Energy-Efficient Programming

By Emmanuel Perez

Abstract

We live in a world where every day more and more people are getting mobile computing devices. Smart phones, laptops, and tablets are now part of the everyday life of businessmen, students, and many others, and so, this raises a concern about energy consumption. While generally when it comes to saving energy on a mobile device we think of what is happening with the hardware, as in how it is physically built, we must think lower than that. Along with questions about how the physical build of the devices consumes energy, we must also ask ourselves about how the programming of the functions affects the amount of energy that is consumed. And so, many experts are studying alternative ways to program our devices and applications to make them more energy efficient. This approach can then not only be used for mobile devices, but also for larger places in which servers are stored where simple changes in code can result in huge energy savings. Also, beyond financial gain, these energy savings will help reduce the carbon footprint left behind by everyone that uses a computing device. So, with energy-efficient programming, businesses, consumers, and even the planet, come out as winners.

1. Introduction

Energy consumption is one of the most talked about topics in the world today as it is one that is directly affecting our way of life and has an impact on what happen in the future to our planet. Things such as global warming and some of the recent extreme weather events that we have had has made the experts tell the world that they should watch how much energy each person consumes. This is because the more electrical energy we use, the greater the amount of carbon dioxide is released into the atmosphere (also known as the carbon footprint). Some have followed the advice of the experts and as a result the use of hybrid or electrical cars has increased around the world. Also, many have outfitted their homes to use either solar energy or natural gas to provide energy for them. But the changes to reduce energy consumption should not stop in big-ticket items such as homes and cars. Every day we use other things that consume energy and leave a carbon footprint behind.

Among those such items are computing devices. It’s no secret that every day more and more computing devices, varying from home computers to smart phones, are being put into the hands of consumers. In my household alone, which could be considered low-middle class, there are three computers (one desktop, two laptops), three internet-enabled cell phones, including a Smartphone, and a Play Station 3 videogame console which can do almost as many functions as a regular consumer-grade computer. Also, from personal experience, I can say that those who cannot afford to own such things go to the local library to use computers when necessary. In addition to that, a 2012 study from the Pew Internet and American Life project says that about 53% of the Americans polled said that they have a Smartphone [4]. So, it goes without saying that computing devices have become an integral part of society at large, so much so that it’s been parodied in Hollywood films, like in the pictures below:
And things do not stop at the consumer level. There are other computing devices that must be considered that consume lots of energy, such as servers. Large and even small companies have their own private servers (such as the one pictured below) that must run every hour of every day, to keep things running smoothly. Running these servers can cost companies a lot of money, and then when you factor in the cost of controlling the temperature in the rooms where the servers are stored in order to prevent overheating, the overall cost goes way up.
You see, with all of these devices and data centers out there, we cannot take for granted the amount of energy that they consume. So, what, then, can we do to make sure that we consume energy wisely?
As regular individuals, one might tend to think that the main way to save energy with computing devices is to make sure that the hardware runs adequately or that it meets the current energy standards. And we would indeed be partly right since we have seen that as time has progressed, physical technology has gotten better to a point where batteries in mobile devices last longer even when they are used constantly. Also, overheating is less of a problem with newer, consumer-grade devices thanks to technological advances and so the computer doesn’t have to have its cooling fan speed up to bring the temperature down quite as often. However, all of our focus should not go to the hardware, and we must think of the lower levels of computing in order to find more ways to regulate energy consumption. We must also look at how devices and applications are programmed, and what programmers can do to reduce the amount of energy consumed.

The purpose of this paper is to inform the reader about a variety of techniques that a software programmer can use in their own line of work to help the cause of those who want to reduce the carbon footprint that we leave behind, and at the same time help the majority of people and businesses around the world that rely on computing devices, from tablets to big servers, reduce their electricity bills. To do so, I looked for papers by experts in the field of programming who have decided to focus their work on energy-efficient programming techniques to see if they had any good suggestions on the topic. After some research I found that the following three papers give us a good idea on what to do at the programming level to achieve low-energy consumption:

- **EnerJ: Approximate Data Types for Safe and General Low-Power Computation** by Adrian Sampson, Werner Dietl, Emily Fortuma, Danushen Gnanapragasam, Luis Ceze and Dan Grossman, University of Washington, Department of Computer Science and Engineering.
- **Energy-Efficient Software Guidelines** by Petter Larsson, Intel Software Solutions Group
- **Developing Green Software** by Dr. Bob Seigerwald and Abhishek Agrawal, Software & Services Group, Intel Corporation, Folsom, CA, USA.

My goal with this paper is to summarize the various techniques that these experts have suggested so as to provide the reader with a simple guide to them. This paper will be broken down into an additional three sections (four including the conclusion), one for each paper, where I will write the summaries. If the reader then wants to read the words of the experts themselves, I will provide the web addresses where you can find them and read them at their own leisure.

