Soft Computing Approach to Performance Analysis of Parallel and Distributed Programs *

Hong-Linh Truong and Thomas Fahringer

Institute for Computer Science, University of Innsbruck
Technikerstrasse 21A, A-6020 Innsbruck, Austria
{truong,tf}@dps.uibk.ac.at

Abstract. This paper describes a novel approach to performance analysis for parallel and distributed systems that is based on soft computing. We introduce the concept of performance score representing the performance of code regions that is based on fuzzy logic. We propose techniques for fuzzy-based performance classification. A novel high-level query language is designed to support the search for performance problems by using linguistic expressions. We describe a fuzzy-based bottleneck search, a performance similarity measure for code regions and experiment factors, and performance similarity analysis. Our approach focuses on the support of making soft decisions on evaluation, classification, search and analysis of the performance of parallel and distributed programs.

1 Introduction

Recently, performance analysis community has focused on developing performance tools for parallel and distributed programs that are capable of supporting semi-automatic performance analysis, dealing with large performance data sets, and analyzing multiple experiments. However the development of automatic and intelligent performance analysis is still at an early stage. Current techniques in existing performance analysis tools have mainly been used to process the performance data that are in the form of precise numerical data. Firstly, these techniques always apply exact analysis methods that result in hard conclusions about performance characteristics of applications. Secondly, existing performance tools interact with the user through complex numerical values and visualizations which are not easily understood by the user. Thirdly, in the real world we largely rely on domain expertise and user-provided inputs as parameters to control the performance analysis and tuning. Such expertise and inputs may be inexact and uncertain. However, existing performance tools do not support the specification and the control of approximate and inexact parameters in data analysis techniques, in other words, these tools do not provide a mechanism to make soft decisions.

The recent emerging *soft computing* [1], however, presents another way for evaluating and analyzing data that is based on the concept of *soft, inexact, uncertainty*. Soft computing aims to support imprecision, uncertainty and approximate reasoning [1].

* The work described in this paper is supported in part by the Austrian Science Fund as part of the Aurora Project under contract SFBF1104 and by the European Union through the IST-2002-511385 project K-WfGrid.

In this paper we present a new approach to the performance analysis that we call the *soft performance analysis*. In this approach, well known soft computing techniques such as fuzzy logic (FL), machine learning (ML) concept, and the combination of FL and ML are studied and developed for performance analysis of parallel and distributed programs. We introduce the concepts of performance score and performance similarity measure. Employing these concepts, we develop several soft techniques and methods for performance analysis such as fuzzy-based performance classification, performance search, similarity analysis, etc.

The rest of this paper is organized as follows. Section 2 outlines the so-called soft performance analysis. Section 3 presents a few preliminaries. We introduce the concept of performance score and performance similarity measure in Section 4 and Section 5, respectively. We describe soft techniques for performance analysis including fuzzy-based performance classification, query language, fuzzy-based bottleneck search and performance similarity analysis in Section 6. Section 7 discusses the related work. Section 8 gives conclusions and the future work.

2 Soft Performance Analysis

Existing performance analysis tools are based on *hard* computing model that is based on binary logic and crisp systems. For example, to classify the performance performance analysis tools normally use a *characteristic function*. That is, given a performance metric and a set of *performance characteristic term*, e.g., *poor, medium* and *good*, each term represents a performance class and is associated with a data set, the performance of a code region is classified according to characteristic terms by using a characteristic function. However, such classification is in binary form, e.g., a performance of the code region is either *good* or not, because the hard computing model does not accept imprecision and uncertainty. Since approximate search, classification and reasoning are not possible, the cycle of finding performance patterns in a large set of performance data has been lengthened because, in the real world, the boundaries between performance classes, performance search constraints, etc., are not clearly seen, thus, exact methods may not yield the expected results. Moreover, current tools focus on supporting the performance analysis through statistical graphics which are not well suited for processing large performance datasets. In practice, both performance data and expertise used in performance analysis domain can be uncertain. For example, in the case of performance classification, performance of code regions is classified into *good*, but depending on the degree of *good* the performance of code regions can be considered as *little good*, *fairly good* or *very good*. When we are not sure about performance data and expertise, we may accept some degrees of uncertainty and approximate in our analysis techniques.

