2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

Helping Programmers Improve the Energy Efficiency of Source Code

Rui Pereira*, Tiago Carção*, Marco Couto*, Jácome Cunha‡, João Paulo Fernandes§, and João Saraiva*
* HASLab/INESC TEC, Universidade do Minho, Portugal
‡ NOVA LINCS, DI, FCT, Universidade NOVA de Lisboa, Portugal
§ Release/LISP, CISUC, Universidade de Coimbra
{ruipereira,saraiva}@di.uminho.pt, marco.l.couto@inesctec.pt, jacome@fct.unl.pt, jpf@dei.uc.pt

***Abstract*—This paper briefly proposes a technique to detect energy inefficient fragments in the source code of a software system. Test cases are executed to obtain energy consumption measurements, and a statistical method, based on spectrum-based fault localization, is introduced to relate energy consumption to the system's source code. The result of our technique is an energy ranking of source code fragments pointing developers to possible energy leaks in their code.**

***Keywords*—Green Computing; Program Optimization; Fault Localization**

I. Introduction

In recent years, awareness within society of the significant side-effects of energy demands has grown, acknowledging the need for sustainable software development [1]. In fact, software developments are keen on developing energy-efficient software [2], and a long list of (mostly recent) efforts that include [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] has tried to provide developers with the libraries, tools, techniques and data to support energy-aware development. Even considering these efforts, the green computing research area is still at an early stage where research issues, challenges and opportunities abound [16], [17], [18].

This paper introduces an in development technique and tool, named SPELL - SPectrum-based Energy Leak Localization, to determine *red* (energy inefficient) areas in software. In this context, a parallel is made between the detection of anomalies in the energy consumption of software during program execution, and the detection of faults in the execution of a program. Having this parallelism established, we adapted fault detection techniques, often used to investigate software bugs or failures in program executions, to detect energetic faults in programs.

Our proposed technique is language independent, allowing the analysis of any programming language as long as we have the needed input data (energy consumption, execution time, etc.). Additionally, it is also context independent, allowing it to be applied to detect *red* areas on various levels of code. This means we could use it to detect the inefficiencies at different granularity levels, be that packages, classes, methods, functions, lines of code, etc. Even more so, the technique allows the presence of different hardware component's energy values (CPU, DRAM, Fans, Hard Drive, and GPU), and may return the analysis of one specific factor (energy, time, or number), or a global analysis considering all three factors.

A software system is executed with a set of test cases, and components of such system (for example, packages, functions, loops, etc) are instrumented to estimate/measure their energy consumption at runtime. Inefficient energy consumption, the so-called *energy leaks*[1], are interpreted in SPELL as program faults, and adapting Spectrum-based Fault Localization (SFL) techniques [19], [20] to relate energy consumption to the system's source code. Our analysis associates different percentage of responsibility for the energy consumed to the different components of the underlying system. Thus, the result of our analysis is a ranking of components sorted by their likelihood of being responsible for energy leaks, essentially pinpointing and prioritizing the developer's attention on the most critical *red* spots in the analyzed system, and giving him more useful information to better support him in making decisions of what parts of the system he needs to optimize.

Supported by our developed tool, our technique was able to identify potential energy leaks in the source code of concrete Java projects in a preliminary study. Based on this identification, a set of expert Java programmers was then asked to improve the efficiency of those projects with and without knowledge of the identified energy leaks. The analysis of their performance in doing so provided statistical evidence that experts with access to located energy leaked were able to better optimize the energy consumption of those projects, and were much faster doing so. This initial study has shown that using our technique helped developers identify and optimize energy problems in 50% less time, while optimizing the energy consumption on average by 18%. Data on one of the projects can be seen in Table I, with the global energy consumption (J) and execution time (ms) for both the original project, and refactored versions (using and not using SPELL), respectively. The table also details the GPS-UP Software Energy Efficient metrics [21] for each version.

Another interesting observation that can be drawn by our case study confirms that optimizing for energy consumption

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-016718 and UID/CEC/04516/2013; and by FLAD/NSF under the project Software Repositories for Green Computing, ref. 300/2015. The first author is also sponsored by FCT grant SFRH/BD/112733/2015.

[1] In this context, an energy leak is essentially a part of the program where it is consuming energy more than it probably should. As if one were to imagine a cup of water, with water leaking over when it should not.

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.80

TABLE I: Study results from one of the Projects in the preliminary study

		Original		SPELL – Time taken: 1h04								No SPELL – Time taken: 1h58							
						Gain (%)		Energy Metrics						Gain (%)		Energy Metrics			
	Test	J	ms	J	ms	J	ms	GU	SU	PU	Cat	J	ms	J	ms	GU	SU	PU	Cat
Project P_1	1	13.6	1341	9.7	959	28.3	28	1.39	1.40	1.00	3	13.3	1324	1.7	1	1.02	1.01	0.99	1
	2	4.6	314	4.3	317	4.9	-1	1.05	0.99	0.94	4	4.9	339	-8.3	-8	0.92	0.93	1.00	8
	3	7.0	695	4.9	482	30.0	31	1.43	1.44	1.01	3	7.1	690	-2.2	1	0.98	1.01	1.03	5
	4	7.1	691	6.5	583	8.4	16	1.09	1.19	1.09	3	7.1	683	0.1	1	1.00	1.01	1.01	3
	5	25.8	2557	21.5	1603	16.5	37	1.20	1.59	1.33	3	25.6	2538	0.7	1	1.01	1.01	1.00	3
	6	20.8	1469	18.2	1808	12.5	-23	1.14	0.81	0.71	4	19.4	1923	6.8	-31	1.07	0.76	0.71	4
	7	3.5	315	3.3	283	7.2	10	1.08	1.11	1.03	3	3.1	304	11.0	3	1.12	1.04	0.92	1
	Total	82.4	7381	68.5	6037	16.8	18	1.20	1.22	1.02	3	80.7	7801	2.1	-6	1.02	0.95	0.93	4

is not always equivalent to optimizing runtime execution [4], [22], [13]. Indeed, with our technique, programmers were able to improve the energy efficiency of projects whose runtime performance of some actually degraded as a consequence of this improvement.

II. Spectrum-based Energy Leak Localization

A. Spectrum-based Fault Localization

Our technique, *SPELL*, is based on spectrum-based fault localization [20], [19], a state of the art technique which uses statistical analysis [23] and execution trace to identify faults in a program's implementation (source code). SFL uses a simple hit spectrum (flag which reflects if a certain component is used or not in a particular execution) to build a matrix A of dimension $n \times m$, where m represents the different components (e.g. methods, classes,etc.) of a program during n independent test executions. Complementing the hit spectrum, SFL uses an error vector to indicate whether each of the n tests succeeded or not. Finally, it applies a coefficient of similarity to calculate which component is the most probable to be faulty.

B. Spectrum-based Energy Leak Localization

In this context, a parallel is made between the detection of faults in the execution of a program with the detection of anomalies in the energy consumption of a program. Having this parallelism established, these fault detection techniques, were adapted to detect *energy leaks*.

In *SPELL*, while it too uses the concept of m components (e.g. programs, packages, classes, methods, statements) and n independent tests (which can be test cases or program simulations), it differs in several ways. The hit spectrum elements for our *SPELL* matrix is not a single flag, but holds a triple of three categories: ($Energy$, $Time$, $Number$). These are expressed in Joules, milliseconds, and number of executions respectively. Additionally, our $Energy$ category is too a tuple which may represent the consumption by each different hardware component (CPU, DRAM, GPS, GPU, screen, etc.) if the chosen energy measurement technique allows this differentiation.

Another difference lies in how the oracle is calculated. While in SFL there is an error vector to reason about the validity of the output obtained during a test, the *SPELL* analysis does not receive this as an input. This is attributed to there being no clear signal as to what can be seen as an excess of energy consumption. Therefore, an error vector is calculated by this technique, a criterion to represent the *greenness* of a component instead of a binary decision, and two different perspectives to calculate the oracle and similarity. These perspectives are called *Component Category Similarity* and *Global Similarity*, an analysis on one specific category (for example only considering energy consumption) or a global analysis considering all three, respectively. These similarity functions are inspired by the Jaccard similarity coefficient [24].

A software developer can now, for example, use jRAPL [25] or the ODROID-XU3[2] to measure his/her program's energy consumption on a method level (here a component m would be a method), with various simulations or tests (n), and obtain a ranking of components sorted by their likelihood of being responsible for the program's energy leak, pinpointing and prioritizing the developer's attention on the most probably hot spot. This gives him/her more useful information to better support the decision making of what and where to optimize.

This language independent technique only requires an input matrix representing the tests, components, and category values. SPELL is currently implemented in Java as a tool-kit containing the implementation of the core technique along with other helpful tools, such as a jRAPL method instrumentation tool and can be found by following the link in the footnote[3].

III. Conclusion

This paper briefly introduced SPELL - a spectrum-based energy leak localization technique to identify inefficient energy consumption in the source code of software systems. This technique is both language independent and context independent, using a statistical method to associate different percentages of responsibility for the energy consumed to the different source code components of a software system, thus pinpointing the developer's attention on the most critical "red" spots. Such software components may be program modules, packages, functions, source code fragments, wherever the developer wishes to detect energy leaks.

Preliminary empirical studies with Java programmers showed that not only can this technique be used in various contexts, but also that developers who used *SPELL* were able to find and optimize a program's energy consumption and performance, spending 50% less time and improving the consumption on average by 18% when compared to those who did not use *SPELL*. These studies also showed that some of the energy optimization that were achieved by programmers actually produced more energy efficient while degrading the runtime performance, helping developers find and optimize a program's energy consumption with good results and indicators of where problems are occurring.

[2]https://www.hardkernel.com

[3]https://github.com/greensoftwarelab/SPELL

REFERENCES

[1] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, M. Mahaux, B. Penzenstadler, G. Rodríguez-Navas, C. Salinesi, N. Seyff, C. C. Venters, C. Calero, S. A. Koçak, and S. Betz, "The karlskrona manifesto for sustainability design," *CoRR*, vol. abs/1410.6968, 2014.

[2] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 22–31.

[3] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser, "Seflab: A lab for measuring software energy footprints," in *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*. IEEE, 2013, pp. 30–37.

[4] G. Pinto, F. Castor, and Y. D. Liu, "Understanding energy behaviors of thread management constructs," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 2014, pp. 345–360.

[5] T. Yuki and S. Rajopadhye, "Folklore confirmed: Compiling for speed= compiling for energy," in *Languages and Compilers for Parallel Computing*. Springer, 2014, pp. 169–184.

[6] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 2–11.

[7] C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, p. 36.

[8] C. Sahin, P. Tornquist, R. McKenna, Z. Pearson, and J. Clause, "How does code obfuscation impact energy usage?" in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2014.

[9] M. Couto, T. Carção, J. Cunha, J. P. Fernandes, and J. Saraiva, *Programming Languages: 18th Brazilian Symposium, SBLP 2014, Maceio, Brazil, October 2-3, 2014. Proceedings*. Cham: Springer International Publishing, 2014, ch. Detecting Anomalous Energy Consumption in Android Applications, pp. 77–91. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11863-5_6

[10] I. Manotas, L. Pollock, and J. Clause, "Seeds: A software engineer's energy-optimization decision support framework," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014.

[11] A. Hindle, "Green mining: a methodology of relating software change and configuration to power consumption," *Empirical Software Engineering*, vol. 20, no. 2, pp. 374–409, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9276-6

[12] S. Li and S. Mishra, "Optimizing power consumption in multicore smartphones," *Journal of Parallel and Distributed Computing*, pp. –, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731516000198

[13] L. G. Lima, F. Soares-Neto, P. Lieuthier, F. Castor, G. Melfe, and J. P. Fernandes, "Haskell in green land: Analyzing the energy behavior of a purely functional language," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 517–528.

[14] R. Pereira, M. Couto, J. a. Saraiva, J. Cunha, and J. a. P. Fernandes, "The influence of the java collection framework on overall energy consumption," in *Proceedings of the 5th International Workshop on Green and Sustainable Software*, ser. GREENS '16. New York, NY, USA: ACM, 2016, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/2896967.2896968

[15] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul, and G. M. Voelker, "edoctor: Automatically diagnosing abnormal battery drain issues on smartphones," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 57–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482634

[16] A. E. Trefethen and J. Thiyagalingam, "Energy-aware software: Challenges, opportunities and strategies," *Journal of Computational Science*, vol. 4, no. 6, pp. 444 – 449, 2013.

[17] P. Lago, "Challenges and opportunities for sustainable software," in *Proceedings of the Fifth International Workshop on Product LinE Approaches in Software Engineering*, ser. PLEASE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 1–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820656.2820658

[18] A. Hindle, "Green software engineering: the curse of methodology," *PeerJ PrePrints*, vol. 3, p. e1832, 2015.

[19] R. Abreu, P. Zoeteweij, and A. J. C. v. Gemund, "Spectrum-based multiple fault localization," in *Proc. of the 2009 IEEE/ACM Int. Conf. on Automated Software Engineering*, ser. ASE '09. Washington, USA: IEEE Computer Society, 2009, pp. 88–99.

[20] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "On the accuracy of spectrum-based fault localization," pp. 89–98, 2007.

[21] S. Abdulsalam, Z. Zong, Q. Gu, and M. Qiu, "Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency," in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*. IEEE, 2015, pp. 1–8.

[22] A. E. Trefethen and J. Thiyagalingam, "Energy-aware software: Challenges, opportunities and strategies," *Journal of Computational Science*, vol. 4, no. 6, pp. 444–449, 2013.

[23] A. X. Zheng, M. I. Jordan, B. Liblit, and A. Aiken, "Statistical debugging of sampled programs," in *Advances in Neural Information Processing Systems*, 2003, p. None.

[24] R. Real and J. M. Vargas, "The probabilistic basis of jaccard's index of similarity," *Systematic biology*, pp. 380–385, 1996.

[25] K. Liu, G. Pinto, and Y. D. Liu, "Data-oriented characterization of application-level energy optimization," in *Fundamental Approaches to Software Engineering*. Springer, 2015, pp. 316–331.