Invited talk: Analysing energy consumption of systems controlled by software

Bernard van Gastel
Open University of the Netherlands and Radboud University, the Netherlands
The Netherlands
bvg@ou.nl,B.vanGastel@cs.ru.nl

ABSTRACT

Energy consumption analysis of IT-controlled systems can play a major role in minimising the overall energy consumption of such IT systems, during the development phase, or for optimisation in the field. As software is increasingly embedded in our daily life, with IT using more and more energy, the software industry should become aware of their energy footprint, and methods must be developed to assist in reducing this footprint.

Recently, we developed a precise energy analysis, to analyse software in conjunction with hardware. It has the property of being parametric with regard to the hardware. In principle, this creates the opportunity to investigate which is the best software implementation for given hardware, or the other way around: choose the best hardware for a given algorithm.

CCS CONCEPTS

• **Software and its engineering → Formal software verification**; *Operational analysis*; • **Hardware → Platform power issues**;

ACM Reference format:
Bernard van Gastel. 2017. Invited talk: Analysing energy consumption of systems controlled by software. In *Proceedings of Programming '17, Brussels, Belgium, April 03-06, 2017,* 2 pages.
https://doi.org/http://dx.doi.org/10.1145/3079368.3079396

1 THE CASE FOR AWARENESS IN THE SOFTWARE INDUSTRY OF ITS ENERGY FOOTPRINT

Energy analysis of IT systems is an important emerging field. Its focus is on analysing the software that controls the IT system using models of the components of the system under analysis. Components can vary from small components such as an Internet of Things sensor to large subsystems as present in self-driving cars.

As traditionally many savings did occur on the hardware side of a computer, energy consumption is almost a blind spot when developing software. Each next hardware generation consumed less energy to perform the same amount of work. However, recently this development has lost its pace. At the same time, it becomes more and more clear that software has a huge impact on the behaviour and the properties of devices it runs on. A recent example of software influencing the working of a device is the Volkswagen scandal. The car manufacturer used software to detect if the car was being tested. If this was found to be the case, the diesel motor was programmed to operate in such a way that it exhausted less toxic gases and fumes. In [8] it is calculated that 44,000 years of human life are lost in Europe because of the fraud, which lasted at least six years. Another example is the control software as used in fridges from Panasonic, which could detect if a test was going on and suppressed energy intensive defrost cycles during this test. These are negative examples, but they do make clear that the software is in control of the device and its (energy) behaviour.

Although the software is evidently in control of the devices, there is almost no time dedicated in most computer science curricula to the energy efficiency of software. This is peculiar since energy is of vital importance to the modern (software) industry. For years, data centres have been located at places where the energy is cheap, and since the rise of the smartphone more software engineers recognise that to get good user reviews, their software should not rapidly deplete the battery charge of the user's phone. Due to the lack of educational attention to energy-aware programming, most aspiring programmers never learn to produce energy efficient code. Software engineers have trouble assessing how much energy will be consumed by their software on a target device, especially when the software is run on a multitude of different systems. With the advent of the Internet of Things, where software is increasingly embedded in our daily life, the *software industry should become aware of their energy footprint*, and methods must be developed to assist in reducing this footprint.

Furthermore, the combination of many individual negative effects can also affect our society at large. Although this effect is less direct, it is no less essential. If devices that are present in large quantities in our society all exhibit the same negative behaviour, such as incurring needlessly a too high energy consumption, they can impact public utilities and our economy. They will consume the finite resources of Earth even faster. Governments increasingly recognise this societal effect, as indicated by the new laws in the European Union issuing ecodesign requirements for many kinds of devices. One of the aims of these requirements is to make devices more energy efficient. Examples of product categories with ecodesign requirements include vacuum cleaners, electrical motors, lighting, heaters, cooking appliances, televisions and coffee machines. Even requirements leading to relative small improvements in energy efficiency can yield large results at scale, even in the case of devices of which one would expect no significant electricity savings to be possible.

Programming '17, April 03-06, 2017, Brussels, Belgium
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4836-2/17/04.
https://doi.org/http://dx.doi.org/10.1145/3079368.3079396

Modern devices and appliances are controlled by software, which makes analysing the energy consumption challenging of these devices, as the behaviour of its software is difficult to predict. To analyse the consumption of hardware, the software controlling the hardware needs to be analysed together with the hardware.

2 INTRODUCING ECA: HYBRID ENERGY ANALYSIS OF SOFTWARE CONTROLLED SYSTEMS

To analyse the hardware and software together, we propose in [2, 3, 6, 11] a hybrid approach, joining energy behaviour models of the hardware with the energy-aware semantics of software and a program transformation. Hardware is modelled as a finite state machine, with both the states and transitions labelled with energy consumption. The interface between hardware and software is made explicit and has to be well defined, enabling both our analysis and easy exchanging of hardware or software components. Using this parametric approach, multiple implementations can be analysed. Such an approach can be used on design level or for optimisation. One can e.g. choose the best software implementation for given hardware, or the other way around: choose the best hardware for a given algorithm.

The programs that can be analysed are written in the software language ECA, which is an imperative language inspired by C and Java. In this language, all interactions with hardware components are made explicit. We use the notation $C::f$ to refer to a function `f()` operating on a component C. Multiple different hardware components can be used simultaneously from the same program, the components are differentiated by a unique name (substituted in the rules for C). Besides this differentiator, the ECA language is a fairly default imperative language sporting recursive functions, loops data structures.

