Knowledge-Based Systems 23 (2010) 343–349

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

From source code to runtime behaviour: Software metrics help to select the computer architecture

Frank Eichinger [a,*], David Kramer [b], Klemens Böhm [a], Wolfgang Karl [b]

[a] Institute for Programme Structures and Data Organisation (IPD), Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany
[b] Institute for Computer Science and Engineering (ITEC), Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany

ARTICLE INFO

Article history:
Available online 20 November 2009

Keywords:
Computer architecture
Data mining
Performance prediction
Source-code metrics
Control-flow graphs

ABSTRACT

The decision which hardware platform to use for a certain application is an important problem in computer architecture. This paper reports on a study where a data-mining approach is used for this decision. It relies purely on source-code characteristics, to avoid potentially expensive programme executions. One challenge in this context is that one cannot infer how often functions that are part of the application are typically executed. The main insight of this study is twofold: (a) Source-code characteristics are sufficient nevertheless. (b) Linking individual functions with the runtime behaviour of the programme as a whole yields good predictions. In other words, while individual data objects from the training set may be quite inaccurate, the resulting model is not.

1. Introduction

The question which computer architecture is best suited for a certain application is of outstanding importance in the computer industry. With the continuous refining of computer architectures, this problem becomes even more challenging. Think of the high degree and various forms of parallelism (multicores), heterogeneity due to application-specific accelerators, interconnection technology on the chip, or the memory hierarchy. The design space is huge and leads to a broad variety of processor architectures. It is not at all obvious which architecture is best suited for a specific application. For example, due to the branch-prediction unit,an application with predictable branches benefits from a long pipeline, while a shorter pipeline is better for unpredictable branch behaviour. The question which architecture yields the best performance is particularly important for high-performance computing where an expensive system is purchased for a few or even only one application.

Traditional approaches use experimental executions, simulations or analytical models to identify the best computer architecture for a given application. For instance, when a computing centre plans to procure a new cluster for a specific application, one way to do so is to compare the runtime behaviour of this application on different platforms. This obviously is time-consuming and expensive, and the platforms in question must be available in the first place. Similar arguments apply to state-of-the-art simulation approaches: In-depth simulation is time-consuming, in particular with machine models that are sophisticated. Finally, due to the increasing complexity of computer systems, establishing analytical models of the computer architectures in question is extremely hard. This may lead to a relatively poor reliability of these models, compared to experimental executions and simulations. In consequence, techniques are sought which help to decide between several platforms for a specific application. Ideally, such techniques should not require any execution or simulation and should be based on an analysis of the application in question. Some approaches exist which can make a decision between several platforms [1,2]. They rely on the assumption that similar programme perform alike when executed on the same machine. However, in addition to measures deduced from the source code, these approaches make use of runtime-related characteristics, such as branch probabilities or instruction counts. To generate these characteristics, simulations or programme runs on real hardware are necessary.

This article reports on the results of a study that investigates another method to determine the best computer architecture for a given application. The method likewise assumes that similar applications have similar execution behaviour. But in contrast to the previous work, we have consciously decided not to take any runtime-related information of the application in question into account. In this current study we characterise the application entirely by means of measures gained from the source code. In other words, we hypothesise that there is a strong correlation between programme properties encoded in the source code and the execution

* Corresponding author. Tel.: +49 721 608 7336; fax: +49 721 608 7343.
E-mail addresses: eichinger@kit.edu (F. Eichinger), kramer@kit.edu (D. Kramer), klemens.boehm@kit.edu (K. Böhm), karl@kit.edu (W. Karl).

0950-7051/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2009.11.014

behaviour, and that this correlation can be exploited. This hypothesis may appear to be unsettling – taking only source-code characteristics into account obviously is much less informative than runtime behaviour! In particular, it is difficult to impossible to infer how often a certain function is typically executed. Another issue is that source-code metrics, i.e., existing measures that quantify characteristics of the source code, typically are defined on the function level rather than on the level of entire programme, while we are interested in predictions for programme as a whole. Having said this, the method examined here is a data-mining approach with the following distinctive feature: It links individual functions with the runtime behaviour of the programme as a whole. Even though this approach clearly is simplistic, i.e., the characterisation of individual functions may be *very* inaccurate, it yields a prediction accuracy for entire programme which is surprisingly high. In retrospect, our explanation is as follows: since applications typically consist of a large number of functions, there is a lot of training data which, on average, compensates for that simplification. That is, we provide evidence that source-code characteristics alone are indeed helpful to predict a good computer architecture. More specifically, our contributions are as follows:

1.1. Software metrics

The software-engineering community has proposed a number of software metrics in order to represent source-code characteristics and properties. Originally, these metrics have been cast as quality measures rather than as performance indicators. Preliminary investigations of ours have revealed that measures based on the control flow of functions are particularly promising to predict runtime behaviour. Consequently, we define and derive a number of metrics, such as graph invariants, based on the control-flow graphs (CFGs) [3] of the functions. We use these metrics in addition to more common ones.

