

Towards a Strategy for Performance Prediction on Heterogeneous Architectures

Silvio Stanzani, Raphael C  be, Jefferson Fialho, Rog  rio Iope, Marco Gomes, Artur Baruchi and J  lio Amaral

¹ N  cleo de Computa  o Cient  fica (NCC) - Universidade Estadual Paulista
[silvio, rmco  be, jfialho, rogerio, mgomes, abaruchi,
julioamaral]@ncc.unesp.br

Abstract. Performance prediction of applications has always been a great challenge, even for homogeneous architectures. However, today's trend is the design of cluster running in a heterogeneous architecture, which increases the complexity of new strategies to predict the behavior and time spent by an application to run. In this paper we present a strategy that predicts the performance of an application on different architectures and classify then according to the performance that the application can achieve on each architecture. The proposed strategy was able to classify correctly three of four applications tested without overhead implications. Our next step is to extend the metrics used to perform the classification in order to increase the strategy accuracy.

Keywords: Performance Prediction, Heterogeneous Systems, Parallel Processing.

1 Introduction

One trend that has become popular in designing computer clusters is the use of heterogeneous architectures that can composed by different models of multicore and manycore architectures [1], which leads to computer infrastructures that are becoming more complex in scale, as well as in the variety of cores and memory systems. Such a trend can be observed in the configuration of several supercomputers, such as, Stampede2¹, JURECA² and Santos Dumont³.

Characterize the performance of applications on different architectures is essential to efficiently use heterogeneous infrastructures. Such characterization can be carried out by the users by profiling the application on each architecture and analyzing the results. Some initiatives have been working on the development of performance prediction techniques, to automatically generate such characterization and support runtime decisions [2,3].

¹ <https://www.tacc.utexas.edu/systems/stampede2>

² http://www.fz-juelich.de/ias/jsc/EN/Home/home_node.html

³ <http://sdumont.lncc.br/>

One of the challenges related to performance prediction on heterogeneous infrastructures is identify the architecture that presents the best performance for a given application. In order to overcome this challenge, the contribution of this work is a strategy based on profiling data to do performance prediction of parallel applications on single machines, without overhead implications, capable of classify available architectures according to the performance level that can be obtained executing the application.

The remainder of this paper is organized as following: Section 2 presents the problem definition and concepts, Section 3 presents the related work, Section 4 presents the strategy proposed, Section 5 presents the evaluation of strategy and Section 6 presents the conclusions and future work.

2 Concepts and Problem Definition

In this section, we are going to present the problem definition and the main concepts related with the strategy proposed.

2.1 Concepts

One technique capable of characterize the performance of an application, abstracting the characteristics of applications and architectures, is called machine balance which defines the performance of an application using the following metric: the ratio of the amount of data transfer to the number of floating-point operations for a particular processor [4].

The information to characterize the performance of an application according to this model can be obtained from code profilers, which perform several measurements during the execution of a small part of application [8] and can be based on two approaches:

- Source code profiling which is the measurement of source code or assembly code behavior
- Hardware profiling which is the collection of the data provided by PMU (Performance Monitoring Units) in order to identify the hardware usage efficiency during the execution of an application.

2.2 Problem Definition

The problem we are tackling is the performance prediction of an application across different computational architectures to support runtime decisions. In this sense, we impose the following restrictions:

- The application is provided without the source code.
- The prediction has to be executed within a time limit. This is important to avoid significant overhead in a production environment.
- The user can provide different versions of the same applications optimized to different resources. Normally, compilers automatically tune the application to different

architectures, also developers can use instructions that is available on a specific architecture for example.

3 Related work

In [5], the authors present a method for profiling C code with OpenACC directives, from that, they create a computation model that will run on an HPC architecture simulator in order to measure some performance metrics. In our case, we use data collected from the real execution. This is very important since we are dealing with heterogeneous architectures that have different microarchitecture implementations.

In [7], the authors propose an approach that uses small input cases for performing the full execution of the program, in order to collect data of all the code regions. Their concern is the execution time of the program and to predict it, they use a regression model, based on small kernels – a similar approach is adopted by [6] - to estimate the execution estimate. Using such statistical models require training, to learn the correct parameters, that could be a hard task depending on the model used to perform the regression. In our proposal, we plan to identify the architectures that provides lower execution time, and not to predict the execution time, also, we want to perform such analysis within a time limit.