Note: In addition to these papers, I also studied Energy Efficient Programming by Clay Breshears from www.drdobbs.com. However, I found that its suggestions were similar to some in the other papers and did not go into as much detail. As a result, I will not be summarizing it for the purposes of this paper.

### 2. EnerJ: Approximate Data Types for Safe and General Low-Power Computation

#### 2.1. Introduction

When it comes to techniques to reduce energy consumption at the software level much of the focus has been on program architecture, performance and power trade-offs and resource management. These techniques have been proven to be effective (as we will see in further sections), there are other things that could be experimented in to provide further savings. One of
these things is whether or not it makes a different to have exact results when running a program or to make-do with a close approximation of it. While we human beings like to have everything tidy and perfect, in programming, when finding an exact result leads the machine to make a greater amount of computations to find it. What if, then, we were able to run a program using approximate results instead of exact ones? For some parts, it is possible. For example, an image-rendering program can tolerate small errors in the pixel data it outputs and the errors could be negligible. But of course, in other parts, such as jump tables; a small error could make your program crash. So then, to be able to to take advantage of the occasions when errors will not have a big effect on a program, EnerJ was created.

EnerJ is an extension of Java that has qualifiers that distinguish between approximate and precise data types. This means that data that is annotated with the “approximate qualifier can stored approximately and computations that involve it can be performed approximately. Using approximate data types can result in energy savings as not as much CPU time is used coming up with an exact answer. Therefore, the program can finish its task faster.

Although EnerJ adds qualifiers to the Java programs, it is backwards compatible an can be run as a regular Java program. EnerJ’s effectiveness in power savings will be proven later on.

2.2. A Type System for approximate Computation

In this section, we will look at the specifics of EnerJ’s extensions to Java. Here is a table with these extensions and a brief description of them. They will be explored in detail for the remainder of section 2.2.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Approx, @Precise, @Top endorse(e) @Approximable @Context APPROX</td>
<td>Type annotations; qualify any type in the program. (Default is @Precise.) Cast an approximate value to its precise equivalent. Class annotation: allow a class to have both precise and approximate instances. Type annotation: in approximable class definitions, the precision of the type depends on the precision of the enclosing object. Method naming convention: this implementation of the method may be invoked when the receiver has approximate type.</td>
</tr>
</tbody>
</table>

2.2.1. Type Annotations

In a program, every value is going to either be an approximate or precise type. In EnerJ, to make sure that we are using the right one, the qualifiers @Approx and @Precise are introduced. However, precise types are the default, so only the former qualifier is truly needed when writing a program. With that in mind, we must mention that it is illegal to assign approximate types to precise types. This is show in the following example:

```java
@Approx int a = ...;
int p; //precise by default
p = a; //illegal
```
You see, we assigned a as an approximate type, then we tried to equate p, a precise type as a, so it wouldn’t work because a could be considered a floating point.

In addition to @Approx and @Precise, the qualifier @Top is introduced. This denotes the common supertype of the other types.

**Semantics of Approximation**

As could be expected, precise values have a traditional guarantee of correctness. However, approximate types do not. A more complex system could be developed to provide more guarantees, but as of right now, this simple system is enough for a wide-range of applications.

### 2.2.2. Endorsements

In the previous section I mentioned that it was illegal to assign an approximate number to a precise one because the approximate type would be more like a floating point. If that was absolute, then there really wouldn’t be a reason why EnerJ should exist. However, there is a way to get around that: an endorsement. With the qualifier endorse() you, the programmer, will be able to use any approximate type as if it was a precise one. Now we will take the previous illegal example and make it legal:

```java
@Approx int a = ...;
int p;
p = endorse(a); //now it is legal
```

With the endorsement, we can now be sure that the approximate type will not have any negative examples in any formula that uses it as a precise type.

### 2.2.3. Approximate Operations

These qualifiers are all well and good, but that’s not all that is needed when it comes to computations. Approximate computations are introduced by overloading operators and methods based on type qualifiers. For example, EnerJ provides two signatures for the + operator on integers: one for adding two precise integers that will result in a precise integer, and one for two approximate integers resulting in an approximate integer. Because the latter computes approximately, it may run on low-power hardware

**Bidirectional typing.**

If we were to type the equation a = b + c, where a is approximate, but b and c were precise, would we have to add some annotations to make sure that it will work properly and the result will actually be approximate? Not really. EnerJ implements a simple form of bidirectional type checking, where if the equation results in an approximate type but the arithmetic operators are precise, the result will still be approximate. This makes it easier to write equations that mix both precise and approximate data.
2.2.4. Control Flow

I have already mentioned how it is not ideal to equate a precise type to an approximate type. But this can’t always be helped with implicit flows that happen with control flow. This happens in the following example:

```java
@Apprx int val=...;
boolean flag; //precise
if (val==5) {flag=true} else {flag=false};
```

Here, val==5 is an approximate type, therefore it affects the function of flag since that is a precise type. What can then be done about this? With EnerJ you don’t have to actively do anything about it as the prohibits the use of approximate values in conditions that affect control flow such as if and if and while statements. However, this can be worked around with an endorsement.

