Energy Optimisation via Genetic Improvement

A SBSE technique for a new era in Software Development

Bobby R. Bruce
University College London
London
United Kingdom
r.bruce@cs.ucl.ac.uk

ABSTRACT

The discipline of Software Engineering has arisen during a time in which developers rarely concerned themselves with the energy efficiency of their software. Due to the growth in both mobile devices and large server clusters this period is undoubtedly coming to an end and, as such, new tools for creating energy-efficient software are required. This paper takes the position that Genetic Improvement, a Search-Based Software Engineering technique, has the potential to aid developers in refactoring their software to a more energy-efficient state; allowing focus to remain on functional requirements while leaving concerns over energy consumption to an automated process.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

Keywords

Genetic Improvement, GI, SBSE, Search Based Software Engineering, energy efficiency, energy consumption, energy optimisation

1. INTRODUCTION

The well-known fairy tale of Goldilocks and the three bears is familiar to many and as such been used in science to popularise instances where circumstances are "just right", not too big or too small; not too hot or too cold. The Goldilocks zone is an area between the minimum and maximum diameter from a given sun where scientists believe life is possible; a Goldilocks economy is one which grows moderately, avoiding both recession and excessive growth rates indicative of a boom with a coming bust. This paper believes the majority of developers, almost entirely unwittingly, have been living through a Goldilocks period of software development where focus has been primarily on desktop machines; a period rapidly drawing to a close.

GECCO '15, July 11 - 15, 2015, Madrid, Spain

ISBN 978-1-4503-3488-4/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768420

The reason this paper refers to this time as a "Goldilocks period" is desktops, due to their size and limited mobility, did not require developers to consider the energy consumed by their applications. Instead they were able to focus solely on other, better understood, non-functional attributes such as execution time or memory consumption. Desktops are simply too small for their energy consumption to be noticed by users. At the same time they are are large enough for their mobility to be low and thus be powered by mains electricity so that power is not limited in the same way as with smaller battery-powered devices.

2. THE PROBLEM

Unfortunately we are moving away from this Goldilocks period of development. On one side we can observe the growth in the smartphone and tablet market where developers are restricted to a limited power supply between charges. The forthcoming "Internet of Things" is likely to constrain developers even further with small circuits containing batteries expected to last a considerable length of time before being replaced or recharged. On the other side we have servers. Though the energy is not constrained in the same way as with smaller computing devices, those in charge of managing these systems are influenced to a greater extent by economic pressures (and in some case ecological concern) over how much energy is being consumed. Servers were estimated to have consumed between 1.1 and 1.5% of global electricity production in 2010 [3] and thus it is easy to see why those in the server industry are eager to cut the amount they use.

The market for both very small and very large computing systems is likely to increase in future, with the familiar middle-ground left to stagnate. As developers are gradually forced out of their cosy Goldilocks zone where energy is a non-issue, and into one where it is a central concern, a question arises: how exactly do we minimise software's energy consumption?

The main issue for the developer who wishes to develop energy efficient software is the gulf of execution between the developer and the energy consumed by the software they deliver. Source-code rarely gives any indication of the amount of energy it shall consume when compiled and run. The Goldilocks period of software development did not require such a relationship to be obvious and therefore few tools exist to map it. Evidence suggests that subtle changes at the source-code level have an impact on the total energy consumed by a compiled product [2, 6, 9] though in ways that

are difficult, if not impossible, to determine outside a trial and error basis.

It seems unlikely there exists a simple set of rules which a developer may follow to make their software more energy efficient. Even if such rules could be formulated, would we trust developers to follow them and would it be cost effective in terms of education cost and additional development time? The most logical path is to develop an automated process capable of finding changes within code that improve energy efficiency while maintaining functional correctness. Though there have been some initial investigations into possible techniques [6, 8] the area remains largely unexplored. It is the position of this paper that Genetic Improvement, GI, a Search-Based Software Engineering (SBSE) technique is the most promising candidate for this role.

3. THE SOLUTION: GI

Genetic Improvement is a technique that treats software code as if it were genetic material which is then evolved to produce more optimal solutions. When implemented to preserve functional qualities it can be viewed as a system for the automatic refactoring of source-code to improve non-functional attributes. Unlike alternative approaches, GI specialises is taking existing human-written code and optimising it. When applied to the source-code level it can produce human readable changes which can then be verified both by developers and modern testing tools.

