Performance Prediction for Java Applications

Andy Georges*, Dries Buytaert*, Kris Venstermans*, Lieven Eeckhout*, Koen De Bosschere*

* *Dept. ELIS, Ghent University, St.-Pietersnieuwstraat 41, B-9000 Ghent, Belgium*

ABSTRACT

In this paper, we present a performance model for Java applications that allows us to determine the relative performance of the same application on two different platforms. During the execution of Java applications, there is a complex interation between the application, the virtual machine, the operating system and the hardware. As such, an execution performance model should take all of the above into account in the model. A reliable performance model that allows estimating the relative performance has important applications on modern hardware platforms. Modern heterogeneous multi-core processors can use such an estimation to assign a given task to the core that can yield the best performance while consuming as few resources as possible.

KEYWORDS: Performance; Java

1 Introduction

As Java technology is becoming increasingly popular for well-known reasons, such as cross-platform compatibility, a robust security model, and its object oriented programming paradigm, it becomes omnipresent on a wide range of systems, from handheld devices to large servers. Moreover, with the advent of heterogeneous multi-core processors, the importance of Java can grow even further. Within this context, an accurate performance prediction model for Java applications presents several important applications. First of all, it allows a user to compare several hardware solutions for a given Java application. Second, having a good performance estimate can allow computation offloading [Chen04], i.e. move the computation from one device to another device. In similar vein, it allows a scheduler to run a computation on the most suitable core in a heteregeneous multi-core system [Kuma04].

The goal of this paper is to explore a first approximation to an accurate performance estimation model. Such a model will guess e.g. the IPC of an unseen application, by relying only on platform independent – high-level – metrics. This is quite challenging because there

[1]E-mail: {ageorges, dbuytaer, kvenster, leeckhou, kdb}@elis.ugent.be

are multiple execution layers that need to be taken into account, i.e. the application, the virtual machine, the operating system and the hardware. In this paper we focus on relative performance estimates, i.e. on predicting which platform will yield better performance for a given application.

2 Methodology

Using the model to estimate the performance of a benchmark requires the following steps: (i) the benchmark is profiled and split into phases, (ii) for each phase the high-level metrics are measured, (iii) the profile of each phase is matched to the profiles in the model, and (iv) the final performance estimate for the benchmark is a weighted sum of the estimates for each of its phases.

Building the model requires steps (i), (ii) and a statistical analysis, which is explained further on.

2.1 Acquiring execution profiles

First, we run each benchmark in the training set through a phase detection mechanism. This yields a set of method-level phases [Geor04]. Each of these phases will be measured separately during a single execution of the application. A phase is defined as a sub-graph in the call-graph of the application. As such, phases are fragments of the execution that show similar behaviour, but that are not necessarily temporarily adjacent. Such granularity is appropriate for identifying major program phases [Bala00, Geor04]. A detailed description of how the phases are detected is given in [Geor04].

The second step entails measuring both the high-level (platform independent) and the low-level (non-functional) metrics (e.g. IPC) we are interested in. Essentially, the model we are building yields a link between high-level platform independent metrics obtained by analysing the Java bytecode from the application, and low-level hardware metrics, such as cycles-per-instruction (CPI). We do this for two platforms, as a proof-of-concept. The key high-level metrics we used are given in Table 2.1.

We capture these metrics during a single run of the benchmarks. Together with the low-level metrics measured per benchmark on each target platform, we obtain a set of performance profiles, one profile per benchmark-input pair. These profiles make up the input for building the model. We distinguish the training and validation profiles, i.e. the phases from benchmarks present in the latter, do not appear in the former.

2.2 Statistical analysis

We use two statistical techniques, namely Principal Components Analysis (PCA) and Cluster Analysis (CA) [John02] to get a better grip on the amount of data gathered during the profiling step, and to allow us to match the phases from the training set.

After capturing the execution profiles containing the high-level metric, we perform a PCA on them. The input for PCA is the set of high-level measurements aggregated per phase. The output is a set of principal components per phase, placing the phases in the PCA space. This technique has two important benefits. First, the resulting PCA space has uncorrelated components, unlike the original metrics, which may show more or less correlation in

Control flow	Data structures
Basic block count	Scalar density
Conditional branches	Array density
Backward jumps	Object size
Branch target frequency	Object loads
Basic block reuse distance	Object stores
	Object reuse distance
Method invocations	Bytecode mix
Methods	Bytecodes
Synchronized methods	Load operations
Final methods	Store operations
Static methods	Arithmetic operations
Virtual methods	Constant operations
Interface methods	Exceptions
Receiver polymorphism	Conversions
	Lock operations
	Remaining operations

Table 1: High-level metrics used in the model

between them. Second, we can make do with less principal components than the number of original metrics, because the first components usually explain most of the variance present in the data. After the PCA has finished, we make sure that the variance along each principal component equals one.