2.2.5. Objects

EnerJ’s qualifiers are not just for primitive types as classes also support approximation. This can be achieved with the @Approximable class annotation. Within the approximable class, you can also make precise and approximate versions of the class. However, precise class types are not subtypes of their approximate counterparts.

2.2.5.1. Contextual Data Types

The @Context qualifier will make any integer or instruction be the type of type that defines a class. For example, if you define a class with the @Approximable qualifier, then within the class you can use the @Context qualifier to make sure that the integers you use are also approximable. Then, if you want to make the class precise, you don’t have to go back and change every single integer again since it will be automatically taken care of.

2.2.5.2. Algorithmic Approximations

There are instances where two different implementations of a class can be made, one for when when the receiver has a precise type and another one for approximate types. For this, the latter one must have the name must consist of the suffix _APPROX. Here’s an example (this is used to find the mean of a calculation):

```java
@Approximable class FloatSet{
    @context float[] nums = ...;
    float mean(){
        float total = 0.0f;
        for (int i = 0; i< nums.length; ++i=2)
            total += nums[i];
        return total / nums.length;
    }
    @Approx float mean_APPROX(){
        @Approx float total = 0.0f;
        for (int i=0; i<nums.length; i +=2)
            total +=nums [i];
        return 2*total/nums.length;
    }
}
```
As you can see, on the second part of the program the _APPROX suffix was used. With this, it will be made explicit that approximate types must go there; so then the compiler will know which one to use. However, although EnerJ is backwards compatible with Java, you must make sure that in a regular scenario, both iterations are interchangeable to prevent any errors when running on Java.

2.2.6. Arrays

Arrays can be declared with approximate types in them; however, the array’s length must always be precise to safeguard memory. So, even if one attempts to make the length approximate with an endorsement, EnerJ will not allow it so as to prevent out-of-bounds errors.

2.3. Formal Semantics

In the “EnerJ” paper, this section studies the semantics of EnerJ through certain equations that were beyond the reach of what I am trying to achieve. As a result, I will not cover it here. However, if it does interest you then I suggest you read the paper in full once you are done with this one.

2.4. Execution Model

EnerJ distinguishes abstractly between approximate and precise data. And it does not define the strategies that are applied. As such, in order to fully use EnerJ’s annotations, an approximation-aware execution substrate is needed. This section will explore the hardware model, ISA extensions used for approximation to fully take advantage of EnerJ’s features, and how the extensions enable energy savings.

2.4.1. Approximation-Aware ISA Extensions

In order to leverage approximate storage and approximate operations, EnerJ’s hardware model has divide itself in order to have separate room for approximate and precise data. Approximate data storage is presented in the form of unreliable registers, data caches, and main memory. The registers differentiate from the ones for precise data by address. The regions of physical memory are specifically labeled for approximate data, and when it is taken out of memory, they are put into a region of cache reserved for that type of data. Here is a visual representation of how the hardware model is divided.
2.4.2. Hardware Techniques for Saving Energy

This section is dedicated to exploring strategies for saving energy with approximate storage and data operations.

Voltage Scaling in Logic Circuits

Aggressive voltage scaling can result in energy reductions of over 30% with about an error rate of about 1%. For testing the following techniques, aggressive voltage is assumed for approximate instructions.

Width Reduction in Floating Point Operations

One direct way to approximate mathematical operations is to ignore the part of the mantissa (the fraction part of a logarithm). So, if a floating number operation has an 8-bit mantissa, it will use 78% less energy per operation than a full 24-bit multiplier. This loosely means that it’s better to use 5.453 than 5.453434568473.

DRAM Refresh Rate

In an approximation-aware DRAM system, the refresh rate of lines containing approximate data might be reduced to 1Hz, which would reduce power by about 20%. However, this may lead to data decay.

SRAM supply voltage

The registers and caches are partly made up of Static RAM cells. If the voltage supply to them was to be reduced, then the leakage current of the cells would be lowered. As a result, though, data integrity could be affected. Most of the errors this causes tend to be read upsets and write failures. The former happens when the stored bit is flipped by it is read, the latter when the wrong bit is written. But the probability of these things happening is $10^{-7.4}$ and $10^{-4.94}$, respectively, so they would happen rarely.

2.5. Implementation

As stated by Adrian Sampson, et al “EnerJ is implemented using the Checker Framework infrastructure, which builds on the JSR 308 extensions to Java’s annotation facility”[]. JSR 308 allows annotations on any type in a program. Thus, the EnerJ type checker extends its rules to the whole of Java.

