Specialising Guava's Cache to Reduce Energy Consumption

Nathan Burles[1](✉), Edward Bowles[1], Bobby R. Bruce[2], and Komsan Srivisut[1]

[1] University of York, York YO10 5DD, UK
{nathan.burles,eab530,ks1077}@york.ac.uk
[2] CREST Centre, University College London, London WC1E 6BT, UK
r.bruce@cs.ucl.ac.uk

Abstract. In this article we use a Genetic Algorithm to perform parameter tuning on Google Guava's Cache library, specialising it to OpenTripPlanner. A new tool, Opacitor, is used to deterministically measure the energy consumed, and we find that the energy consumption of OpenTripPlanner may be significantly reduced by tuning the default parameters of Guava's Cache library. Finally we use Jalen, which uses time and CPU utilisation as a proxy to calculate energy consumption, to corroborate these results.

Keywords: Parameter tuning · Library specialisation · Energy profiling · Reduced power consumption

1 Introduction

The practice of releasing software with configurable parameters is common. This is due to the widely accepted belief that few pieces of software are truly optimal for all situations and therefore an interface is required to allow a more optimal solution to be deployed. Configurable parameters allow developers to release software for general use instead of developing multiple versions, each tailored to a specific environment.

The issue with this model is that few know how to properly tune parameters. To do so requires in-depth knowledge of the software to be configured and the domain in which it is to be deployed. It is for this reason Automatic Parameter Tuning is advantageous. Automated Parameter Tuning is the automatic process of tweaking software parameters until optimal (or near optimal) configurations are found depending on non-functional (and occasionally functional) properties desired by the user.

Research into parameter tuning has focused primarily on execution time [9,17–19], memory consumption [17], and occasionally functional attributes such as output precision [7]. The rise in mobile computing technology [6] with limited battery life, and growth in large server farms responsible for consuming large amounts of energy [10] has sparked a new wave of research into energy efficient software [2,7,12,15]. For this reason we wish to use SBSE to tune and specialise parameters to reduce energy consumption.

This paper will outline a method of optimising parameters to reduce energy consumption using Genetic Algorithms (GAs) to tune Google Guava's[1]

[1] Available at https://github.com/google/guava.

M. Barros and Y. Labiche (Eds.): SSBSE 2015, LNCS 9275, pp. 276–281, 2015.
DOI: 10.1007/978-3-319-22183-0_23

`CacheBuilder` class as used by OpenTripPlanner[2]. Guava's Cache is similar to a map, in that it stores a set of associated keys and values. They differ mainly in their approach to persistence—a map retains all stored associations until they are explicitly removed, whereas a Cache generally evicts entries automatically in order to conserve memory.

The ECJ Toolkit. We have chosen to use ECJ [11] to implement the GA as it is one of the most popular Java-based toolkits for evolutionary computation [4,20]. Using ECJ requires little setup, only the configuration of parameters; the selection of the desired Evolutionary Algorithm, the Evolutionary Algorithm parameters (population size, mutation rate, etc.), and the desired fitness evaluation function. For our requirements ECJ serves as a black-box Evolutionary Algorithm which we trust to be fully tested and reliable.

2 Related Work

Previous work in tuning parameters to reduce energy consumption has been successful, albeit in a round-about manner. In 2011 Hoffmann et al. introduced PowerDial [7], a system for dynamically modifying trade-offs between accuracy in computation and use of system resources during load peaks. Though not directly reducing energy consumption per se, the framework aims to reduce the amount of computing infrastructure required to manage load peaks in server farms that translates to significant reductions in energy consumption (as well as capital costs).

Optimising other non-functional attributes using automatic parameter tuning has also been successful. Wu et al. [21] used Genetic Algorithms to tune both "shallow" and "deep" parameters for both execution time and memory consumption in the widely used *dlmalloc* memory allocator. They were able to show clear trade-offs between these two attributes with possible configuration options resulting in up to 21 % reduction in memory consumption and a 12 % reduction in execution time.

Although it is a relatively new area of interest, using SBSE techniques such as GAs to reduce energy consumption has previously been used by both Schulte et al. in 2014 [15] and Bruce et al. in 2015 [2] to reduce energy consumption as a post- and pre-compilation process respectively.

3 Implementation

The improvement process is as follows:

1. Find variation points within the `CacheBuilder` class, i.e. the declaration of default values for parameters, and identify the valid values for each of these parameters, for example integers: `int initialCapacity = ?`, or enumerated types: `Strength keyStrength = Strength.{strong, weak, soft}`.
2. Generate a template version of the `CacheBuilder` class to allow the variation points to be easily replaced by the respective element in a solution vector.
3. Given k parameters, the solution representation is a vector containing k integers $[S_1, \ldots, S_k]$, where S_k is in the range identified earlier.

[2] Available at http://www.opentripplanner.org.

4. Given an assignment, the default parameter values can be replaced and the modified source file can be written out to disk. The library containing the mutated parameters is compiled and evaluated as part of OpenTripPlanner by a measure related to its energy consumption.