Like natural evolution, GI is capable of optimising for specific environmental conditions such as hardware configurations [4] and input data [7]. This property is advantageous in the current development environment which has seen significant growth in the number hardware configurations and operating systems available for development. While the model up to this point has been, at best, one of "code once, deploy everywhere", GI has the potential for the automation of bespoke solutions for specific cases; a new model of "code once, optimise everywhere" is feasible.

GI has already been shown capable of decreasing software execution time [4, 5, 7, 10] and, in most cases, doing so by making small changes to the source-code that would be difficult for a human developer to find. The question therefore arrises, if GI has been shown effective for execution time, can the same techniques be applied to reduce software energy consumption?

In 2015 an investigation into using GI to optimise energy efficiency was carried out [1]. The aim of the investigation was to find how much energy reduction could be achieved when specialising MiniSAT for a particular problem domain. The results showed that GI was capable of reducing energy consumption by as much as 25%. When taking into consideration MiniSAT's reputation for being difficult to optimise, it is an encouraging first-step into the area of energy reduction via Genetic Improvement.

Though these initial successes are promising more research is required. Optimising Energy consumption using GI has yet to be carried out on a sufficiently large application (the investigations into MiniSAT only optimised 478 lines of code). As software size increases the necessity of profiling software to direct GI to the most costly areas of code becomes necessary though research into tools for profiling software's energy consumption has largely been ignored until recently. Ultimately new methods for measuring or estimating software's energy consumption at a finer granularity is required for better optimisation to take place.

Some readers may also point out the irony in stressing the end of the"Goldilocks period" of software development then subsequently touting an investigation into MiniSAT running on a Macbook Pro. This irony has not gone unnoticed and next stages of research must move to optimising software on other devices. Academics, like real-world developers, also have difficulty moving from the comfortable and well-supported Goldilocks development zone.

4. CONCLUSION

Within this paper we have advocated the position that GI, a SBSE technique, has the potential to offer software developers a method of automatically optimising their software to reduce energy consumption while maintaining functional properties. As developers shift from an environment primarily focused on desktop application development towards one centred on development for the small and mobile to the large and stationary, techniques such as GI are needed more than ever. Therefore future investigations into improving energy efficiency using GI are necessary.

5. REFERENCES

[1] B. R. Bruce, J. Petke, and M. Harman. Reducing Energy Consumption Using Genetic Improvement. In *GECCO 2015*, 2015. To appear.

[2] C. Bunse, H. Höpfner, S. Roychoudhury, and E. Mansour. Choosing the "best" sorting algorithm for optimal energy consumption. *ICSOFT*, 2009.

[3] J. Koomey. Growth in data center electricity use from 2005 to 2010, Aug. 2011.

[4] W. B. Langdon and M. Harman. Evolving a CUDA kernel from an nVidia template. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, July 2010.

[5] W. B. Langdon and M. Harman. Optimising Existing Software with Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 2013.

[6] I. Manotas, L. Pollock, and J. Clause. SEEDS: a software engineer's energy-optimization decision support framework. In *Proceedings of ICSE 2014*, pages 503–514, New York, New York, USA, May 2014. ACM Press.

[7] J. Petke, W. B. Langdon, M. Harman, and W. Weimer. Using genetic improvement & code transplants to specialise a C++ program to a problem class. In *Proceedings of EuroGP 2014*, Granada, Spain, 2014.

[8] E. Schulte, J. Dorn, S. Harding, S. Forrest, and W. Weimer. Post-compiler software optimization for reducing energy. In *Proceedings of ASPLOS 2014*, pages 639–652. ACM, 2014.

[9] W. G. P. Silva, L. Brisolara, U. B. Corrêa, and L. Carro. Evaluation of the impact of code refactoring on embedded software efficiency. In *Proceedings of the 1st Workshop de Sistemas Embarcados*, pages 145–150, 2010.

[10] D. R. White, A. Arcuri, and J. A. Clark. Evolutionary improvement of programs. *IEEE Transactions on Evolutionary Computation*, 15(4):515–538, Aug. 2011.