2.5.1. Type Checker

The type qualifiers provided by EnerJ (those listed in Table 1) are JSR 308 type annotations. And as previously stated:

- The default type is precise, so only the @Approx qualifier is necessary to change semantics
- Classes defined as approximated only affect instructions annotated with the @Context qualifier.
2.5.2. Simulator

A system that used EnerJ was evaluated in a compiler that runs EnerJ code as if it was running in the architecture as described in section 2.4. Among the commands that were instated in this EnerJ compiler were method calls, object creation/destruction, arithmetic operations, and memory accesses to collect statistics.

Also implemented was a runtime system. This system is implemented as a Java library and is invoked by the commands. It records memory-footprint and arithmetic-operations statistics. At the same time, it injects transient faults to emulate approximate execution.

2.5.3. Approximation

The simulation implemented the techniques described in section 2.4.2. Here are the results:

<table>
<thead>
<tr>
<th>DRAM refresh: per-second bit flip probability</th>
<th>Mild</th>
<th>Medium</th>
<th>Aggressive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory power saved</td>
<td>17%</td>
<td>22%</td>
<td>24%</td>
</tr>
<tr>
<td>SRAM read upset probability</td>
<td>$10^{-6.7}$</td>
<td>$10^{-7.4}$</td>
<td>$10^{-8}$</td>
</tr>
<tr>
<td>SRAM write failure probability</td>
<td>$10^{-5.59}$</td>
<td>$10^{-4.94}$</td>
<td>$10^{-8}$</td>
</tr>
<tr>
<td>Supply power saved</td>
<td>70%</td>
<td>80%</td>
<td>90%*</td>
</tr>
<tr>
<td>float mantissa bits</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>double mantissa bits</td>
<td>32</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Energy saved per operation</td>
<td>32%</td>
<td>78%</td>
<td>85%*</td>
</tr>
<tr>
<td>Arithmetic timing error probability</td>
<td>$10^{-6}$</td>
<td>$10^{-4}$</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>Energy saved per operation</td>
<td>12%*</td>
<td>22%</td>
<td>30%</td>
</tr>
</tbody>
</table>

The probability number represents the probability that an error will occur.

2.6. Results

Now that we have seen what the power savings are in a simulation, we are now going to take a look at an actual study. In this study, several pre-existing applications were taken and then were retrofitted into EnerJ. This was done so by manually annotating each application. The rule while doing this was that each instruction would be annotated in such a way that never caused the applications to crash, as it was important to show that EnerJ allows you to write approximate programs without having them fail enormously. Table 3 shows what applications were used, and a brief description of their function. In Figure 2, we see what fraction of the code of each program was approximated in terms of DRAM and SRAM storage, as well as integer and floating point operations.
Quality of Service Metrics

For each application, the degradation of output quality was measured for the approximate executions, and then was compared to the precise executions. In order to do this, for each application Quality of Service (QoS) metrics had to be defined. These metrics are found in Table 3.
2.6.1. Energy Savings

In Figure 3 we can see the total power consumed by each application at four different stages, when there is zero approximation, and when there’s mild, medium, and aggressive amounts of it. It also shows us how much energy was saved. For example, if we look at the results for the Raytracer application, which is the one that had the most significant savings, we saw that in the jump from zero to aggressive approximation, it only consumed about 52% of energy compared to when it had no approximations, meaning 48% of energy was saved. Overall, we see savings varying from 9% (SOR in mild configuration) to 48%. In average, the three approximation considerations gave us savings of 19% (mild), 24% (medium), and 26% (aggressive). It is also worth noting that the applications with the higher percentage of floating point operations (seen in column four of table 3), gave us the biggest savings. Therefore, this proves that if we are indeed allowed to run these operations as approximate, rather than specific (again, using, for example, 5.5 instead of 5.4895434), does wind up giving us significant savings.

Figure 4- Error output and the three approximation levels over 20 runs
2.6.2. Quality-of-Service Tradeoff

In Figure 4, we can see the percentage of errors per application at the varying degrees of approximation. As we can see, on mild and medium, in every application there is little to nonexistent error output, and where there is it is nearly negligible. Medium approximation only has a big impact on FFT and SOR, which is understandable since they are pretty much operating systems. Aggressive approximation had the biggest impact on the three kernels, and ZXing (the barcode reader), which, once again, is understandable since their functions tend to rely on precision. Although jMonkeyEngine had about 40% error output in aggressive approximation, this is unacceptable since it is a videogame engine, and videogame player don’t like it when their game gets buggy. So overall, despite the fact that approximation does result in some error, most of the time it is negligible. But it is the programmer’s responsibility to know to what degree approximation must be used.

2.6.3. Annotation Effort

In column seven of Table 3, we can see the number of declarations that were annotated in each application and kernel. We can also see the number of endorsements needed. As you can see, not a lot of effort had to be put into making the annotations. For example, in ZXing, which had about 26,000 lines of code and only 4% of its declarations had to be annotated. But once again, then key is where to put them so that your application will not crash.

2.7. Conclusion

Approximate computing is far from perfect, but as we have just seen it can give us significant power savings in a variety of systems, from mobile phones to servers.

