Specialising Guava's Cache to Reduce Energy Consumption

Nathan Burles[1], Edward Bowles[1],
Bobby R. Bruce[2], Komsan Srivisut[1]

[1] Department of Computer Science, University of York
[2] CREST Centre, University College London

UNIVERSITY of York

Parameterised Software

Software is often released with configurable parameters, as it is rare for a system to be optimal for all situations.

Parameters allow a single, general version to be released—and then tailored to its environment.

Unfortunately, *tuning parameters can be hard*—often needing in-depth knowledge of the software as well as the deployment domain. . .

Parameter Tuning

Parameter tuning is therefore an ideal target for automation, and research has primarily focused on:

- Execution time, e.g. [6, 10, 11, 12].
- Memory consumption, e.g. [10].
- Occasionally functional attributes such as output precision, e.g. [5].

More recently, research concerning energy efficiency has come to the fore, e.g. [2, 5, 7, 9].

OpenTripPlanner

We wish to apply SBSE to tune and specialise parameters of Guava's `CacheBuilder` class to reduce the energy consumption of OpenTripPlanner [http://www.opentripplanner.org/].

> *OpenTripPlanner (OTP) is an open source platform for multi-modal and multi-agency journey planning.*

Essentially OTP reads mapping data (OpenStreetMap) and "General Transit Feed Specification" (GTFS) data from public transport systems, and allows users to find routes.

CacheBuilder

Guava's `CacheBuilder` class is used to create instances of Guava's `Cache`, which is:

> *A semi-persistent mapping from keys to values. Cache entries are manually added using get(Object, Callable) or put(Object, Object), and are stored in the cache until either evicted or manually invalidated.*

OTP uses a `Cache` to store previously accessed "tiles"—the map requires processing to determine routes/stops/etc; once an area has been processed it is stored for future re-use.

Implementation Outline

In outline, the process is as follows:

- Find declarations within the `CacheBuilder` class, and identify valid values for each of these, e.g. ints: `int initialCapacity = ?`, enums: `Strength keyStrength = Strength.{strong, weak, soft}`.
- Generate a template version of the `CacheBuilder` class to allow the variation points to be easily replaced by the respective element in a candidate solution.
- Given an assignment, the parameter values are replaced and the modified source file written to disk. The library is compiled and evaluated as part of OTP in terms of its energy consumption.

Variation Points

For convenience of implementation, the search for parameters and template-creation was *performed manually* using the API documentation to determine valid substitutions.

Due to dependencies and mutual exclusions, there are **9 parameters** to modify: 6 integers, and 3 binary/ternary values.

The range of substitution values differed across variation points, e.g. *initialCapacity*: [0, 100000], *concurrencyLevel*: [1, 32], or *keyStrength*: [0, 1] (mapping to {strong, weak}).

Measuring Energy Consumption: Existing methods

Hardware tools exist, such as the data provided by an Uninterruptible Power Supply or electronic watt-meter. These are useful for a *coarse-grained overview*. . . their precision and accuracy is too low for comparing very similar algorithms.

Software alternatives such as:

- JALEN [8]—targets Java; estimates power consumption as a function of execution time, CPU utilisation, and clock frequency (low precision as reliant on timing: experiments must be repeated multiple times within a measurement).
- Wattch [1]—cycle-level simulator, provides ability to distinguish between very similar programs. . . requires a parameterised model of CPU, and doesn't support Java

Opacitor (1)

Opacitor traces the execution of Java code, using a modified version of OpenJDK. This JVM generates a histogram counting the number of times each Java opcode was executed.

Uses a model of energy costs of each Java opcode created by Hao et al. [4] in their eLens work to calculate the energy consumption.

It provides ability to distinguish programs down to *single instruction*. . . but accuracy depends on the model used!

OPACITOR (2)

Most important in this work is comparative accuracy, e.g. $a < b$ is correct, rather than $a = 3.78543\text{J}\ldots$ this depends on the variability of energy consumption by opcodes, particularly those dependent on operands.

A further complication is the significant **variability between different runs** of the exact same Java program: both *Garbage Collection* and *Just-in-Time compilation* are non-deterministic, which greatly affects execution time and power consumption.

OPACITOR (3)

In OPACITOR, therefore, during evolution GC and JIT are both disabled. This allows runs to be repeatable.

In final testing, all features are enabled to ensure final results are valid on a standard JVM.

This leads to an important benefit of OPACITOR, compared to timing or wall-power measurements—its determinism means that it can be executed in parallel or concurrently with other software.

Experimentation

We used a GA to search the space of solutions, running this 5 times with different seeds. The results from different runs were similar, although not exactly the same due to the minimal difference between e.g. `initialSize = 50` vs. `51`.

An evaluation consisted of starting the OTP server (loading TriMet data), requesting 25 randomly-selected routes from 100 available, and stopping the server. Final testing used 25 routes not used during training.

Each fitness evaluation took **over 2 minutes**, so the ability to parallelise these was important.

Results (1)

Measurement technique	GA	Original			(J) OTP Overhead
	J	J	p	e	
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

Table: Energy used by test program as measured by OPACITOR, OPACITOR with JIT and GC enabled, and JALEN. Each measurement shows the mean and standard deviation over 100 runs of OTP using two versions of `CacheBuilder`—the GA results vs. original.

Results (2)

Measurement technique	GA	Original		
	J	J	p	e
OPACITOR	3569.70	3830.41	–	–
OPACITOR with JIT and GC	154.71 σ1.57	235.84 σ1.75	<.001	1.00
JALEN	121.34 σ2.18	153.05 σ1.84	<.001	1.00

Table: Energy used by test program as measured by OPACITOR, OPACITOR with JIT and GC enabled, and JALEN. Each measurement shows the mean and standard deviation over 100 runs of OTP using two versions of `CacheBuilder`—the GA results vs. original.

Conclusions (1)

We have demonstrated that optimising parameters using GAs can reduce energy consumption—in this case by optimising a library used by OpenTripPlanner.

We measured the energy consumption using OPACITOR, a tool which traces the execution of a program and calculates the total cost using a model of individual opcode costs.

Conclusions (2)

Unsurprisingly, the parameters which had the largest effect were `initialCapacity` and `maximumSize`—the former affecting the initial memory allocation and in particular subsequent re-hashing, and the latter changing the point at which old tiles are removed from the `Cache`.

The default value for `initialCapacity` was **16**, and we found improvement using higher values—the best **between 125 and 150**.

The default for `maximumSize` was set by OTP as **50**, and so with this value (or when the GA chose a value too low) this adds extra computation. Again, anything greater than **125–150** provided the best results.

Conclusions (3)

The best solution found by the GA was able to perform the route-finding using approximately 9% less energy (or c. 34% when the OTP overhead is subtracted).

JALEN corroborated these results, although not as closely as in previous work [3]. This is likely due to the *difficulty in terminating OTP at the correct time*, i.e. after all computation, but without wasted time. JALEN is also unaware of the *difference in energy costs of different instructions*. . . even if they take the exact same time (e.g. same number of clock cycles), the energy consumption of instructions will differ.

References I

- David Brooks, Vivek Tiwari, and Margaret Martonosi. "Wattch: a framework for architectural-level power analysis and optimizations". In: *27th Annu. Int. Symp. on Comput. Archit., ISCA '00*. ACM, 2000, pp. 83–94.
- Bobby R. Bruce, Justyna Petke, and Mark Harman. "Reducing Energy Consumption Using Genetic Improvement". In: *GECCO*. To aappear. 2015.
- Nathan Burles et al. "Object-Oriented Genetic Improvement for Improved Energy Consumption in Google Guava". In: *Search-Based Software Engineering*. Springer, 2015, pp. 255–261.

References II

- Shuai Hao et al. “Estimating mobile application energy consumption using program analysis”. In: *35th International Conference on Software Engineering*. IEEE. 2013, pp. 92–101.
- Henry Hoffmann et al. “Dynamic knobs for responsive power-aware computing”. In: *ACM SIGPLAN Notices*. Vol. 46. 3. ACM. 2011, pp. 199–212.
- Takahiro Katagiri et al. “FIBER: A generalized framework for auto-tuning software”. In: *High Performance Computing*. Springer. 2003, pp. 146–159.

References III

Irene Manotas, Lori Pollock, and James Clause. "SEEDS: a software engineer's energy-optimization decision support framework". In: *Proceedings of the 36th International Conference on Software Engineering*. NY, USA: ACM Press, 2014, pp. 503–514. ISBN: 9781450327565. DOI: 10.1145/2568225.2568297.

Adel Noureddine et al. "Runtime monitoring of software energy hotspots". In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2012, pp. 160–169.

References IV

Eric Schulte et al. "Post-compiler software optimization for reducing energy". In: *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 639–652. ISBN: 9781450323055. DOI: 10.1145/2541940.2541980.

Cristian Ţăpuş, I-Hsin Chung, Jeffrey K Hollingsworth, et al. "Active harmony: Towards automated performance tuning". In: *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press. 2002, pp. 1–11.

References V

Richard Vuduc, James W Demmel, and Jeff Bilmes. "Statistical models for automatic performance tuning". In: *Computational Science - ICCS 2001*. Springer, 2001, pp. 117–126.

R Clint Whaley and Jack J Dongarra. "Automatically tuned linear algebra software". In: *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*. IEEE Computer Society. 1998, pp. 1–27.