method may give a different set size, the largest size value was used to avoid not having enough metrics. The SVD method described in [13] was used to obtain the final set of metrics.

4. Results

The results from two different experiments done to test the proposed methodology are shown in this section.

4.1. Experiment with parallel implementation

In this experiment, our application was parallelized using OpenMP constructs. Two different algorithms for matrix-vector multiplication were used with three different compiler flags and three different problem sizes. Problem sizes were varied by changing the physical specifications of the antennas under study.

Those metrics most correlated with execution time, using a threshold of correlation higher than 0.9, are shown in table 2, where the correlation was negative in all cases. Negative correlation is interpreted as follows: execution time increases when the metric value decreases.

Analysis of variance (ANOVA) at significance level $\alpha = 0.05$ was done on these metrics to establish the effect of factors. Table 3 shows ANOVA results for those metrics obtained in table 2.

We proceeded to perform a multidimensional analysis on the data. First we want to obtain the number of metrics required for preserving most of the information on the data. When we used the three different criteria for finding the number of metrics required for keeping most of the variance, we found that only three metrics were selected. Analyzing the data in detail, we noticed that principal component analysis was very biased towards the data with the largest values. This is a well known characteristic of PCA

Table 2. Metrics with largest correlation with execution time.

Rank	Label	Description
1	lwrit/s	Accesses of system buffer cache to write
2	lread/s	Accesses of system buffer cache to read
3	c0t0d0/wps	Write per second per disk
4	c0t0d0/util	Percentage of disk utilization per disk
5	disk/s0	Disk operations per second
6	page/mf	Minor faults in units per second
7	vflt/s	Address translation page faults per second

[3] and can be solved by normalizing the data. We normalized the data using the Euclidean norm and then proceeded with the analysis.

Figure 5 shows an example of a plot of the eigenvalues of the correlation matrix to use *scree test* and the *greater-than-one criteria*. Notice the change in the slope of the curve at five eigenvalues and then at eight eigenvalues. Scree test might have two or three inflection points in the curve and this is one of the cases. Notice also that only nine eigenvalues are greater than one. Table 4 show how many metrics should be kept to preserve the variability of the performance metrics outcome, according to the three methods explained in section 2.4.3.

Table 5 shows those metrics selected by the method for this experiment. These metrics describe activity which experts usually look for when tuning a program: *paging activity, cpu utilization, memory faults, and virtual memory statistics*.

Table 6 shows ANOVA results for those metrics.

Table 3. ANOVA on the metrics presented in table 2.

Factor	Metrics affected by the factors
Size (S)	execution time, disk/s0, page/mf, vflt/s
Algorithm (A)	execution time, lwrit/s, lread/s, c0t0d0/wps, c0t0d0/util, disk/s0, page/mf, vflt/s
Compiler Option (C)	execution time, lwrit/s, lread/s, c0t0d0/wps, c0t0d0/util, disk/s0, page/mf, vflt/s

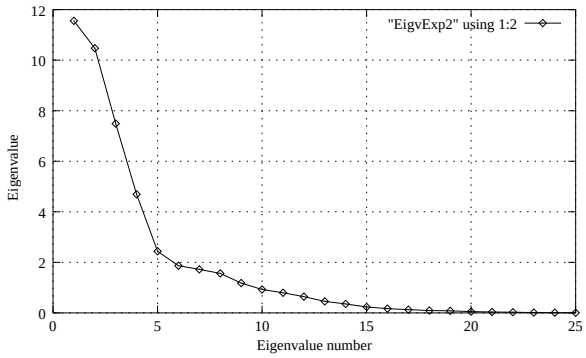

Figure 5. Eigenvalues of correlation matrix.

We can notice that cpu context switches is affected by all three factors.

4.2. Experiment with serial implementation

In this experiment, our application was using the same basic algorithms as in the previous experiment, but running serially. Other factors remain the same.

The metrics highest correlated with execution time using a threshold of correlation higher than 0.9 are shown in Table 7.

Table 8 shows ANOVA for these five metrics.