To address the above-mentioned issues, we investigate performance analysis techniques that are based on soft computing. The soft performance analysis we propose aims to develop techniques for performance tools that can (i) extract useful performance information from large, dynamic and multi-relational performance measurement sources, (ii) support the specification and control of approximate and inexact parameters, commands and requests in existing performance analysis tools, and (iii) interact with the user through high level notions and concepts expressed in linguistic expressions.

We outline the approach as follows. Firstly, fuzzy logic (FL) can help representing and normalizing quantitative data. We can represent *performance score* of metric values by using fuzzy set (FS). By employing the concept of performance scores, we can develop several techniques that support soft, inexact and uncertainty in performance analysis. The application of FL theory also involves the concept of linguistic variables and the use of linguistic variables is particular useful for the end-user because humans employ mostly words in computing, as presented in the concept of computing with words [2]. Therefore, by using FL, performance tools can provide a way to perform the analysis and to interpret performance results with linguistic terms. Secondly, when processing large and diverse performance data, information about performance summaries, similarities and differences of data items in that data become more important as we cannot examine each data items in detail. Similarity measure techniques can be exploited to reveal the performance similarities and differences. ML techniques [3] can be utilized to discover patterns in very large performance datasets. For example, machine learning is combined with fuzzy computing to provide fuzzy clustering for performance data. Due to the space limit, this paper presents only a few points of our approach, focusing on FL and performance similarity techniques. More detail of soft performance analysis can be found in [4].

3 Preliminaries

3.1 Performance Experiment Data

A program contains a set of instrumented code regions. Performance data collected in each experiment of the program is organized into a *performance experiment data*. An experiment is associated with a set of processing units. A processing unit pu is a triple (n, p, t) where n, p and t are computational node, process identifier and thread identifier, respectively. A region summary rs is used to store performance metric records of executions of a code region cr in a processing unit pu. A performance metric record pm is represented as a tuple (m, v) where m is the metric name and v is the metric value. We denote $rs(m)$ as the value of performance metric m stored in region summary rs.

We use performance data obtained from experiments of three Fortran applications named 3DPIC (MPI program), LAPW0 (MPI program) and STOMMEL (mixed OpenMP and MPI program). All experiments are conducted on a cluster of 4CPU SMP nodes using MPICH library for Fast-Ethernet 100Mbps and Myrinet.

3.2 Representing Performance Characteristics under Fuzzy Logic Theory

An FS is used to map metric values onto membership values in the range $[0, 1]$. An FS is expressed as a set of ordered pairs $FS = \{(v, \mu(v))|v \in U\}$ where $\mu(v)$ is the membership function determining the degree of membership of v, and U is the universe of discourse of v. Let v be a metric value with the universal of discourse U. U is characterized by a given set of *performance characteristic terms* $T = \{t_1, t_2, \cdots, t_n\}$; performance characteristic terms are linguistic terms such as *poor, medium* and *high*. Each t_i is associated with a membership function $\mu_i(v)$ which determines the membership of v in t_i. v can be classified according to these terms. A *modifier* (e.g. *slightly*) is an operation that modifies a performance characteristic term (e.g. *bottleneck*). The modification results in a new fuzzy set represented by a new phrase (e.g. *slightly bottleneck*). In our experiments, we use the NRC-IIT FuzzyJ Toolkit [5] for fuzzy computing.

4 Performance Score

When evaluating and comparing performance of code regions most existing performance tools are normally based on quantitative measurement values and do not employ quantization or normalization techniques to evaluate multiple metrics. We present the concept of *performance score* which is used to evaluate the performance of a code region within a base, e.g. the parent code region or the whole program. The concept is based on (i) a set of selected performance metrics characterizing the performance of the code region, and (ii) a weight set representing the significance of performance metrics. Given a code region cr, let rs be the region summary of cr with a set of n performance metrics $\{m_1, m_2, \cdots, m_n\}$. Suppose the number of performance metrics measured is the same for every code regions. rs can be represented in n dimensional space. Let $v_i = rs(m_i)$ be the value of metric m_i in rs and let s_i be a score that represents the performance of rs with respect to metric m_i. We compute s_i as follows

$$s_i = \mu_i(v_i), \mu_i(v) : [0, V_{m_i}] \rightarrow [0, 1] \tag{1}$$

where $\mu_i(v)$ is the membership function determining the performance score, and V_{m_i} is the maximum observed value of m_i. V_{m_i} is dependent on the level of code region analysis. For example, if we analyze performance scores of rs with its parent rs_{parent} as the base, $V_{m_i} = rs_{parent}(m_i)$.

