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

Locating Energy Hotspots in Source Code

Rui Pereira
HASLab/INESC TEC & Universidade do Minho, Portugal
ruipereira@di.uminho.pt

I. Motivation

We as a society have built our exceptional development pace on top of a widespread use of energy resources. Unfortunately, over the past number of years we have begun to understand this problem of growing energy demands and concern ourselves over the impact is has done to the environment, such as global warming.

Much of this can be attributed to the exponential growth we are witnessing in the ICT sector. A contribution of 8% of the overall energy consumption comes from this [1], as does 50% of the energy costs of an organization [2]. While computer sciences and environmentalism isn't a common pair, this specific context has raised awareness [3], [4], and the realization of the need for sustainable software development [5].

As a matter of fact, software developers have shown to be eager to develop energy-efficient software [6], with recent efforts including [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] to provide libraries, measurement tools, techniques and data to help energy-aware development. Even so, the green computing research area is still in initial stages with continuous issues, challenges, and opportunities to tackle [20], [21], [22].

Research has defined many techniques to improve software developers' programs with optimal runtime, for example, partial and/or runtime compilation, advanced garbage collectors, parallel execution, etc. Additionally, techniques to improve their productivity such as advanced type and modular systems, IDEs, and testing and debugging framework have also been developed. Comparing this to energy-aware software engineering, we see a clear deficit with the former being clearly complete in terms of what it has achieved [21].

Studies have shown how different design patterns, using Model-View-Controller, information hiding, implementation of persistence layers, code obfuscations, refactorings, and different Java based collections have a statistically significant impact on energy usage [23], [24], [25], [10], [14], [11], [12].

While developers have several ways of obtaining energy consumption readings [7], [24], [26], [27], [28], [29], or model based energy estimation [30], [31], the notion of what they mean, how to interpret, and what relevance the consumption of certain software components have in the program's consumption is yet to be provided and analyzed.

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 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.

This paper briefly touches on a in development technique named *SPELL* - SPectrum-based Energy Leak Localization, to identify critical *red* (energy hotspots) areas in software. This technique is both language and context independent, and based on the spectrum-based fault localization technique, a state of the art technique to identify program faults in program execution [32], [33]. *SPELL* uses a statistical method to associate different percentage 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 in his code. Initial studies have also showed that using this technique helped developers identify and optimize energy problems in 50% less time and optimizing the energy consumption on average by 18% when compared to those who did not use *SPELL* [34].

II. Spectrum-based Localization

A. Spectrum-based Fault Localization

Our technique, *SPELL*, is based on spectrum-based fault localization [32], [33], a state of the art technique which uses statistical analysis [35] 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, often used to detect software bugs or failures, were adapted to detect *energy leaks*[1].

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

[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.2017.151

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.

A software developer can now, for example, use jRAPL [29] 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].

C. Integrated Development Environment

The current implementation of the *SPELL* technique displays the responsibility percentages and analysis in a textual format. As the main motivation of this work was to produce a tool to help developers become energy-aware while developing software, different ways of how to display this information is being discussed. Currently, a prototype linking *SPELL* and GZoltar has been developed [36] so that we freely inherit a graphical visualization tool which allows, with the aid of a sunburst graph and flagging (as shown in Figure 1), hierarchical navigation of code. Using colors (red for problems and green otherwise), it shows immediate information on the status of different parts of the code.

III. Conclusion

This paper briefly introduces *SPELL*, a spectrum-based energy leak localization technique to identify inefficient energy consumption in source code. Using a statistical method to associate different percentages of responsibility to different components within a program, it can help focus the developer's attention on the most critical areas. This technique is both language independent, and context independent (applicable on any level of a program: packages, classes, methods, etc.). This paper also shows a possible way to represent the information from *SPELL*'s analysis in an easy to understand graphical representation to further help the developer become energy-aware.