1.2. Classification framework

We propose a classification setting for our specific context and evaluate it. This setting is not obvious: While most metrics are available at the function level, we want to choose the best architecture for a programme as a whole. Instead of potentially lossy aggregation approaches, we propose a framework where we first learn at the function level before deploying classifier-fusion techniques to come up with predictions at the programme level.

1.3. Evaluation

Our case study features an evaluation using five systems from the online database of the SPEC CPU 2000 and 2006 benchmark suites. The results are that, for 'relatively similar' computer architectures to choose from, and with the runtime behaviour of only few programme used as training data, our approach achieves an average prediction accuracy of 78% when choosing between two systems.

1.4. Correlation of software metrics and runtime behaviour

Our main concern, from a 'research' perspective, has been to confirm (and to exploit) the relationship between source-code properties and runtime behaviour, on different platforms. Besides the fact that the approach investigated here does indeed yield a statement regarding the computer architecture best suited, our evaluation shows that the correlation between source-code properties and runtime behaviour is remarkably strong.

Paper outline: Section 2 presents related work, Section 3 describes the process of acquiring software metrics, before we describe the data-mining process in Section 4. Section 5 presents our results, which are discussed in Section 6. Section 7 concludes.

2. Related work

In the past, various approaches to predict the runtime or the runtime behaviour of given applications have been proposed. Newer approaches propose the use of machine-learning approaches for this prediction.

In [4,5] the authors use multilayer neural networks to predict the performance of the multigrid solver SMG 2000 on a BlueGene/L cluster. The parameter space includes the cluster configuration as well as the size of the grid used. The training set used consists of performance results on an actual platform using a collection of random points from the parameter space. In contrast to our approach, these approaches can only be used to predict the performance of a parametrised application on a cluster with different configurations. In addition, they require time-consuming training-data generation.

Another possible use of neural networks is described in [6]. Here, İpek et al. use neural networks to predict the performance of points in the design space. Neural networks are used to approximate the design space and to create a model of it. The model built predicts the performance of points with high accuracy and has been applied to memory hierarchy and chip-multiprocessor design spaces.

To ease the generation of analytical models of complex high-performance systems, Kühnemann et al. have developed a compiler tool for automated runtime prediction of parallel MPI programme [7]. The tool analyses the source code of MPI programme to create an appropriate runtime-function model for the communication overhead and for the computation. Properties of the underlying machine are needed for proper prediction of the computation effort.

Another method for performance prediction is [2] from Joshi et al. They use inherent programme characteristics to measure the similarity between programme. Instead of using microarchitecture-dependent measures for characterisation, such as cycles per instruction, cache-miss rate or runtimes, they use microarchitecture-independent ones. These measures include the instruction mix, the size of the working set and branch probabilities. To generate the measures, either simulation or execution of the application is necessary. Based on [2], the authors exploit the similarity between programme for performance prediction of applications in the SPEC CPU 2000 benchmark suite [1]. They use microarchitecture-independent characteristics and performance numbers from an application to build a so-called benchmark space. To predict the performance of an application, the developer has to compute a point in the benchmark space using the same characteristics. Comparing our approach to [1] reveals that both approaches can predict the runtime behaviour of an application in question on given platforms and have advantages and disadvantages. [1] uses runtime-related microarchitecture-independent characteristics in the prediction process. The advantage is that predictions are likely to be more precise. A drawback is that the execution of the application on an existing platform or a detailed simulation is necessary. Saveedra and Smith [8] use a similar approach as proposed in [1], but they use programme and machine characteristics to estimate the performance of a given Fortran programme on an arbitrary machine. A drawback of all these approaches is the usage of architecture-dependent characteristics which are time-consuming to create. Our approach in turn does not require such characteristics.

Finally, [9] studies the same problem as this current paper, but with a different approach based on graph mining and control-flow graphs. The technique described here yields better results.