The value of s_i is in the range $[0, 1]$; 0 means the lowest score, 1 means the highest score. A higher performance score might be used to imply a higher performance or to indicate a lower significant impact. The exact semantics of the value of the performance score is defined by the specific implementation. As a result, performance scores can be used in various contexts such as to indicate (i) a significant impact level: the higher a performance score is, the higher impact the code region has, or (ii) a severity, the higher a performance score is, the more severe the core region is. There are several ways to select $\mu(v)$, depending on the specific analysis and approximate model used. The most simple way is to define the membership function μ as $\mu(v_i) = \frac{v_i}{V_{m_i}}$ which assumes that the score is based on linear model. We can choose trapezoid, S-function, Z-function, triangle, etc., and tool-defined function for $\mu(v)$.

Each rs is associated with a vector of performance scores $\vec{s}$. However, we may only select a subset of $\vec{s}$ as metrics to represent the performance of the code region. Like quantitative measurement values, we can compare two performance scores of two different metrics. However, because performance scores are normalized values, we can aggregate performance scores $\vec{s}$ of rs into a single score by using the overall weighted average (OWA) operator. Let $\{s_1, s_2, \cdots, s_n\}$ be performance scores of rs and $W = \{w_1, w_2, \cdots, w_n\}$ be the set of weights. w_i is a weight factor associated with metric m_i. The aggregate performance score for $\vec{s}$ may be computed as follows

$$OWA(\vec{s}) = \frac{\sum_{i=1}^{n} (|s_i w_i|)}{\sum_{i=1}^{n} w_i} \tag{2}$$

For the sake of simplicity, normally $w_i \in (0,1)$ and $\sum_{i=1}^{n} w_i = 1$. OWA score is particular useful for support of decision making in performance analysis and tuning because very often we have to decide which are the focused metrics of the code regions that should be tuned and optimized in order to achieve a better performance. Hence we use the notation (m_i, w_i) to denote m_i with its associated weight w_i.

We use performance score in ranking analysis, fuzzy C-means clustering, fuzzy rules, and similarity analysis. The former three analyses are covered in [4].

5 Performance Similarity Measure

Most existing performance tools employ numerous displays, e.g., process time-lines and histograms, to compare performance measurements and visualize that measurements. Those displays are crucial but the user has to observe the displays and perceive the similarity and the difference among these values. Moreover, it is difficult to compare multivariate data through visualization. We propose methods to compute the *performance similarity measure* which can be used as a metric to indicate the performance similarity among code regions and among experiment factors. Formally, let o_i and o_j be objects, a similarity measure is a function $sim(o_i, o_j) \rightarrow [0,1]$ that compares o_i with o_j where 0 denotes complete dissimilarity and 1 denotes complete similarity. Performance similarity measure can help uncovering similar/dissimilar performance patterns among code regions, e.g., for making decisions in dynamic performance tuning [6].

5.1 Similarity Measure for Code Regions

Let rs_i and rs_j be region summaries of cr. Let s_{il} and s_{jl} be performance score of rs_i and rs_j with respect to metric m_l, respectively. We use Equation 1 to compute s_{il} and s_{jl}. The performance similarity measure $sim_{ij}(rs_i, rs_j)$ is defined as follows

$$sim_{ij}(rs_i, rs_j) = 1 - d_{ij}, d_{ij} = \sqrt{\sum_{l=1}^{n} (|s_{il} - s_{jl}|^2)} \tag{3}$$

where d_{ij} is the distance measure between rs_i and rs_j; d_{ij} is computed based on Euclidean distance. Note that we can use other distance functions, e.g., Minkowski, Manhattan, Correlation and Chi-square, and can use weight factors associated with metrics.