Using the method presented in [13] and the results from Table 4, those metrics shown in Table 9 were obtained as the most relevant ones. These metrics describe buffer and paging activity, virtual memory statistics, and cpu utilization.

Table 10 shows ANOVA results for these metrics.

4.3. Discussion

Results have led to several interesting findings. Those metrics with highest correlation with execution time will allow us to look for possible places where to improve the code. These are not necessarily the same metrics which will

Table 4. Number of features to select.

Test	Experiment with parallel implementation	Experiment with serial implementation
Scree test.	8	6
Cumulative percentage (95%).	9	7
Greater than 1.	9	6
Max. of the three methods.	9	7

Table 5. Metrics with highest information content.

Item	Name	Description
1	memory/free	Usage of virtual and real memory. Free size of the free list (Kbytes).
2	pflt/s	Page faults from protection errors per second (illegal access to page).
3	page/re	Paging activity in units per second. Page reclaims.
4	c0t1d0/wps	Writes per second per disk.
5	%wio	Portion of time running idle with some process waiting for block I/O.
6	page/sr	Paging activity in units per second. Pages scanned by click algorithm.
7	page/pi	Paging activity in units per second. Kilobytes paged in.
8	page/po	Paging activity in units per second. Kilobytes paged out.
9	faults/cs	Trap/Interrupt rates per seccond. CPU context switches.

Table 6. ANOVA on the metrics shown in table 5.

Factor	Metrics affected by the factors
Size (S)	faults/cs
Algorithm (A)	faults/cs
Compiler Option (C)	memory/free, page/po, faults/cs

Table 7. Metrics with largest correlation with execution time.

Rank	Label	Description
1	c0t0d0/wps	Writes per second per disk
2	disk/s0	Disk operations per second
3	lwrit/s	Accesses of system buffer cache to write
4	c0t0d0/util	Percentage of dik utilization per disk
5	lread/s	Accesses of system buffer cache to read

retain the largest amount of information about the status of the system. We have to take into account that information is inversely proportional to probability of occurrence and those metrics which are highly correlated carry less information than non-correlated metrics.

Those metrics highly correlated with execution time are

Table 8. ANOVA on the metrics presented in table 7.

Factor	Metrics affected by the factors
Size (S)	None
Algorithm (A)	lwrit/s, c0t0d0/util, lread/s
Compiler Option (C)	lwrit/s, c0t0d0/util, lread/s

Table 9. Metrics with highest information content.

Item	Name	Description
1	atch/s	Page faults per second that are satisfied by reclaiming a page currently in memory (attaches per second).
2	pflt/s	Page faults from protection errors per second (illegal access to page).
3	bread/s	Reads per second of data to system buffers from disk.
4	memory/free	Usage of virtual and real memory. Free size of the free list (Kbytes).
5	pgin/s	Page-in requests per second.
6	cpu/wt	Report the percentage of time the system has spent waiting for I/O.
7	execution time	Total execution time.

Table 10. ANOVA on the metrics selected by SVD.

Factor	Metrics affected by the factors
Size (S)	execution time
Algorithm (A)	cpu/wt, execution time
Compiler Option (C)	atch/s, memory/free, cpu/wt, execution time.

very similar for both experiments, specially those related to buffer activity, I/O, disk operation. This might be caused by the nature of our application which is very demanding in terms of I/O and memory access. Buffer activity, I/O, and paging activity are the activities most correlated with execution time. But this buffer cache activity is related to disk access directly since the buffer cache under Solaris 5.7 is used to cache inode, indirect block, and cylinder group related disk I/O only [1]. Analysis of means of this variable shows that algorithm B for matrix-vector multiplication causes a much lower buffer activity for read and for write than algorithm A. Algorithm A refers to a matrix-vector multiplication algorithm where we tried to minimize thread interaction in OpenMP by making the loops as independent as possible. Algorithm B modifies algorithm A by splitting loops

into smaller ones by removing the if condition showing in algorithm A. This splits the matrix by opposite diagonal elements. Compiler options one and three have much smaller buffer activity than other compiler options. An analysis of means on execution time showed that compiler options one and three also resulted in the longest execution times.

