EE 790 | Fall 2017 | DRZ



makes it clear that we will need different methods for different types of software, as there is no 'one size fits all' here.

As seen in Figure 1, the author has provided a breakdown of some of the important methods for measuring energy efficiency in software. We will discuss each of them here.

2.3.1. Black Box Measurement

Black box measurement effectively treats the entire software system as one thing, ignoring all of the components that actually make up the system. With this measurement method, we are only interested in the software as a whole, which means that should any problems be found, we will need a more sophisticated method to find where they are occurring within the system.

One of the methods falling under this category is known as benchmarking. This involves simply running the software against a standard set of tests. While the tests are running, energy consumption is measured. There are several different benchmarks that can be used for measuring energy efficiency, such as EnergyBench.

The other method falling under this category is individual measurement. This process involves coming up with several specific use cases, and measuring the performance of different configurations or types of software by running through the scenarios. These performance measurements can then be compared.

2.3.2. White Box Measurement

Unlike black box measurement, the white box methods look at software for what it really is: a complex system of different modules interacting together. These methods allow you to pinpoint where deficiencies in energy efficiency lie. However, they require a greater understanding of the system being measured, as the tester must interact with the code itself.

The white box method mentioned by the author is called source code instrumentation. This involves inserting instructions into the code itself meant to monitor different aspects of the system. It is much more difficult to implement than the black box methods, and because of this, there are not many examples that can be given.

2.4. Energy Efficient Programming Methodologies and Common Problems

This section is basically the meat of the paper, and discusses a lot of different methods to assist in developing energy efficient software. The methods are broken up into three categories: application software, system software, and general.

2.4.1. Application Software Efficiency

In this section the author lays out many different methods for improving the efficiency of

counter. In some situations it can be beneficial to get rid of the loop entirely, and simply repeat the instructions that would have been executed individually to get rid of this overhead. Although this may look ugly, it can help.

A third method for reducing energy consumption is to utilize multi-threaded approaches where you can. For a task that does not require all of its steps to be done in sequence, many times parts of the task can be split and done simultaneously. Although this can be difficult to manage, it obviously has the benefit of getting the work done faster, which meets the goal of increasing idle time. Figure 2 shows the benefits of this approach. As you can see, increasing the number of threads causes an increase in the power consumption at the start, but since the task is completed much quicker, the power drops off to idle much faster.

Another consideration to make is whether or not there is already an existing library that can handle your task. Many software developers prefer to write their own code for everything, but often enough, there is already an excellent solution that has been proven to be energy efficient, saving you the effort.

Finally, choosing the right programming language for your task can make a huge difference as well. All languages handle things differently, meaning that energy consumption can vary drastically depending on which language a system is implemented in. This may require expert knowledge of the ins and outs of different languages, but it has been shown to have a dramatic effect on energy consumption. Figure 3 shows the results of an experiment running the same recursive algorithm in different languages. It is clear that in this case, the best language is C++, but this could simply be owed to how C++ handles function calls. In other cases, there will obviously be different results.

2.4.1.2. Data Efficiency

This section deals with methods to improve the efficiency of data movement. The basic idea here is to complete a task with as few memory accesses as possible, and moving data over as short a distance as possible.

The closer data is stored to the processor, the less energy is consumed when accessing that data. This basically means that you should try to store data as close to the processor as posally mBT 12.0 Tf [(pos)ch.2 (y mB) -0.2 (i) 0.2(c) 0.2 (a) 0.2 (l) 0.20.2 (he)055 cm Bn.2 (C+

EE 790 | Fall 2017

There is currently a model that can be referred to for energy efficient development, known as the GREENSOFT model. The GREENSOFT Model is a conceptual reference model for green and sustainable software that includes a product life cycle model for software products, sustainability metrics and criteria for software, software engineering extensions for sustainably sound software design and development, as well as appropriate guidance [1]. Unfortunately, most of the reference material for information on this model is locked behind a paywall. Here the author discusses the problem with energy consumption measurements. Although there are existing ways to measure the energy consumption of software, these methods require that a specialized environment be set up with extra equipment. Many are not willing to invest in that. It is also difficult or impossible to expand them to more complex environments such as data centers, or software that is distributed across multiple servers. Other methods investigating the code itself are also difficult to implement, as they require expert knowledge on the subject under study. So, if we could design software to be efficient from the start, these methods would not be required.

3.3. Sustainability as a Quality Attribute

process described in Figure 6 with the goal of reducing energy consumption. When an energy hotspot was found, they used the tactic of increasing hardware utilization. It was discovered that this hotspot was not multithreaded, and it was possible to do so. Figure 7 shows the results of this adjustment. As can be seen, CPU activity was reduced drastically. Applying this adjustment reduced the task energy consumption of generating the documents by 67.1%.

4. ESUML-EAF

This paper proposes a framework developed by the authors that can be used to create energy efficient design models for software. The framework is called Embedded Software modeling with UML 2.x - Energy Analysis Framework. The reason for this name will become clear at a later point. Again, this method is intended to be used in the design phase of software. It is also currently only intended for use in embedded software. The advantage of using a framework such as this is that it allows developers to fulfill the energy consumption requirements in the early phases of software design rather than later. This reduces the feedback that would have been caused had the requirements not been met later. Feedback is essentially the need to redo insufficient work from a previous phase.

4.1. Introduction

The field of embedded software is growing, and as such, the requirement for low energy is growing as well. Most existing studies in this area have focused on the hardware, but software has an impact too. The complexity and size of embedded software affects the energy consumption of the system that the software is embedded in. Fortunately, there has been some research on the software side as well.

The first energy consumption analysis technique was proposed in 1994. The basic idea is to break down the source code into low level instructions, measure the consumption of each of these instructions individually, and sum them. This led to new techniques being studied, and not all of them are instruction level. There are also source code based techniques, and even model based techniques. The issue with instruction and source code based techniques, however, is that although they are accurate, they require a lot of time to analyze. They also require even more time to go back and redo unsatisfactory results, because at this point the code is already written.



measurement of energy consumption. This is acquired by actually executing or simulating each instruction, and can take days. Source code level techniques utilize higher level languages, such as C, to profile the code. This makes it faster than instruction level techniques, but it still takes a lot of time. This led researchers to search for a faster way to analyze energy consumption, and they came up with the model based techniques.

4.3. Framework Architecture

This section discusses the architecture of the framework that was developed. The overview can be seen in Figure 11. The major components are the ESUML modeler, CFG generator, ESUML energy library, energy realizer, and result viewer.

EE 790 | Fall 2017





an encryption algorithm using AES, and image encoding algorithm utilizing Huffman codes, the algorithm for data retrieval found in cell phones, and the image conversion algorithm found in digital cameras. These algorithms were each modeled using this framework. They were also each implemented by the authors as well, so that source code analysis time could be compared to model based. The results of this experiment are shown in Figures 14 and 15. As shown, the model based technique performs faster for every algorithm. Since we are only analyzing small algorithms rather than entire projects, the analysis time for source code is still relatively small, but the numbers for source code analysis would increase a lot faster than the numbers for model analysis. It is also shown that the elapsed time for model based analysis remains relatively stable. Figure 14 also shows that the deviation in energy consumption estimation between source code and model based techniques never exceeds ten percent. This means that the impact to accuracy is present, but negligible. Clearly this framework is an efficient way to choose the best possible design model for your product.