Based on this language, (natural) semantics of this language are defined, which includes energy consumption. A program transformation is given, based on the energy-aware semantics. This transformation derives an energy function which is executable. If both a concrete input and one or multiple hardware models are specified, the parametric function will result in the energy consumption occurring when running the software on the given hardware. One can view these derived energy functions to signify the energy behaviour when the software is executing and controlling the hardware.

The hardware conceptually consists of a *component state* and a set of *component functions* which operate on the component state. We use finite state models to model these hardware components, with the transitions constituting function calls on the hardware. Energy usage is expressed by labelling both the vertices and edges with energy consumption, which can be in any unit. Labels on vertices constitute time-bound energy consumption, i.e. power draw. Edges are labelled with the consumption of a certain amount of energy, not time-bound but corresponding to the transition. Depending on your needs, you can model energy consumption in one way or the other. Each edge corresponds to an explicit call to a component function in the source code. Besides the ones above described, there are no additional requirements. We use the power draw function ϕ which translates a component state to a power draw. The result of this function is used to calculate energy consumption for the time spent in a specific state.

3 OTHER APPROACHES

Besides our approach to energy analysis, there are other approaches. The programmer can look for programming guidelines and design patterns, which in most cases produce more energy-efficient programs, e.g. [9, 10]. Then, he/she might make use of a compiler that optimises for energy-efficiency, e.g. [12]. If the programmer is lucky, there is an energy analysis available for the specific platform at hand, such as [5].

However, for most platforms, this is not a viable option. In that case, the programmer might use dynamic analysis with a measurement set-up. This, however, is not a trivial task and requires a complex set-up [1, 4]. Moreover, it only yields information for a specific benchmark [7]. Nevertheless, these approaches are always applicable. A programmer might, however, prefer an approach that yields additional insight in a more predictive manner.

REFERENCES

[1] Miguel A Ferreira, Eric Hoekstra, Bo Merkus, Bram Visser, and Joost Visser. 2013. SEFLab: A lab for measuring software energy footprints. In *2nd International Workshop on Green and Sustainable Software (GREENS), 2013*. IEEE, 30–37. https://doi.org/10.1109/GREENS.2013.6606419

[2] Bernard van Gastel. 2016. *Assessing sustainability of software; analysing correctness, memory and energy consumption.* Ph.D. Dissertation. Open Universiteit. http://sustainablesoftware.info/download/thesis-met-cover.pdf

[3] Bernard van Gastel and Marko van Eekelen. 2017. Towards practical, precise and parametric energy analysis of IT controlled systems. In *Proceedings of the Fifth International Workshop on Foundational and Practical Aspects of Resource Analysis (FOPARA'17).*

[4] Erik A. Jagroep, Jan Martijn van der Werf, Sjaak Brinkkemper, Giuseppe Procaccianti, Patricia Lago, Leen Blom, and Rob van Vliet. 2016. Software Energy Profiling: Comparing Releases of a Software Product. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16).* ACM, New York, NY, USA, 523–532. https://doi.org/10.1145/2889160.2889216

[5] Ramkumar Jayaseelan, Tulika Mitra, and Xianfeng Li. 2006. Estimating the Worst-Case Energy Consumption of Embedded Software. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium.* IEEE, 81–90. https://doi.org/10.1109/RTAS.2006.17

[6] Rody Kersten, Paolo Parisen Toldin, Bernard van Gastel, and Marko van Eekelen. 2014. A Hoare Logic for Energy Consumption Analysis. In *Proceedings of the Third International Workshop on Foundational and Practical Aspects of Resource Analysis (FOPARA'13) (LNCS)*, Vol. 8552. Springer, 93–109. https://doi.org/10.1007/978-3-319-12466-7_6

[7] F. A. Moghaddam, T. Geenen, P. Lago, and P. Grosso. 2015. A user perspective on energy profiling tools in large scale computing environments. In *Sustainable Internet and ICT for Sustainability (SustainIT), 2015.* 1–5. https://doi.org/10.1109/SustainIT.2015.7101364

[8] Rik Oldenkamp, Rosalie van Zelm, and Mark A.J. Huijbregts. 2016. Valuing the human health damage caused by the fraud of Volkswagen. *Environmental Pollution* 212 (2016), 121 – 127. https://doi.org/10.1016/j.envpol.2016.01.053

[9] Eric Saxe. 2010. Power-efficient software. *Commun. ACM* 53, 2 (2010), 44–48. https://doi.org/10.1145/1646353.1646370

[10] Steven te Brinke, Somayeh Malakuti, Christoph Bockisch, Lodewijk Bergmans, and Mehmet Aksit. 2013. A design method for modular energy-aware software. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*, Sung Y. Shin and José Carlos Maldonado (Eds.). ACM, 1180–1182. https://doi.org/10.1145/2480362.2480584

[11] Bernard van Gastel, Rody Kersten, and Marko C. J. D. van Eekelen. 2015. Using Dependent Types to Define Energy Augmented Semantics of Programs. In *Foundational and Practical Aspects of Resource Analysis - 4th International Workshop, FOPARA 2015, London, UK, April 11, 2015, Revised Selected Papers (Lecture Notes in Computer Science)*, Marko C. J. D. van Eekelen and Ugo Dal Lago (Eds.), Vol. 9964. 20–39. https://doi.org/10.1007/978-3-319-46559-3_2

[12] Dmitry Zhurikhin, Andrey Belevantsev, Arutyun Avetisyan, Kirill Batuzov, and Semun Lee. 2009. Evaluating power aware optimizations within GCC compiler. In *GROW-2009: International Workshop on GCC Research Opportunities.* 1–9. https://doi.org/10.1.1.470.8078