The second technique we use is Cluster Analysis, more specifically a k-means clustering algorithm. K-means is an algorithm that will cluster a given set of data points into disjoint clusters such that the sum-of-squares with respect to the cluster center is minimal. After CA, we get a set of clusters in which each cluster contains phases that closely resemble each other.

2.3 Prediction

The first two items have been briefly discussed above. Matching the profiles of the validation benchmark to the profiles of the training set is done using the statistical techniques discussed above. First, we inject the phase profiles into the PCA space of our model. Second, we check to which cluster each phase belongs in the PCA space. The low-level values assinged to the phases are then those of the cluster to which the phase was assigned, i.e. the mean value of the phases in the cluster.

3 Preliminary Results

For the validation of our model, we have used Jikes RVM [Alpe00]. This virtual machine was instrumented to allow counting the high-level platform independent metrics. Additionally, we have previously extended Jikes RVM to allow measuring low-level performance metrics, by accessing the hardware counters of the processor [Geor04]. The benchmarks were taken

from three major suites: SPECjvm98, SPECjbb2000, and the Dacapo[2] benchmark suite. From the latter we have used all applications except batik, bloat, chart and pmd, due to restraints in Jikes RVM. The SPECjbb2000 benchmark was actually changed to perform a set amount of transactions, instead of running for a set amount of time. In total, we used 14 benchmarks.

The non-functional metric on which we focus is the so-called BPC, or bytecodes-per-cycle. This is computed by counting the number of executed bytecodes (per phase) at the high level and the number of cycles spent in each phase at the low level.

We have run our experiments on two platforms. The first is an Athlon XP clocked at 1GHz with 1GiB of RAM. The second platform is an IBM Power4, clocked at 1Ghz and with 1GiB RAM. We have used our model with a variable number of clusters in the analysis. It turns out that when retaining 25 principal components, we correctly predict in 10 out of 14 benchmarks which platform will show a higher BPC rate. The number of clusters ranged between 100 and 130 yields the best result with a slight dropoff for a higher and a smaller number of clusters. Then the correct predicted percentage of benchmarks drops toward 50%, which is about as good as taking an educated guess.

4 Conclusion and future work

We have made a first step towards performance prediction for Java applications. We focussed mainly on relative performance, where we predict which platform yields a better BPC rate. For this we used a rigorous statistical approach. We attain over 70% accuracy when predicting the platform which shows a higher BPC rate.

Of course, the model is susceptible to improvements. 70% correct predictions is a bit low to be really useful, so we will try to enhance the model. One possibility is by seeking other high-level characteristics that can have an important influence on performance. Another option is to improve the statistical model.

5 Acknowledgement

This research was funded by Ghent University, by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), by the Fund for Scientific Research-Flanders (FWO-Flanders) and by the HiPEAC network.

References

[Alpe00] B. ALPERN, C. ATTANASIO, J. BARTON, M. BURKE, P. CHENG, J. CHOI, A. COCCHI, S. FINK, D. GROVE, M. HIND, S. HUMMEL, D. LIEBER, V. LITVINOV, M. MERGEN, T. NGO, J. RUSSELL, V. SARKAR, M. SERRANO, J. SHEPHERD, S. SMITH, V. SREEDHAR, H. SRINIVASAN, AND J. WHALEY. The Jalapeño Virtual Machine. *IBM Systems Journal*, 39(1):211–238, 2000.

[Bala00] R. BALASUBRAMONIAN, D. ALBONESI, A. BUYUKTOSUNOGLU, AND S. DWARKADAS. Memory Hierarchy Reconfigurtion for Energy and Performance

[2]We used the 20050224 beta version

in General-Purpose Processor Architectures. In *Proceedings of the 33th Annual International Symposium on Microarchitecture (MICRO-33)*, December 2000.

[Chen04] G. CHEN, B. KANG, M. KANDEMIR, N. VIJAYKRISHNAN, M. IRWIN, AND R. CHANDRAMOULI. Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):795–809, 2004.

[Geor04] A. GEORGES, D. BUYTAERT, L. EECKHOUT, AND K. DE BOSSCHERE. Method-Level Phase Behavior in Java Workloads. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Languages, Applications and Systems (OOPSLA'04)*, pages 270–287, October 2004.

[John02] R. JOHNSON AND D. WICHERN. *Applied Multivariate Statistical Analysis*. Prentice Hall, fifth edition, 2002.

[Kuma04] R. KUMAR, D. TULLSEN, P. RANGANATHAN, N. JOUPPI, AND K. FARKAS. Single-ISA Heterogeneous Mult-Core Architectures for Multithreaded Workload Performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA-31)*, pages 64–75, June 2004.