One analysis that we completed after looking at the results was to select a subset of the data to analyze effects caused by the compiler options. We divided the data in two mutually exclusive sets, one with the `-fast` flag and one without it. The statistical analysis shows that no significant difference exists between all compiler options without the `-fast` flag and also no significant difference exists between all compiler options with the `-fast` flag in terms of the effect on execution time. This is shown in Table 11. When we studied the effects of permutations on the compiler options flags we found that they cause no significant effect on the outcome.

Table 11. ANOVA studying the effect of the `-fast` flag. Main effects: S - Problem Size, A - Algorithm, and C - Compiler Options. *Yes* means there is an effect and *No* that there is no effect.

	Parallel experiment			Serial experiment		
Flag	S	A	C	S	A	C
No fast	Yes	Yes	No	Yes	No	No
Only fast	Yes	Yes	No	Yes	Yes	No

When we analyzed the same application with the serial code, we found that paging activity, buffer activity, and cpu utilization contain the most relevant information on the status of the system.

5. Conclusions

A methodology based on an unified view of performance analysis, statistics, and multidimensional data analysis has been presented. We have used a powerful statistical tool to identify correlations between low-level performance information and high-level code abstractions. We are interested in calling other researcher's attention in applying these techniques to their applications and platforms. The information collected about algorithms or compiler options will aid the application programmer in making decisions about their code. This approach will complement traditional exploratory data analysis.

Future work will include the use of additional techniques for feature selection and the design of a knowledge based system for providing feedback to the programmer. We will also begin a study on cluster architectures [14], using our methodology to evaluate communication characteristics of large-scale scientific applications.

6. Acknowledgments

This work was supported by the following grants: NSF EIA-9700732, NSF ACI-9624149, and NSF EIA-9977071. Special thanks to Leo Kempel and the Electromagnetics Laboratory at Michigan State University for providing the application code and to Vijay S. Kesavan for his assistance.

References

[1] A. Cockcroft and R. Pettit. *Sun Performance and Tuning: Java and the Internet*. Sun Microsystems Press, 2nd edition, 1998.

[2] M. Coffin and M. J. Saltzman. Statistical analysis of computational tests of algorithms and heuristics. *INFORMS J. Comput.*, 12(1):24 – 44, Winter 2000.

[3] W. R. Dillon and M. Goldstein. *Multivariate Analysis: Methods and Applications*. John Wiley, 1984.

[4] T. Fahringer, M. Gerndt, B. Mohr, F. Wolf, G. Riley, and J. L. Träff. Knowledge specification for automatic performance analysis APART. Technical Report FZJ-ZAM-IB-2001-08, Cent. Inst. for Appl. Math., Res. Centre Jülich, Aug. 2001.

[5] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, Inc., 1991.

[6] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, Inc., 1986.

[7] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *Computer*, 28(11):37 – 46, Nov. 1995.

[8] B. Mohr, A. D. Malony, S. Shende, and F. Wolf. Design and prototype of a performance tool interface for OpenMP. In *Proc. 2nd LACSI Symposium*, Oct. 2001.

[9] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, Inc., 1997.

[10] G. D. Riley and J. R. Gurd. Requirement for automatic performance analysis APART. Technical Report FZJ-ZAM-IB-9919, Cent. Inst. for Appl. Math., Res. Centre Jülich, Nov. 1999.

[11] A. Ruiz and P. E. L. de Teruel. Nonlinear kernel-based statistical pattern analysis. *IEEE Trans. Neural Networks*, 12(1):16 – 32, Jan. 2001.

[12] X.-H. Sun, D. He, K. W. Cameron, and Y. Luo. Adaptive multivariate regression for advanced memory system evaluation: Application and experience. *Performance and Evaluation: An International Journal*, 45(1):1 – 18, 2001.

[13] M. Vélez-Reyes and L. O. Jiménez. Subset selection analysis for the reduction of hyperspectral imagery. In *Proc. IGARRS '98*, pages 1577 – 1581 Vol. 3, 1998.

[14] J. S. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *Proc. IPDPS 2002*, Apr. 2002.