To determine the performance similarity among executions of code regions across a set of experiments, we use Equation 3 to measure the performance similarity. Given a code region cr and a set of experiments $\{e_1, e_2, \cdots, e_n\}$. Let rs_i be region summary of cr in experiment e_i. We compute similarity measure $sim(rs_1, rs_i)$, $i : 2 \rightarrow n$ by using various membership functions. Given metric m_i, when determining performance score, the maximum observed value V_{m_i} is obtained from e_1 which is the base experiment.

5.2 Similarity Measure for Experiment Factors

Experiment factors which can be controllable, e.g. problem size, the number of CPUs and communication libraries, or uncontrollable such as CPU usage, have significant impact on the performance of the applications. Without considering the similarity between experiment factors, it is difficult to explain cases in which the performance of code regions is not similar because the experiment factors can be different. Therefore, initially we try to address this problem by measuring similarity between controllable factors.

Let $sim_f(e_i, e_j)$ be similarity measure for factor f between experiments e_i and e_j. Given a set of controllable factors $F = \{f_1, f_2, \cdots, f_n\}$, similarity measure is computed for each factor $f_i \in F$. There is no common way to compute sim_f as a controllable factor and its role depend on each experiment. The objective of our analysis is to find out the relationship between the performance similarity of the code regions, sim_o (e.g. $sim(rs_i, rs_j)$), and sim_{f_i}. Naturally we expect that the similarity measures of the controllable factors of two experiments and the similarity measures of the performance of these experiments behave in a similar fashion, e.g. if the controllable factors are very similar then the performance of experiments should be very similar.

6 Soft Techniques for Performance Analysis

6.1 Performance Classification

Performance classification classifies the performance of code regions according to performance characteristic terms. Formally, given a metric value v and a set of performance characteristic terms $T = \{t_1, t_2, \cdots, t_n\}$, v are classified according to that terms. In existing performance tools, the classification gives a binary result: v belongs to only one $t_i \in T$, with no degree of membership. Conversely, the fuzzy-based classification determines the degree to which v fits into t_i, for all $t_i \in T$.

To classify performance of code regions, we firstly define a set of performance metric terms for each performance metric m by partitioning the universal of discourse of metric m into segments and each segment is described by a performance metric term which is associated with a FS. Performance characteristic terms can be defined based on training data. After membership functions are determined, the membership degree of v is computed based on quantitative value v of m.

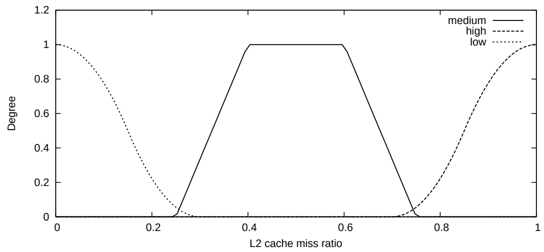
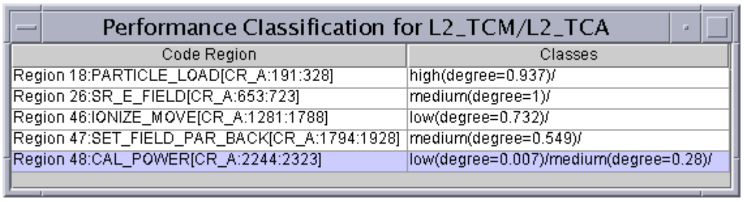


Fig. 1. Performance characteristic terms *low, medium, high* with their associated fuzzy sets.

To demonstrate this analysis, we classify code regions of 3DPIC application executed on 4 processors according to performance characteristic terms $T = \{low, medium, high\}$ representing the L2 cache miss ratio. Three FSs Z-function, trapezoid and S-function are associated with *low, medium*, and *high* term, respectively, as shown in Figure 1. We then conduct the classification with a few selected code regions. Figure 2 presents the result with five selected code regions. As shown in Figure 2, the code region `PARTICLE_LOAD` has *high* L2 cache miss ratio. However, code region `CAL_POWER` is member of both *low* and *medium*.