3. Energy-Efficient Software Guidelines

EnerJ is not for everyone as it is restricted to the Java language. However, in this paper, Petter Larsson makes suggestions that can be helpful when dealing with just about every programming language, from C to Java and beyond.

3.1. Computational Efficiency

3.1.1. Loops

When a you, a programmer have to use loops as part of a program, you have the choice of dividing the work the loop has to do into many small loops or just put all of the work within one loop. For example, when adding the contents of two different arrays, you could either have two small, tight, loops, one for each of the arrays to bring the contents into memory, and then maybe even one to do the addition. Or perhaps you could put all of this into a larger while loop. The work and the results would be the same, however, it is recommended that unless absolutely necessary, you try to use the least amount of loops as possible. Doing so would reduce the comparison and testing overhead associated with small loops. Without this overhead, then program will be faster. However, there is a downside to this, as code size and register use may increase.
Also, it is recommended that busy wait looks, also known as spinning loops, be avoided as they take up a lot of CPU time that other instructions may need to use. Therefore, work will not be completed as fast as possible and more energy will be consumed.

3.1.2. Performances Libraries/Extensions

As this paper was written commissioned by Intel, the energy-saving performance libraries mentioned here are specific for their products. These libraries include the Intel Math Kernel Library, which contains, per the paper, “…optimized implementations of common algorithms in areas such as audio, video, imaging, cryptography, speech recognition, and signal processing.” The paper also provides with web addresses to where you can find more information about them, so if you want to find out more about them then I suggest you refer to the source for this summary.

3.1.3. Algorithms

Another seemingly easy way to make energy-efficient programs and applications, and one that should really be a no-brainer for anyone making any sort of workable program is to have great algorithms. By “great” I mean high performance algorithms and data structures that complete tasks faster, therefore allowing the processor to go back to a position of doing little-to-no work as soon as possible, therefore you will save energy. In addition to this, if the task that you have to complete allows your program to use less complex algorithms, then used them, as they will finish their work faster. However, if this less complex algorithm is recursive be careful as these are known to not be energy-efficient as they add more overhead by using more stack than regular algorithms.

Another way for your programs to be more efficient is to add a “hot switch” algorithm, which, for example, when used in a video application, will select a lower-quality video encoder or decoder when a computing device is running on battery. I will cover more on these types of techniques for “smarter” programs later on.

3.1.4. Compiler Optimization

Once again, here they refer to Intel-specific compiles and don’t offer much explanation as to how this saves energy other than the following: “Extended optimizations can be achieved by using application profiling to provide insights such as the most common execution paths.” So, once again, I suggest you refer to the source material for this summary and follow the web address they provide to find out more.

3.1.5. Drivers

Device drivers act as translators between hardware and the applications that use them. We could also save energy with these by looking for alternative implementations of them that are more energy-friendly. For example, if you constantly use a Bluetooth device with your computer, then an easy way to make sure that you are using the least amount of energy possible is to just update to the latest version of the application that controls your device. Of course, not everything is that simple as sometimes updates of these drivers may involve updating a whole operating system or program libraries, but they are another option to making programming more energy-friendly.
3.1.6. Programming Language

Depending on the task at hand, you may have no choice but to use a particular programming language. However, if you have choices then consider using one that causes less frequent wakeups. So, instead of using a high-level run-time language it would be better to stick to something that is lower level, such as C. Or perhaps you could give EnerJ a chance.

3.2. Maximize Idle

This section focus on some things that you can do to make sure your programs run faster so that it can go back to a state where it does little or no activities as the more time the system remains “resting” the greater the energy savings are.

3.2.1. Multithreading

When your computing system has multiple cores, you can use that to your advantage when writing a program. With multiple cores, rather than having threads (instructions) be done one by one, each different core can handle different ones. This will result in a faster completion of the work of the program and a faster return to idle state. However, you have to make sure that the threads are balanced to take full advantage of this Make sure that you put to use every core you can use. If the threads are not balanced, then it will cause unnecessary traffic at the cores, which will then elongate the time to complete the work of the program.

3.3. Data Efficiency

How the application data is handled can have an effect in reducing the energy required to complete a task. One way to do this would be to pre-fetch any data that your program might need prior to running it. This way, you can prevent frequent reads and writes, which alone would take up energy.

Another way to handle data efficiently is to move data close to the processing entity. This means that the efficiency of the program will improve if, for example, we move the necessary information to the cache and then have the program fetch it from there rather than getting it from the main memory all the time.

Also, if you are using multiple cores in your program, it would be to your advantage to to see how memory is shared between them as suing shared resource can prevent other some cores from going into a sleep state because the data is still in use although that core is some cores may no longer be busy. To prevent this, try to synchronize the threads on different cores to work simultaneously so that they may also idle at the same time.