In [3], the authors propose a tool for making performance prediction named COMPASS. They also generate an execution model – a translation of the source code to the Aspen Language, that should be analyzed by a performance evaluation tool, named Aspen Performance Prediction Tools that should also be fed with an architecture descriptor file, containing the information regarding the architecture on which the performance prediction should be done. Also, the code can be instrumented to help the tool to better generate the program execution model. In our case, no manual instrumentation is needed. The metrics are obtained using standard tools and also no source code translation is required.

4 A Strategy for Performance Prediction

In this session, we will present a strategy to classify a set of different architectures according to the performance that an application can achieve on each architecture.

The strategy receives the following inputs:

- Application: can be one version to be executed in all resources or can be composed of compiled version of the same source code optimized to each computational architecture.
- Set of resources: the architectures we are considering in this work are the following Intel generations:
 - Intel Xeon: SandyBridge, Haswell and Skylake;
 - Intel Xeon Phi: Knights Landing.

We will describe the profiling collection performed by strategy on Section 4.1 and the data analysis and output on Section 4.2.

4.1 Profiling Collection

The strategy characterizes the performance of an application on different architectures according to the machine balance model [4], measuring two characteristics of the execution of an application on each architecture: the amount of operations that an application is capable of execute and the amount of work that is wasted with overhead.

We use Intel VTune⁴ to perform hardware profiling and Intel Advisor⁵ to perform source code profiling. We fixed period of time to execute each profiling tools equals to 30 seconds to each tool. A script manages the execution of profiling tools to collect the following metrics to define the measurements:

- Intel Advisor:
 - Ratio of giga floating-point operations per second (GFLOPS);
 - Arithmetic Intensity (AI) which is the ratio of giga floating-point operations by data transfer;
- Intel Vtune:
 - Clockticks per Instructions Retired (CPI). That is calculated by dividing the number of unhalted processor cycles (Clockticks) by the number of instructions retired, that is the instructions that effectively finishes its execution.

The GFLOPS measures the amount of operations that an application is capable of executing considering the useful work and any other overhead that can be present in the application, such as, cache miss, data transfer latency from main memory to processor, vectorization overhead, and so on.

The strategy identifies the overhead present in the application by the means of AI and CPI:

- The CPI is a measure that helps estimate the overhead present in the application, because one the consequences of high overhead is a decrease on retired instructions.
- The AI measures the throughput of instructions execution. In this sense, higher level of AI indicates that the execution of application demands less data transfers from memory to CPU, which means lower latency in executing instructions.

4.2 Data Analysis and Output

The strategy returns the performance prediction by the means of a numbered list that classify architectures from lowest to highest performance level, based on a metric that we call estimated processing capacity (EPC) which is defined in the formula on equation (1).

⁴ <https://software.intel.com/en-us/intel-vtune-amplifier-xe>

⁵ <https://software.intel.com/en-us/advisor>

$$EPC(Application, Resource) = \frac{GFLOPS * AI}{CPI} \quad (1)$$

The GFLOPS represents the amount of operations that an application is capable of perform on an architecture and AI represents the throughput of instructions execution, so to define the comparison metric we multiply these two variables in order to obtain a quality estimate about how the application is using the architecture. We divide by CPI, because it represents the applications overhead on an architecture, CPI level is proportional to overhead level. In this sense, lower CPI level will have lower impact on EPC.

The strategy calculates the EPC for an application on each resource. The list of value is normalized by one to the amount of resources defined in the resource set. This normalization is important because the value returned from EPC can not be used to characterize a proportional performance level across different architectures.

5 Evaluation

In this section, we present an evaluation of the strategy in Session 5.1 we present the workload and hardware infrastructure used to evaluate the strategy and in Session 5.2 we present the results obtained.

5.1 Workload and Hardware Description

We created a workload composed by applications with different characteristics from a set of problem organized by Intel [9]:

- A matrix multiplication code developed using Intel Intrinsics;
- A numeric model in finance optimized using AVX-512 ER (Exponentials and Reciprocals);
- A N-Body simulation with most part of code vectorized;
- A Diffusion simulation with most part of code developed with scalar instructions.

The hardware infrastructure we are going to use to evaluate the strategy is described in the Table 1.

Table 1. Hardware Infrastructure used for Evaluation.

Architecture	Processor	Cores	Threads	Dram
SandyBridge	2x 2.6GHz	8	32	64GB
Haswell	2x 2.3GHz	36	72	128 GB
Skylake	2x 2.1GHz	48	96	192 GB
Knights Landing (cache mode)	1x 1.4GHz	68	272	192 GB
Knights Landing (flat mode)	1x 1.4GHz	68	272	192 GB

5.2 Results and Discussion

We executed the strategy with the workload on the infrastructure presented in Section 5.1 and show the results in Table 2.

Table 2. Comparing Strategy Output (Classification of Architecture) against Execution Time.

Numeric Model in Finance					
Architecture	Skylake	Haswell	SandyBridge	KNL (Flat-Mode)	KNL (Cache Mode)
Execution Time (Seconds)	458	1036	3443	235	224
Classification	3	2	1	4	5
Diffusion					
Architecture	Skylake	Haswell	SandyBridge	KNL (Flat-Mode)	KNL (Cache Mode)
Execution Time (Seconds)	511	309	220	425	1557
Classification	3	4	5	2	1
N-Body					
Architecture	Skylake	Haswell	SandyBridge	KNL (Flat-Mode)	KNL (Cache Mode)
Execution Time (Seconds)	306	467	1253	343	347
Classification	5	4	3	1	2
Matrix Multiplication (Intrisincs)					
Architecture	Skylake	Haswell	SandyBridge	KNL (Flat-Mode)	KNL (Cache Mode)
Execution Time (Seconds)	172	159	344	132	227
Classification	4	5	1	3	2

The results show that three architectures were correctly classified and one architecture were classified incorrectly.

All the metrics collected by the profilers (GFLOPS, CPI and AI) corresponds to an average of all loops present in the application. In the three applications classified correctly one loop dominates the execution of all application.

In the case of N-body the strategy classified KNL as presenting the worse performance to execute N-Body, but the execution time shows that KNL presents the best performance. This happens because two or more loops dominates the execution time, and currently the strategy does not associate weights to each loop according to its impact on execution time. In this sense, some loops in the critical path that presents great impact on execution time and higher performance on KNL was not expressed in the average result.

6 Conclusions and Future Work

This work presented a strategy to classify architectures according to the performance achieved executing a given application. It can be helpful when buying new compute nodes for a running cluster, decide in which nodes of a cluster an application is preferable to run (schedule decisions for example) and so on. Three of four applications used in our experiments were correctly predicted using our strategy. The low overhead of the strategy is another feature that differs from existing predictions strategies.

As future work, we intend to extend the metrics used to perform the classification to increase accuracy, such as weight of loops in execution time and others events (i.e. cache hit rate).

6.1 Acknowledgements

The authors would like to thank the Center for Scientific Computing at the São Paulo State University (NCC/UNESP) for the use of the manycore computing resources, partially funded by Intel in the context of the following projects: "Intel Parallel Computing Center", "Intel Modern Code Partner", and "Intel/Unesp Center of Excellence in Machine Learning".

References

1. Yang, X. J., et al.: The TianHe-1A supercomputer: its hardware and software. *Journal of computer science and technology*, 344-351 (2011).
2. Rosales, C., et al.: Performance Prediction of HPC Applications on Intel Processors. In: *Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE (2017).
3. Lee, S., Meredith, J. S., Vetter, J.S.: Compass: A framework for automated performance modeling and prediction. In: *Proceedings of the 29th ACM on International Conference on Supercomputing*, ACM (2015).
4. McCalpin, J. D.: Memory bandwidth and machine balance in current high performance computers. *IEEE computer society technical committee on computer architecture (TCCA) newsletter* 2, 19–25 (1995).
5. Obaida, M. A., et al.: Parallel Application Performance Prediction Using Analysis Based Models and HPC Simulations. In: *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM (2018).
6. Benoit, N., Louise, S.: A First Step to Performance Prediction for Heterogeneous Processing on Manycores. *Procedia Computer Science* 51, 2952-2956 (2015).
7. Escobar, R., Boppana, R.V.: Performance prediction of parallel applications based on small-scale executions. In: *High Performance Computing (HiPC)*, 2016 IEEE 23rd International Conference, IEEE (2016).
8. Browne S., Dongarra J., Garner N., London K., and Mucci P.. 2000. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (SC '00)*. IEEE Computer Society, Washington, DC, USA, , Article 42.
9. Reinders J. and Jeffers J. *High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches*, 2015.