New performance characteristic terms can also be built by combining existing ones with modifiers. For example, we can classify code regions according to *very low* L2



Performance Classification for L2_TCM/L2_TCA

Code Region	Classes
Region 18:PARTICLE_LOAD[CR_A:191:328]	high(degree=0.937)/
Region 26:SR_E_FIELD[CR_A:653:723]	medium(degree=1)/
Region 46:IONIZE_MOVE[CR_A:1281:1788]	low(degree=0.732)/
Region 47:SET_FIELD_PAR_BACK[CR_A:1794:1928]	medium(degree=0.549)/
Region 48:CAL_POWER[CR_A:2244:2323]	low(degree=0.007)/medium(degree=0.28)/

Fig. 2. Membership in {*low, medium, high*} L2 cache miss ratio for selected code regions of 3DPIC.

cache miss ratio; the term *very* is a fuzzy modifier. The use of modifiers allows us to extend and enhance the description of performance characteristic terms.

6.2 Fuzzy Query for Performance Search

The fuzzy-based approach offers the possibility of search of performance data with words. Fuzzy-based search that uses linguistic expressions has been widely employed in database systems, information retrieval, etc., but not in existing performance tools.

$\langle Statement \rangle$ `::=` $\langle Expr \rangle$ | $\langle Statement \rangle$ `OR` $\langle Expr \rangle$
$\langle Expr \rangle$ `::=` $\langle Term \rangle$ | $\langle Expr \rangle$ `AND` $\langle Term \rangle$
$\langle Term \rangle$ `::=(METRIC is` $\langle F\_Expr \rangle$`)`

Fig. 3. Top-level syntax of PERFQL.

We propose a fuzzy-based query language for search of performance data. Queries are constructed based on fuzzy modifiers, *AND* and *OR* operators, and performance characteristic terms. Figure 3 presents the top-level syntax of our PERFQL (Performance Query Language based on fuzzy logic). `METRIC` is a metric name or a *metric expression*. A metric expression consists of operands and +, -, *, / arithmetic operators; operands are metric names. F_Expr describes the syntax of generic linguistic expressions (see [5] for the syntax). These expressions are constructed from performance characteristic terms and modifiers. For example, the following query can be used to find code regions which have high wallclock time and poor L2 cache miss ratio: `"(wtime is HIGH_EXECUTION_TIME) AND (`$\frac{L2\_TCM}{L2\_TCA}$ `is POOR_CACHE_MISS)"` , where `HIGH_EXECUTION_TIME` and `POOR_CACHE_MISS` are performance characteristic terms.

PERFQL allows the user to easily define queries for search of performance data by using words, not numerical expressions. Thus, it is easy to be understood and interpreted by the user. Moreover, fuzzy-based queries enable approximate search thus interesting performance data which is slightly less or greater than the crisp condition can be easily obtained.

6.3 Fuzzy Approach to Bottleneck Search

There are several tools supporting bottleneck search, e.g., [7, 8]. These tools, however, support crisp-based searching as the search is conducted by checking crisp threshold. Given a performance metric, a threshold is pre-defined. During the search, the performance metric is evaluated against the threshold, and when the performance metric exceeds the threshold, a bottleneck is assumed to exist in the code region. There are two drawbacks of current crisp search strategy. Firstly, the search does not give the degree of severity of the bottleneck, e.g. *extremely* or *slightly* bottleneck. Secondly, there is no support to specify inexact bottleneck search statements such as *negligible bottleneck*. These statements are important as the threshold, by nature, is not an exact value.

We propose fuzzy-based bottleneck search that addresses the above-mentioned drawbacks. Figure 4 outlines the fuzzy-based bottleneck search. Given a threshold, we can use FSs to represent the severity of bottleneck and the negligible bottleneck range besides the FS representing the bottleneck threshold. For example, in Figure 4 we define a Pi-function FS used to check the negligible (close to) bottleneck points and S-function FS used to check the severity of bottleneck. When searching the bottleneck points, the value of metric used in bottleneck search is evaluated against these FSs. Not only we can locate bottleneck points as usual but also we can provide the severity of bottleneck, and are able to find negligible bottleneck points.