3.4. Context/Power-Aware Behavior

This section is about steps that applications could be programmed to take when a computing device is going from being in use to going into sleep or hibernation mode or when the power source is changed.
3.4.1. Handling Sleep Transitions Seamlessly

Power Awareness in applications can be improved by having them adapt to platform sleep and wake-up transitions. This means that they should be able to handle the transitions without requiring a re-start, losing data, or changing state. This means, that, for example, if you are working on your laptop, and then you get up to rest or do another task that takes you away from the computer and then it goes to sleep, then you should not risk losing unsaved data just because you unplugged it or because it is going to sleep. Here are some things that applications should be able to do in order to prevent this from happening:

- The application should save the data or state of the system prior to the sleep transition and then restore it when it is woken up
- Close all files and I/O devices prior to the transition
- Close all communication links before the transition and then restoring them once it is awoken
- Stopping any ongoing user activity such as streaming video or downloads upon transition.

3.4.2. Respond/Adapt to System Power Events

When running a computing system on battery, it is advisable to program application so that when the battery is at a low power level that work or state are saved in case the battery does run out of power.

3.4.3. Scale Behavior Based on Machine Power State

You, programmers, should consider the following additions to the programming of your applications that should take effect when battery power is low:

- Do not allow your application to download any material
- Perhaps include a special “low-power” algorithm that will kick-in in this situation so that your application will run smoothly while draining as little power as possible
- Reduce the quality of video and audio playback in DVD applications if being used during travels and an electrical power source is not available
- Turn off spell check so that no energy is drained from the battery processing proper ways to spell the words you type.

3.4.4. Context Awareness Toolkits

Once again, this section covers Intel-specific material about context awareness tool kits available in their machines such as the Intel Laptop Gaming TDK. For more specifics on these products, please refer to the source of this summary.

3.4.5. Unused Peripherals

Program applications that may not be used by many users, such as applications that require Bluetooth-enabled products, in such a way that they will not be usable, or even will not open unless a Bluetooth device is connected to the computer.
3.5. Tools and Testing for Energy Efficiency

In this section, the focus is on tools available across different platforms that address power-related frameworks, optimizations, and measurements. These include Intel Power Checker for Microsoft and PowerTop for Linux, but there are many more, and they don’t go into great detail about them. However, once again in the in paper they do provide web addresses of where you can find more information about them, so, I again suggest you read it to find out more about these tools.

4. Developing Green Software

4.1. Introduction

This paper works as an extension of the previous one, although it doesn’t focus on every topic presented. The difference between the two is that this one goes into more detail about how the topic presented will help conserve the energy consumption of a battery on a computing device. Throughout the years there has been gradual improvement on this area of computing when it comes to hardware, but there are still many things that can be optimized. Software can play a big part in all of this.

4.1.1. Processor P and C States

The primary interest of software developers should, in theory, be the CPU. As such, knowing the CPU’s defined energy states can help developers make great decisions when it comes to energy-aware programming. These states define what the CPU should do during various states of activity. These states are known as C-states and P-states.

C-states are defined as “…core power states that define the degree to which the processor is ‘sleeping’” [3]. Therefore, these kinds of states are often known as “sleep” states. The most-common C-state is C0, as it is available in all CPUs. During this state the processor is at its most active and executing instructions. In between periods of activity, however, the state will change to allow the processor to rest. As the state gets deeper, a new part of the CPU will be turned off, therefore the ability to save power will increase. Table 4 from the web article Everything You Need to Know about the CPU C-states Power-saving Modes gives a brief overview of what each C-state is and does. For more detailed information. For full details on the states, I would suggest reading the full article.

P-states, also referred to as performance states, “define the frequency at which the processor is running” [3]. P-states only work during C0. Just like with C-States, the deeper the state-is, the less energy will be consumed as the voltage and frequency is lowered. Here are the three most common P-states:

- P0- Maximum power and frequency
- P1- Less that P0, voltage frequency scaled
- Pn- Lowest rated voltage/frequency
<table>
<thead>
<tr>
<th>Mode</th>
<th>Name</th>
<th>What it does</th>
<th>CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>Operating State</td>
<td>CPU fully turned on</td>
<td>All CPUs</td>
</tr>
<tr>
<td>C1</td>
<td>Halt</td>
<td>Stops CPU main internal clocks via software; bus interface unit and APIC are kept running at full speed.</td>
<td>486DX4 and above</td>
</tr>
<tr>
<td>C1E</td>
<td>Enhanced Halt</td>
<td>Stops CPU main internal clocks via software and reduces CPU voltage; bus interface unit and APIC are kept running at full speed.</td>
<td>All socket LGA775 CPUs</td>
</tr>
<tr>
<td>C1E</td>
<td>—</td>
<td>Stops all CPU internal clocks.</td>
<td>Turion 64, 65-nm Athlon X2 and Phenom CPUs</td>
</tr>
<tr>
<td>C2</td>
<td>Stop Grant</td>
<td>Stops CPU main internal clocks via <a href="#">hardware</a>; bus interface unit and APIC are kept running at full speed.</td>
<td>486DX4 and above</td>
</tr>
<tr>
<td>C2</td>
<td>Stop Clock</td>
<td>Stops CPU internal and external clocks via hardware</td>
<td>Only 486DX4, Pentium, Pentium MMX, K5, K6, K5-2, K6-II</td>
</tr>
<tr>
<td>C2E</td>
<td>Extended Stop Grant</td>
<td>Stops CPU main internal clocks via hardware and reduces CPU voltage; bus interface unit and APIC are kept running at full speed.</td>
<td>Core 2 Duo and above (Intel only)</td>
</tr>
<tr>
<td>C3</td>
<td>Sleep</td>
<td>Stops all CPU internal clocks</td>
<td>Pentium II, Athlon and above, but not on Core 2 Duo E4000 and E6000</td>
</tr>
<tr>
<td>C3</td>
<td>Deep Sleep</td>
<td>Stops all CPU internal and external clocks</td>
<td>Pentium II and above, but not on Core 2 Duo E4000 and E6000; Turion 64</td>
</tr>
<tr>
<td>C3</td>
<td>AltVID</td>
<td>Stops all CPU internal clocks and reduces CPU voltage</td>
<td>AMD Turion 64</td>
</tr>
<tr>
<td>C4</td>
<td>Deeper Sleep</td>
<td>Reduces CPU voltage</td>
<td>Pentium M and above, but not on Core 2 Duo E4000 and E6000 series; AMD Turion 64</td>
</tr>
<tr>
<td>C4E/C5</td>
<td>Enhanced Deeper Sleep</td>
<td>Reduces CPU voltage even more and turns off the memory cache</td>
<td>Core Solo, Core Duo and 45-nm mobile Core 2 Duo only</td>
</tr>
<tr>
<td>C6</td>
<td>Deep Power Down</td>
<td>Reduces the CPU internal voltage to any <a href="#">value</a>, including 0 V</td>
<td>45-nm mobile Core 2 Duo only</td>
</tr>
</tbody>
</table>
4.2. Energy Saving Software Techniques

Now you know a bit about C-states and P-states, but why must you about them? Because the programs you write must behave accordingly to whatever state the CPU is in. If they behave properly, then the energy-saving features built into the CPU will work efficiently. Those that do not behave appropriately will lead to lower battery life and higher energy costs.

Now, what can you do to make sure your program behaves accordingly? Here I will present some techniques that will help your programs to be idle as soon as possible. In the section 3.2 I’ve already covered what this means, and we will touch up on some of the same material, but will go a bit further with them.

4.2.1. Computational Efficiency

We all know the old adage “time is money” and know that it is true from experience. Well the same applies to energy consumption in a computing device when it comes to running programs. The faster a program finishes its task, the faster it can get to an idle state and to saving energy. The key to this is to have computational efficiency. Here are a few things that you can do to achieve this:

**Efficient Algorithms**

When writing a program, there are a variety of algorithms and data structures with which you can achieve your task. For example, in a problem it might be more efficient to use a stack than a queue or a B-tree might be more effective than a hash function. So do properly study the task that your program must do and come up with the best, most energy-efficient way for them to complete their tasks. We had already covered this on section 3.1.1 but it bears repeating.

![Figure 5- Average power consumed if a program is run on a different number of threads.](image-url)
Multi-threading

We have also covered this previously, on section 3.2.1 but here there is proof that using all the available cores in a computer is great for saving energy. Using the Cinebench 11.5 benchmarking application from Maxon Computer GmbH, the energy consumed on a 4-core, 8-thread Intel processor was measured. The following figure shows the CPU energy over time using 1 thread, 2 threads, 4 threads, and 8 threads.

The results in Figure 5 show that indeed completing the workload with a single thread case takes longer. Sure, the 8 thread case consumes larger amounts on power right away, but the tasks take less time to complete. Therefore, in the long run, running the program with the single thread will consume more energy. All in all, the 8-thread case consumed about 25% less power than the single thread case.

Vectorization

Vectorization is when programs that operate one operation at a time on a single thread are modified to perform various operations simultaneously. One way to do this would be to instead of using scalar C-code use advanced instructions such as Single-Instruction Multiple Data for multiple level data parallelism. The scope of this is beyond what I was trying to achieve with this paper, but they do provide an example of what happens when you vectorize a program. For this example they took two different audio decode algorithms and optimized with the Intel Advance Vector Extensions (Intel AVX) instruction set. They tested them both with AVX on and off and here are the results:

Table 5- Performance and Power Impact of AVX Instructions

<table>
<thead>
<tr>
<th>Audio Decode</th>
<th>Time (s) AVX Off</th>
<th>Time (s) AVX On</th>
<th>Speedup</th>
<th>Avg. Power (W) AVX Off</th>
<th>Avg. Power (W) AVX On</th>
<th>Savings on Avg Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>144</td>
<td>87</td>
<td>1.66X</td>
<td>12.7</td>
<td>9.4</td>
<td>4.3</td>
</tr>
<tr>
<td>#2</td>
<td>203</td>
<td>151</td>
<td>1.34X</td>
<td>13.7</td>
<td>11.1</td>
<td>2.6</td>
</tr>
</tbody>
</table>

As you can see, the time that the AVX was on was lower than when it was on. This is because with the vectorization, the decoding was faster. And as you can see, the average power with AVX on is lower.