Variation Points. The first two items in the process are performed manually, as the most reliable way to determine valid values for the parameters is by reading the API documentation. In total, due to dependencies and mutual exclusions, there are 9 parameters which may be modified—6 integer values and 3 binary or ternary values.

Mutating the Source Code. For each variation point, the range of potential substitution values were selected to be appropriate. For example the *initialCapacity* and *maximumSize* parameters were assigned the range [0, 100000], whereas *keyStrength* was assigned the range [0, 1] (mapping to {strong, weak}) and *valueStrength* the range [0, 2] (mapping to {strong, weak, soft}). As a template version of the `CacheBuilder` class has been created, the substitution values can be directly inserted and written to disk—using an enumeration where appropriate for the selection of reference strengths.

Measuring Energy Consumption. We have used a new tool, Opacitor, to measure the energy consumption during the evolutionary process. Opacitor is designed to make measurements deterministic, meaning that multiple runs are no longer required and very similar algorithms can be accurately compared. Using a modified version of OpenJDK, Opacitor counts the number of times each Java opcode was executed. Combined with a model of the energy costs of each Java opcode, created by Hao et al. [5], the tool is then able to calculate the number of Joules used.

In order for a Java program running in a standard environment to be deterministic, various features of the Java Virtual Machine (JVM) must be disabled—namely Just-In-Time compilation (JIT) and Garbage Collection (GC). It is not appropriate to explicitly disable GC, and so instead the initial memory allocated to the JVM is increased to the point that GC does not occur. These features are re-enabled after the evolution has completed, to allow a comparison to occur under realistic conditions.

An important benefit of a model- and trace-based tool such as Opacitor, when compared to more common approaches such as timing or physical energy measurement, is that is can be run concurrently with other programs without any detrimental effects. This means that fitness evaluations can be executed in parallel on a multi-core system.

Previous work [16] used Jalen [14] to successfully calculate the energy required by Quicksort. Jalen uses time and CPU utilisation as a proxy to calculate energy consumption, and so we have used this tool to corroborate the results generated by Opacitor during the final comparison.

4 Experiments

We used a metaheuristic search to specialise Guava's Cache library to suit OpenTripPlanner with the property of reduced energy consumption. We decided to

use a GA [8] to search the space of solutions, since this has been shown to be an effective approach for a number of assignment problems [3]. The solution representation used is a vector of integers $r \in \mathbb{Z}^k$. The representation is constrained such that each integer falls within its respective bounds, for example a boolean parameter $r_{bool} \in [0, 1]$ or a size parameter may be $r_{size} \in [1, 100000]$ (limited to ensure the memory requirement does not exceed that available to the experimental machine).

The GA was configured with a population of 100, running for 100 generations. New populations were generated using an elitism rate of 5 %, single-point crossover with a rate of 75 %, and one-point mutation with a rate of 25 % with candidates selected using tournament selection with arity 2. These parameters were selected, after preliminary investigations, in order to provide a sufficient opportunity for the evolution to proceed without requiring an excessive number of fitness evaluations as each evaluation takes over 2 min on a 3.25 GHz CPU. During experimentation we ran the GA five times, with a different seed to the random number generator, in order to test the robustness of the evolution.

5 Results

Statistics were generated using the Astraiea statistical testing framework [13] which performs tests in accordance with the guidelines of Arcuri and Briand [1], namely the Wilcoxon/Mann Whitney U Statistical Tests and Vargha-Delaney Effect size tests. The results were obtained with 100 samples in each dataset. The result of each of the runs of the GA was a similar set of parameter settings, differing only in the exact integers used for the size parameters, and so the first results generated were used for the final comparison between the original library and our specialised version.

The results of the final comparison between Cache versions are shown in Table 1. During the evolution, with JIT disabled and GC avoided, the best set of parameters found used 13596.94 J. This compares favourably with 13857.65 J required by the original version, although it may not initially appear to be a particularly sizeable reduction. This is due to the overhead incurred by OpenTripPlanner when initially loading mapping and transit data—this is unaffected

Table 1. Energy (J) required to exercise Guava's Cache library, as used by OpenTripPlanner (mean of 100 runs, and standard deviation σ), as well as the p-values (p) and effect size measures (e) comparing our result to the original.

Measurement technique	GA	Original			OpenTripPlanner
	J	J	p	e	Overhead (J)
Opacitor	13596.94	13857.65	–	–	10027.24
Opacitor with JIT and GC	807.69 σ1.57	888.82 σ1.75	<.001	1.00	652.98 σ1.27
Jalen	783.79 σ2.18	815.50 σ1.84	<.001	1.00	662.45 σ1.48

by the Guava Cache, and so these measurements are also included in Table 1 to allow a more useful comparison. Subtracting this overhead shows the improvement more representatively—the evolved version used 3569.70 J and the original version used 3830.41 J. As the measurements in this case are not subject to noise, only one measurement for each of the versions is generated and thus no statistical tests are required.