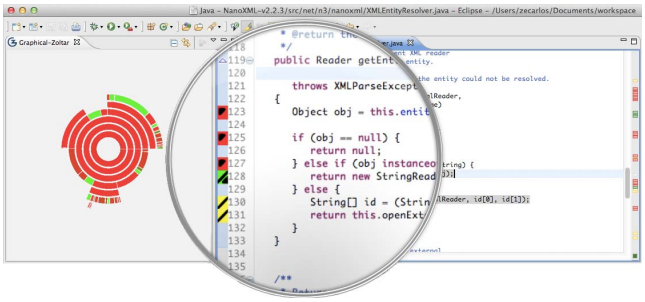

Fig. 1: *SPELL* embedded Gzoltar graphic visualization tool

The main difference between this technique and those aforementioned or existing, is that it is not a new form of estimating or measuring the energy consumption, but a technique which using various energy measurements calculates the probability of a component having an energy problem, and presents this directly to the developer. The technique is for developers, unlike works such as [19], to essentially debug their programs, independent of language and applicable on any abstract level (class, method, line, etc.); and without needing to know energy inefficient practices a priori such as [37]. It does not give energy readings, but reasons about variables such as energy and time to present a target area of where one should focus their attention to optimize, parallel to how SFL digests test cases to attribute a probability to a component as being faulty. This reasoning can be focused on one category ($Energy$, $Time$, or $Number$), for an optimization on any of the three, or focused on a global analysis of all three.

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* [34]. 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. This further helps show that the already existing profilers are not enough for energy consumption optimization, as optimizing for energy consumption does not always implicate performance optimization.

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

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

References

[1] H. T. Mouftah and B. Kantarci, "Chapter 11 - Energy-Efficient Cloud Computing: A Green Migration of Traditional {IT}," in *Handbook of Green Information and Communication Systems*, M. S. Obaidat, A. Anpalagan, and I. Woungang, Eds. Academic Press, 2013, pp. 295–330.

[2] R. R. Harmon and N. Auseklis, "Sustainable it services: Assessing the impact of green computing practices," in *Management of Engineering & Technology, 2009. PICMET 2009. Portland International Conference on*. IEEE, 2009, pp. 1707–1717.

[3] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 437–445, Dec 1994.

[4] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. ACM, 2003.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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

[14] 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.

[15] 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

[16] 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

[17] 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.

[18] 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

[19] 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

[20] 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.

[21] 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

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

[23] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "The impact of source code transformations on software power and energy consumption," *Journal of Circuits, Systems, and Computers*, 2002.

[24] D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 2013, pp. 78–89.

[25] C. Sahin, F. Cayci, I. L. M. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. IEEE, 2012, pp. 55–61.

[26] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using RAPL," *SIGMETRICS Performance Evaluation Review*, vol. 40, no. 3, pp. 13–17, 2012. [Online]. Available: http://doi.acm.org/10.1145/2425248.2425252

[27] A. Noureddine, R. Rouvoy, and L. Seinturier, "Monitoring energy hotspots in software," *Automated Software Engineering*, pp. 1–42, 2015.

[28] N. Grech, K. Georgiou, J. Pallister, S. Kerrison, J. Morse, and K. Eder, "Static analysis of energy consumption for llvm ir programs," in *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '15. New York, NY, USA: ACM, 2015, pp. 12–21. [Online]. Available: http://doi.acm.org/10.1145/2764967.2764974

[29] 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.

[30] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. of the 8th Int. Conf. on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2010, part of ESWeek '10 Sixth Embedded Systems Week, Scottsdale, AZ, USA, October 24-28, 2010*.

[31] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 92–101.

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

[33] 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.

[34] R. Pereira, T. Carção, M. Couto, J. Cunha, J. ao Paulo Fernandes, and J. ao Saraiva, "Mind the leak: Helping programmers improve the energy efficiency of source code," in *Proceedings of the 39th International Conference on Software Engineering*, 2017, to appear.

[35] 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.

[36] J. Campos, A. Riboira, A. Perez, and R. Abreu, "Gzoltar: an eclipse plug-in for testing and debugging," in *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, M. Goedicke, T. Menzies, and M. Saeki, Eds. ACM, 2012, pp. 378–381.

[37] Y. Liu, C. Xu, S.-C. Cheung, and J. Lü, "Greendroid: Automated diagnosis of energy inefficiency for smartphone applications," *IEEE Transactions on Software Engineering*, vol. 40, no. 9, pp. 911–940, 2014.