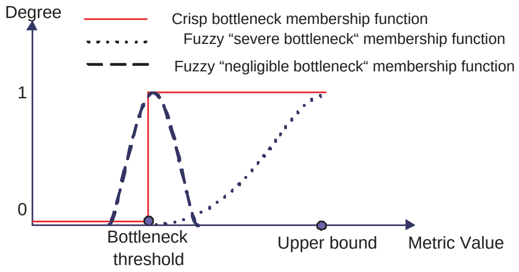
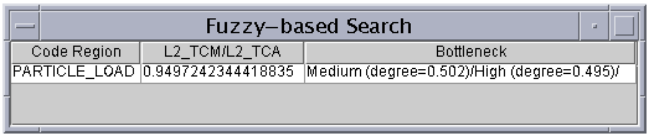


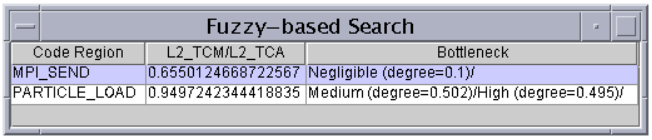
Fig. 4. Fuzzy vs crisp bottleneck search.



Fuzzy–based Search

Code Region	L2_TCM/L2_TCA	Bottleneck
PARTICLE_LOAD	0.9497242344418835	Medium (degree=0.502)/High (degree=0.495)/

(a) Without negligible bottleneck search



Fuzzy–based Search

Code Region	L2_TCM/L2_TCA	Bottleneck
MPI_SEND	0.6550124668722567	Negligible (degree=0.1)/
PARTICLE_LOAD	0.9497242344418835	Medium (degree=0.502)/High (degree=0.495)/

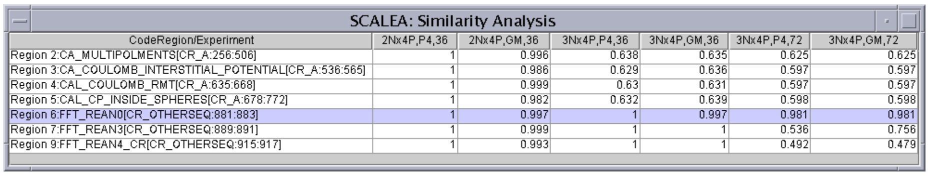
(b) With negligible bottleneck search

Fig. 5. Example of fuzzy-based bottleneck search.

Very simply, to show advantage of fuzzy-based bottleneck search, we experience with 3DPIC code to locate code regions that may have L2 cache access problems. Suppose a code region whose L2 cache miss ratio exceeds 0.7 is a bottleneck. In the first case we use a set of performance characteristic terms $T = \{low, medium, high\}$ representing the severity of the bottleneck. Three different fuzzy sets Z-function with range $[0.7, 0.8]$, Pi-function with range $[0.75, 0.95]$ and S-function with range $[0.9, 1]$ are associated with *low, medium*, and *high* term, respectively. We apply this search with 3DPIC code executed with 4 processes and we find that there is only one bottleneck as shown in Figure 5(a). The bottleneck falls into both classes *medium* and *high*, as shown in Figure 5(a). Since we are not certain about the threshold we decided to use another triangle FS with parameter $(0.65, 0.7, 0.75)$ to describe close area of the pre-defined bottleneck threshold. The result is that we find another code region as presented in Figure 5(b).

6.4 Similarity Analysis

We have implemented similarity analysis for all region summaries of a given code region in one experiment, and for region summaries of a set of selected code regions in a single or multiple experiment(s).


SCALEA: Similarity Analysis

CodeRegion/Experiment	2Nx4P,P4,36	2Nx4P,GM,36	3Nx4P,P4,36	3Nx4P,GM,36	3Nx4P,P4,72	3Nx4P,GM,72
Region 2:CA_MULTIPOLMENTS[CR_A:256:506]	1	0.996	0.638	0.635	0.625	0.625
Region 3:CA_COULOMB_INTERSTITIAL_POTENTIAL[CR_A:536:565]	1	0.986	0.629	0.636	0.597	0.597
Region 4:CAL_COULOMB_RMT[CR_A:635:668]	1	0.999	0.63	0.631	0.597	0.597
Region 5:CAL_CP_INSIDE_SPHERES[CR_A:678:772]	1	0.982	0.632	0.639	0.598	0.598
Region 6:FFT_REAN0[CR_OTHERSEQ:881:883]	1	0.997	1	0.997	0.981	0.981
Region 7:FFT_REAN3[CR_OTHERSEQ:889:891]	1	0.999	1	1	0.536	0.756
Region 9:FFT_REAN4_CR[CR_OTHERSEQ:915:917]	1	0.993	1	1	0.492	0.479

Fig. 6. Similarity analysis for LAWP0. We used `(wtime, 1.0)` to compute similarity measure. Experiment `2Nx4P,P4,36` is selected as the base. `1Nx4P` means 1 SMP node with 4 processors. `P4` and `GM` correspond to MPICH CH_P4 and Myrinet, respectively. The problem size is either 36 or 72 atoms. Distance measure is based on Euclidean function.

Figure 6 presents an example of using similarity analysis to examine selected code regions in 6 experiments. The first observation is that the performance of code region `FFT_REAN0` in the last 5 experiments is almost complete similar to the first experiment. The performance of `FFT_REAN3, FFT_REAN4` is almost similar in the first 4 experiments. This suggests that the performance of these code regions is not affected by changes of number of processors, communication libraries, even problem sizes (in case of `FFT_REAN0`). All code regions have similar performance in the first two experiments, suggesting the use of Myrinet does not increase much performance. This is confirmed by many cases in which communication libraries are different but the performance is very similar.

Factor	Fuzzy Set	Range	Factor Category
atoms	linear	[0,72]	problem size
CPU	S-function	[0,64]	machine
network	S-function	[0,158.20]	communication

Table 1. Parameters for controllable factors.

Table 1 shows an example of parameters of controllable factors. Table 2 presents the result of an example in which similarity is measured for code region `CA_MULTIPOLMENTS` in 6 experiments of LAPW0 by using parameters in Table 1. Performance score of the code region is based on S-function and distance measure is based on Euclidean function. In some cases, communication factor has very little impact on the performance, e.g., the network between the first and the second experiment is quite dissimilar while other factors are very similar, but the performance is very similar. A similar result obtained if we examine the fifth and sixth experiments. The CPU factor has significant impact on some cases. E.g., factors of the third experiment are the same as those of the first experiment, except that CPU factors are slightly different. However, the performance of the code region is quite different.

Experiments	**2Nx4P, P4,36**	**2Nx4P, GM,36**	**3Nx4P, P4,36**	**3Nx4P, GM,36**	**3Nx4P, P4,72**	**3Nx4P, GM,72**
$sim_{f_{atoms}}$ ({atoms,1})	1	1	1	1	0.5	0.5
$sim_{f_{CPU}}$ ({(CPU,1)})	1	1	0.9531	0.9531	0.9531	0.9531
$sim_{f_{network}}$ ({(network,1)})	1	0.1519	1	0.1519	1	0.1519
sim_o ({(wtime,1)})	1	0.996	0.638	0.635	0.625	0.625

Table 2. Example of similarity analysis with experiment factors for `CA_MULTIPOLMENTS` region in 6 experiments. The first experiment is selected as the base.

7 Related work

FL has been used in performance monitoring of parallel and distributed programs, e.g. performance contracts [9], but has not been exploited in data analysis techniques, e.g. performance classification, of existing performance tools.

APART introduces the concept of performance property [10] that characterizes a specific negative performance behavior of code regions. However, performance property is associated with a single performance metric. A performance property cannot represent a set of performance metrics. There is no concept of weight operator associated with performance properties. Also, our performance score is based on FL that allows the representation of fuzzy concepts such as *near* and *very*. Performance score can be computed based on linear and non-linear model with various membership functions.

Toward high-level scalable and intelligent analysis, classification based on machine learning has been used for classifying performance characteristics of communication in parallel programs [11]. Ahl and Vetter used multivariate statistical techniques on hardware performance metrics to characterize the system [12]. However, they do not deal with cases of multiple variables with different scales and weight factors. In [13], statistical analysis is used to study different (controllable and uncontrollable) factors that affect the mapping process of scientific computing algorithms to advanced architectures.

In [14] dispersion statistics is used to characterize the load imbalance by measuring the dissimilarity of performance metrics; metrics are normalized by measuring deviation from a mean value of a data set. Our similarity measure is based on fuzzy-based performance scores and is applied to not only code regions but also experiment factors.

In [6], historical data is used to improved automatic tuning systems. Performance score, similarity measure and fuzzy rules are fitted well for describing parameters and for improving decision making in performance tuning.

8 Conclusion and Future Work

This paper proposes a new approach to performance analysis that is based on soft computing. On the one hand, soft performance analysis techniques provide flexible, scalable and intelligent techniques for analyzing and comparing the performance of complex parallel and distributed applications. On the other hand, they interact with the user through high level notions. We complement existing work and contribute flexible and convenient methods to deal with uncertainty in the performance analysis, e.g. fuzzy-based bottleneck search, and to conduct the analysis in the form of high level notions, e.g. fuzzy-based search query. Still the soft performance analysis approach is just at an early stage, we believe it is a promising solution to provide soft, scalable and intelligent methods for automatic performance analysis.

Our future work is to study the application of soft performance analysis for dynamic performance tuning. Our proposed techniques could be applied to the performance analysis of large-scale complex dynamic Grid environments on which resources and their usage are unpredictable, performance data collected tends to be more imprecision and uncertainty. Moreover, performance similarity can be used to analyze and compare diverse Grid resources. Linguistic variables and fuzzy rules can be used in specifying and controlling service level agreements (SLAs) in the Grid.

References

1. Zadeh, L.A.: Fuzzy logic, neural networks, and soft computing. Commun. ACM **37** (1994) 77–84
2. Zadeh, L.A.: Fuzzy Logic = Computing with Words. IEEE Transactions on Fuzzy Systems **4** (1996) 103–111
3. Mitchell, T.M.: Machine Learning. McGraw Hill, New York, US (1997)
4. Truong, H.L.: Novel Techniques and Methods for Performance Measurement, Analysis and Monitoring of Cluster and Grid Applications. PhD thesis, TU WIEN, Austria (2005) http://dps.uibk.ac.at/̃truong/publications/linh-diss.pdf.
5. FuzzyJ Toolkit: http://ai.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit.html (2004)
6. Chung, I.H., Hollingsworth, J.K.: Using Information from Prior Runs to Improve Automated Tuning Systems. In: ACM/IEEE SC2004, Pittsburgh, PA (2004)
7. Cain, H.W., Miller, B.P., Wylie, B.J.: A Callgraph-Based Search Strategy for Automated Performance Diagnosis. In: Euro-Par 2000 Parallel Processing. (2000) 108–122
8. Fahringer, T., Seragiotto, C.: Aksum: A performance analysis tool for parallel and distributed applications. Performance Analysis and Grid Computing (2003)
9. Vraalsen, F., Aydt, R.A., Mendes, C.L., Reed, D.A.: Performance contracts: Predicting and monitoring grid application behavior. In: Proceedings of GRID 2001. Volume LNCS 2242., Denver, Colorado, Springer-Verlag (2001) 154–165
10. Fahringer, T., Gerndt, M., Mohr, B., Wolf, F., Riley, G., Träff, J.: Knowledge Specification for Automatic Performance Analysis. Technical report, APART Working group (2001)
11. Vetter, J.: Performance analysis of distributed applications using automatic classification of communication inefficiencies. In: Conference Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, New Mexico, ACM SIGARCH (2000) 245–254
12. Ahn, D.H., Vetter, J.S.: Scalable Analysis Techniques for Microprocessor Performance Counter Metrics. In: IEEE/ACM SC'2002, Baltimore, Maryland (2002)
13. Santiago, N.G., Rover, D.T., Rodriguez, D.: A Statistical Approach for the Analysis of the Relation Between Low-Level Performance Information, the Code, and the Environment. In: Proceedings of 2002 International Conference on Parallel Processing Workshops (ICPPW'02), Vancouver, B.C., Canada, IEEE Computer Society Press (2002) 282–
14. Calzarossa, M., Massari, L., Tessera, D.: A methodology towards automatic performance analysis of parallel applications. Parallel Comput. **30** (2004) 211–223