More interestingly, significant results were also found when noise was reintroduced—enabling JIT and using the JVM's default memory allocation settings (allowing for GC when necessary). In this case the GA's solution required 807.69 J, compared with 888.82 J for the original (or 154.71 J and 235.84 J respectively after subtracting the overhead). As the energy measurement now contains noise, due to the non-determinism of JIT and GC, the p-values and effect size measures are calculated. Vargha and Delaney suggest that a large difference between data sets is indicated by a value of 0.71, and so the results show that a significantly improved version of Guava's Cache has been found by the GA.

To help corroborate these figures, and support the assertion that Opacitor provides realistic results, we used Jalen (with JIT and GC) to provide additional measurements. The results generated by Jalen can be seen to support the results provided by Opacitor.

6 Conclusion

We have demonstrated a method of optimising parameters to reduce energy consumption using Genetic Algorithms, applied to Google Guava's `CacheBuilder` class as used by OpenTripPlanner and using a new tool, Opacitor, to evaluate the energy consumption.

Our results showed that specialising libraries to software packages can provide significant improvements, with the best solution in this case providing a saving of approximately 9 %. The results generated by Opacitor were corroborated using Jalen, which uses time and CPU utilisation as a proxy for energy consumption. As such, it is reasonable to claim that specialising the library has improved both the energy consumption and the execution time of the software using it. Modifying the Cache's default parameters also has an effect on the memory consumption, and so future work should investigate the trade-off between energy/time and memory.

Acknowledgement. Work funded by UK EPSRC grant EP/J017515/1. Data available at https://github.com/nburles/burles2015specialising.

References

1. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab. **24**(3), 219–250 (2012)
2. Bruce, B.R., Petke, J., Harman, M.: Reducing energy consumption using genetic improvement. In: GECCO (2015, to aappear)
3. Chu, P.C., Beasley, J.E.: A genetic algorithm for the generalised assignment problem. Comput. Oper. Res. **24**(1), 17–23 (1997)

4. Gagné, C., Parizeau, M.: Genericity in evolutionary computation software tools: principles and case-study. Int. J. Artif. Intell. Tools **15**(02), 173–194 (2006)
5. Hao, S., Li, D., Halfond, W.G., Govindan, R.: Estimating mobile application energy consumption using program analysis. In: 35th International Conference on Software Engineering, pp. 92–101. IEEE (2013)
6. Heggestuen, J.: Business insider: one in every 5 people in the world own a smartphone, one in every 17 own a tablet (2013). http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10. Accessed 3 May, 2015
7. Hoffmann, H., Sidiroglou, S., Carbin, M., Misailovic, S., Agarwal, A., Rinard, M.: Dynamic knobs for responsive power-aware computing. ACM SIGPLAN Not. **46**, 199–212 (2011). ACM
8. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge (1992)
9. Katagiri, T., Kise, K., Honda, H., Yuba, T.: FIBER: a generalized framework for auto-tuning software. In: Veidenbaum, A., Joe, K., Amano, H., Aiso, H. (eds.) ISHPC 2003. LNCS, vol. 2858, pp. 146–159. Springer, Heidelberg (2003)
10. Koomey, J.: Growth in data center electricity use from 2005 to 2010, August 2011
11. Luke, S., Panait, L., Balan, G., et al.: A java-based evolutionary computation research system, March 2004. http://cs.gmu.edu/~eclab/projects/ecj
12. Manotas, I., Pollock, L., Clause, J.: SEEDS: a software engineer's energy-optimization decision support framework. In: Proceedings of the 36th International Conference on Software Engineering, pp. 503–514. ACM Press, New York (2014)
13. Neumann, G., Swan, J., Harman, M., Clark, J.A.: The executable experimental template pattern for the systematic comparison of metaheuristics. In: Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation Companion, pp. 1427–1430. ACM (2014)
14. Noureddine, A., Bourdon, A., Rouvoy, R., Seinturier, L.: Runtime monitoring of software energy hotspots. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 160–169. IEEE (2012)
15. Schulte, E., Dorn, J., Harding, S., Forrest, S., Weimer, W.: Post-compiler software optimization for reducing energy. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 639–652. ACM (2014)
16. Swan, J., Burles, N.: Templar-a framework for template-method hyper-heuristics. In: Machado, P., Heywood, M.I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K. (eds.) EuroGP 2015, LNCS, vol. 9025, pp. 205–216. Springer, Heidelberg (2015)
17. Tăpuş, C., Chung, I.H., Hollingsworth, J.K., et al.: Active harmony: towards automated performance tuning. In: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, pp. 1–11. IEEE Computer Society Press (2002)
18. Vuduc, R.W., Demmel, J.W., Bilmes, J.: Statistical models for automatic performance tuning. In: Alexandrov, V.N., Dongarra, J., Juliano, B.A., Renner, R.S., Tan, C.J.K. (eds.) ICCS 2001. LNCS, vol. 2073, pp. 117–126. Springer, Heidelberg (2001)
19. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, pp. 1–27. IEEE Computer Society (1998)
20. White, D.R.: Software review: the ECJ toolkit. Genet. Program Evolvable Mach. **13**(1), 65–67 (2012)
21. Wu, F., Weimser, W.: Deep parameter optimisation. In: GECCO (2015, to appear)