4.2.2. Data Efficiency

We already looked at what data efficiency means in section 3.3, but to reiterate: data efficiency reduces energy costs by moving data as little possible. It can be done so with:

- Software algorithms that minimize data movements.
- Memory hierarchies that keep data close to processing elements.
- Application software that efficiently uses cache memories.

Here are some examples of where data efficiency methods can be applied.
Managing Disk I/O

The writers of “Developing Green Software” analyzed the power characteristics of the hard disc power during various activities and came up with guidelines to optimize the power during disk I/O. Here are their analyses and their results:

- When reading large volumes of sequential data, reading larger pieces of it requires lower processor utilization and less energy. So, use block sizes of 8KB or greater for improved performance.
- Applications that deal with random I/O or I/O operations with multiple files should use asynchronous I/O to take advantage of the native command queuing (NCQ). Queue up all the read requests and use the events or callbacks to determine if the read requests are complete. This will improve performance and save energy.
- It costs more, in terms of energy and performance, to read a fragmented file than when reading a contiguous file. Pre-allocating big sequential files as they are created, can help avoid this. Also, when multiple threads are fighting for disk I/O, queue the I/O calls and utilize NCQ. Reordering the queue may help optimize the request as well as improve performance and save energy. Another side effect of threads competing for I/O is thrashing (sort of like an infinite waiting loop), so consolidate all the read and write operations into a single thread to reduce the probability of thrashing.

Pre-Fetching and Caching

This was covered in section 3.3, but here we will see if it actually works. A study was conducted between three different DVD playback applications. Each had two modes, one for maximum energy savings and one for no power savings. The power consumption for each was measured in both modes using a standard definition DVD that was included with the MobileMark 2005 benchmarking tool.

Here are the individual characteristics of each DVD application exactly as described in DVD Playback Power Consumption Analysis by Rajshree Chabukswar, which is referenced in Developing Green Software:

DVD Application 1:

• No power-saving mode: Data is read as it is processed. The DVD is accessed 100% of the time during the playback.

• Power-saving mode: Data is buffered to reduce DVD spin-ups and spin-downs. The buffered data is processed over certain durations of time. The power-saving modes are divided into three modes, based on the amount of total data being buffered with each DVD spin-up. For this study, we focus on the highest-level power-saving mode: aggressive/max power-saving” [6].

DVD Application 2:

“1. Maximum Performance/No Power-Saving Mode: The processor runs at maximum available speed, irrespective of the system power scheme.
“2. Maximum Power-saving Mode: The processor runs at optimal speed, based on the power policy selected” [6].

DVD Application 3:

“Battery optimized performance-enabled (maximum power-saving) or disabled (no power-saving)” [6].

And here are the results of the analysis:

<table>
<thead>
<tr>
<th>Application</th>
<th>Mode</th>
<th>DVD Energy (mWHRs)</th>
<th>CPU Energy (mWHRs)</th>
<th>Platform Energy (mWHRs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVD App 1</td>
<td>No Save</td>
<td>869.84</td>
<td>663.92</td>
<td>6618.76</td>
</tr>
<tr>
<td></td>
<td>Max Save</td>
<td>263.41</td>
<td>762.99</td>
<td>6039.41</td>
</tr>
<tr>
<td>DVD App 2</td>
<td>No Save</td>
<td>897.82</td>
<td>3329.53</td>
<td>10143.56</td>
</tr>
<tr>
<td></td>
<td>Max Save</td>
<td>895.57</td>
<td>1064.02</td>
<td>7509.18</td>
</tr>
<tr>
<td>DVD App 3</td>
<td>No Save</td>
<td>780.93</td>
<td>703.25</td>
<td>6202.04</td>
</tr>
<tr>
<td></td>
<td>Max Save</td>
<td>781.04</td>
<td>554.87</td>
<td>6023.57</td>
</tr>
</tbody>
</table>

As you can see, DVD application 1, which buffers all the content of the DVD in maximum save mode, consumes less energy. It reduced DVD power consumption by about 70% and overall energy savings are about 10% lower than the other techniques. So, it is better to buffer so as to reduce the number of DVD spin-ups and spin-downs. Also, we see that application three, which works with a platform that optimizes battery power has lower CPU energy consumption, so it’s better that the OS sets the appropriate P-state and adjust frequency as needed.

4.2.3. