Project Work Book: Programming Java for Longer Battery Life

Supervisor: Dr Alexander Brownlee

Anna Frances Rasburn - 2411187

2018

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Brief Summary** | **Detailed Summary** | **To Do** |
| Thursday 19th April | Initial Meeting with Dr Brownlee | * Discussion of the project * We talked about the different ways we could do this project * Defined project definition, requirements, structure and timeline | * Read Resources about the topic * Start the preliminary report |
| Friday 18th May | Dissertation Proposal Submitted | * Dr Brownlee signed the document * Submitted to reception | * Do some reading and research over the summer |
| Sunday 15th July | Reading: Search-Based Energy Optimization of Some Ubiquitous Algorithms | * One of the first papers about applying search-based software engineering to minimize the energy consumption of JVM running programs. * Opacitor – a tool for measuring the energy consumption of JVM programs using a bytecode level model of energy cost. * Advantages of Opacitor:  1. Deterministic – ‘[something] in which no randomness is involved in the development of future states of the [thing]’. A determistic model should produce the same output from a given input or start. <https://en.wikipedia.org/wiki/Deterministic_system> 2. Unaffected by the rest of the computational environment 3. Can detect small changes in execution.  * Energy reduction up to 70% * 3 considerations for automatically reducing computational energy:  1. Tuning software to particular distributions of data 2. Trading off energy use against functional properties 3. Handling internal dependencies.  * Criticism: only experimented on a Samsung galaxy s3 * Criticism: only looks at CPU and does not consider display, wifi and GPS * ‘Dependent of the total number of CPU cycles, there can be a large difference in the energy consumed by different opcodes’ * Opacitor ‘can detect small changes in execution profile, down to opcode level’ * Three SBSE approaches for optimisation  1. Genetic Programming – to obtain better pivot functions for quicksort 2. Hyperparameter – search over the efficiency tradeoff of MLPs 3. Genetic Algorithms – OOP for guava and Apache commons collection classes  * Program behaviour: ‘Design by Contract’ – a software correctness methodology. Uses pre and post conditions to a document the change in state caused by a piece of a program. * Software development now considers small scale computing * CPU is often used as a proxy for energy use. These previous approaches are inaccurate as it omits CPU idle states. * A lot of research have measured power consumption directly from hardware * SBSE algorithm often repeats measurements, extends run-time and reduces confidence in the results. * Opacitor comparison: LLVM toolchain – measures energy consumption due to each bytecode at runtime. * Code perforation – finding parts of the code that can be skipped, such as loop iterations. * Data structures like lists, sets and maps tend to be less CPU heavy * Fitness function is the nergy consumption measured by Opacitor. Fitness is determined deterministically * All non-deterministic features of the JVM are disabled like JIT and GC * Templar – a software framework for customising algorithms via the generative technique of the template method. * Multi-Layer Perceptron – a modification of the standard linear perceptron with ‘hidden’ layers of neurons with nonlinear activation functions. It can distinguish data that are not linearly separable and are often applied to classification tasks like image recognition. * Energy is reduced to around 20% of the worst-case solutions for all data sets. * SBSE allows the developer to easily explore the trade-off between functional and non-functional(accuracy and energy) * Open Closed Principle – both framework implementations and their client programmers can be assured that their code can interoperate without unexpected behaviour * Design by Contract is the explicit codification of the LSP via preconditions, postconditions and invariants. * Benefits to catch any situation where dependencies exist between variation points. * Criticism: programs failing the unit test were rejected by the search as invalid. * An SBSE approach is able to accommodate these interactions where they exist without the developer needing to specifiy where they are, achieving better performance without additional developer effort. * Criticism: No consideration for different versions of JVM |  |
| Saturday 16th June | Reading: Reducing Energy Consumption using Genetic Improvement | * Big Idea:GI seeks to imptove software’s non-functional properties by treating program as if it was genetic material * GI can successfully be used to reduce energy consumption by 25%. * The total ICT infrastructure generated 1.9% of global carbon dioxide consumption in 2010 * There is a big disconnect between a developer’s source code and the energy it need when it is complied. * Petke et al – made MiniSAT which is a popular Boolean satisfiability solver can be optimised for execution time so it is 17% faster than CIT * Fitness of a candidate solution is calculated my measuring the total energy consumed across all the tests in the training set. * A fitness greater than 1 indicates a solution that consumes less energy. * A fitness more than 1 indicates a solution that consumes more energy. * Crossover is done by selected one parent based on fitness and the other one is random. * Crossover continues till after selection, the population has doubled. * Intel power gadget tracks the OS an hardware and estimates the energy consumption of second and higher intel core processer. Doesn’t include energy consumed in main memory or in I/O tasks. * Combinational Interaction Testing (CIT) – Black box sampling technique to test highly configurable software. * Ensemble computation – the study of an NP complete variant of the Boolean circuit problem where one must find a smaller circuit that satisfies requirements. * GI is yet to applied to bigger hardware and applications. |  |
| April | Reading: Specialising Guava’s Cache to Reduce Energy Consumption | * Big Idea: the use of Genetic Algorithms on parameter tuning with the Google Guava’s cache Library. * The creation of the tool Opacitor which is a tool that measures the energy consumed of a program. * Jalen – it uses time and CPU utilisation to calculate energy consumption. * Due to the numerous factors of parameter tuning regarding optimization, they used a SBSE to tune and specialise parameters because of the limited battery life of mobiles. * ECJ Toolkit is the most popular toolkit for evolutionary computation so it may be very useful in my project. * Previous work has included Hoffmann et al ‘s ‘PowerDial’ which is ‘a system for dynamically modifying trade-offs between accuracy in computation and use of system resources during load peaks’ * PowerDial reduces the amount of computing infrastructure instead of really focusing on energy consumption. * Wu et al used Genetic Algorithm’s for the tuning od deep and shallow parameters using dlmalloc memory allocator for execution time and memory consumption. * Mutation of source code – each variation point has a range of potential substitution values were selected till appropriate. In the template version of CacheBuilder the substitution values can be directly inserted. * Measuring Energy Consumption – Opacitors time to shine. It is designed to stop multiple runs and algorithms can be compared. * Using OpenJDK, Opacitor counts the number of times each java opcode is executed. Opacitor calculates the number of joules used. * Features of the JVM have to be disabled for example JIT and Garbage Collection. The initial memory allocated to the JVM is increased to the point so the GC is not used. Once the evolution is done then then these features are re-enabled. * Opacitor can run concurrency with other programs so the fitness can be calculated in parallel on a multi-core system. * When JIT was enabled using the JVM’s default memory allocation, the results found that the noise was reduced. * Future work is to investigate the trade-off between energy/time and memory. |  |
| Key Terms:   * Hyperparameter Optimization – the problem of choosing a set of optimal hyperparameters for a learning algorithm. * Hyperparameter – a parameter whose values are set before the learning process begins * Automated Parameter Tuning – automatic process of tweaking software parameters until optimal configurations. * Variation points – the declaration of default values for parameters. | |
| April | Reading: Object-oriented Genetic Improvement for Improved Energy Consumption in Google Guava | * Big Idea: using a metaheuristic search to find a semantically equivalent version of the immutable map that uses less energy. * Comparing the metaheuristic search to an independent exhaustive search at each variation point so the metaheuristic has a better performance. * Previous work – Sahin et al – measuring the effect of 6 refactoring techniques on Java programs and finding out that the energy usage are highly end-application dependent. * Semantics preserving are done through behavioural equivalence which is important to object-orientated principles. * Mantos et al’s ‘SEEDS framework’ which alternates subtypes container classes are substituted into bytecode so it minimises power consumption. * DRAW IMPLEMEMTATION DIAGRAM here * Use a map instead of a collection as they have a more efficient Big O (o(1)). * Mutating source code – each variation point we need to determine the interface of the created object. * Opacitos is unaffected by anything else executing so it can be executed simultaneously with other programs without difficulty. Object-Oriented Genetic Improvement – a technique by which non-functional properties so time/energy consumption may be optimised. |  |
| Key Terms:   * Metaheuristic – a high-level problem independent algorithmic framework that provides a set of guidelines to develop evolutionary algorithms. * Genetic Program – technique whereby computer program are encoded as a set of genes that are then modified using an evolutionary algorithm. * Grammatical Evolution – to find an executable program that will achieve a good fitness value for the given objective function. * Liskov Substitution Principle – if something expects a parent class but gets a child, the child should conform to what is expected of the parent. | |
| Wednesday 12th September | Initial welcoming meeting back with Supervisor | * High level introduction to generic algorithms and deep tuning * Talk about how to begin project and different ways we can go * IDEA: Does the version of java affect energy/execution time * We talked that we want to get the fundamentals right first and if we do go to Android Application ‘that will be the icing of the cake’ * Agreed that would take the initial route of * Switch circuit ? – why if(a+b) is different to if(b+a). this is because if a is false then we won’t even check if b is false as we just give up. If we swap and b is true then we still see a. It is useful in using methods. * Looked at Gin and Jalen * Looked whether to go creating my own GA or going via gin. * I’m gpnna make my own GA and unit tests to test with. As gin is open source so it want to change approach I can do this. * REMEMBER TO EDIT CONFIGURATIONS TO RUN THE APPLICATION * LOOK AT THE GIN IF STATEMENTS FOR ‘SUCCESS’, FAILURE AS STARTING POINT * Gin has never been used with Mac before so this may be interesting | * Try writing a piece of code that emulates the test framework of the gin code. * Research and read about Genetic Algorithms * Due Date: 19th September |
| 13th September | Reading: Artificial Intelligence – A Guide to Intelligent Systems  Page.222-232 | * Genetic algorithm – ‘a class of stochastic search algorithms based on biological evolution’ * Generation – The iterative process run of a GA * Mitchell – typical number of a simple GA can range from 50 to 500. * Run – the entire set of a generation, at the end of a run we find out ot more highly fit chromosome. * DO DIAGRAM OF GENETIC ALGORITHM HERE * The fitness of a population may remain stable for a number of generations before a superior chromosome appears. |  |
| Friday 14th September | Got GinFork code to work | * Met up with Sandy to ask for help with running the ginfork application * Problem seemed to be with the run configurations. Initially I used these configurations: gin.LocalSearch examples/simple/Triangle.java -cp=examples/simple/-className=Triangle. The run configurations I use now: examples/simple/Triangle.java -className=Triangle -cp=examples/simple/ -steps=10 |  |
| Saturday 15th and Sunday 16th | Implemented a basic genetic algorithm | * I created my own genetic algorithm simulation with 3 basic classes of individual, population and GA. * It runs okay, it generates a new population and calculates the fitness and the fitness improves as the generation count increases. * DO A UML DIAGRAM OF THIS |  |
| Friday 14th – Wednesday 19th September | Play around with GinFork | * Played with the. Firstly began by trying to get it to break and see how it runs. * Added breakpoints and ran for large number of steps to see differences. |  |
| Wednesday 19th September | Second meeting with Supervisor | * Presented to Supervisor my basic genetic algorithm. We discussed and improved it. * Asked questions I had about Gin code * Discussed how my GA and gin code is different as my GA has a population and gin doesn’t technically. * Gin has a ‘hill walker’ structure. * Discussed the interim report and ethics checklist. | * Ethics checklist * Interim report beginning * Play with Gin code * Make a GIN like edit myself * Get the jcodec code to work |
| Friday 21st September | Ethics Checklist submitted | * Marked no to everything as it is a research project therefore has no end users |  |
|  | Reading: An Introduction to Genetic Algorithms  Page 1 - | * Since 1980s Biologically motivated computing has had a resurgence with Neural networks to Machine Learning to Genetic Algorithms. |  |
|  | Reading: Essential of Metaheuristics |  |  |
| Saturday 22nd September | Beginning of Interim Report | * Started writing the introduction and state of the art sections so I can show Sandy. * Added references too. |  |
| Sunday 23rd September | Started implementing my own edit in a similar vain to GIN |  |  |
| Wednesday 26th September | Third Meeting with Supervisor | * Discussed how to run jcodec with sandy as I realised I had been doing it wrong. Also developed a better understanding of how jcodec works. * Discussed the interm report and which examples we can talk about in state of the art * Discussed the ways we can go with the project, whether to use opacitor at the end or introduce it from the beginning if the project. * Opacitor does not have binary code that is implemented to run on a MAC OS so that may be a challenge as the project progresses but will work out a solution. * Discussed the test runner areas of GIN code. This is the really important part of GIN. * Going to make my own code using test runner which may be quite challenging but gonna try and get as much as I can done in a week. * Need to look at the article ‘Genetic Improvement of Software: A comprehensive survey’ <- look at types of changes and transformations. | * Run gin targeting jcodec * Extracting testrunner * Continues with interim report |
| Thursday 27th September |  | * Copied over the test runner classes as a starting point to make my own wrapper * Continued with interim report |  |
| Friday 28th September |  | * Continued with Interim report * Made an initial project with classes from the original gin code |  |
| Saturday 29th September |  | * Tried implementing a project similar to gin. |  |
| Sunday 30th September |  | * Realised the code I was writing over the weekend was practically a copy of gin so made a new project structure and started implementing a much more simple version. Not yet completed. * Changes I wish to make. I think the gin code is good but at the moment it ignore tests that aren’t unit so I wish to make a better ground that is extensible and can be changed so more stuff can be added later on. * Also wish to simplify the structure by trying to see the edit as an individual and a patch as a population so in a similar vain to the genetic algorithm I made in the beginning. * Also changed to use maven and spring framework as wanted an excuse to practice these more. * Also realised the bad structure of my project and git repository so deleted it and started again with a better structure |  |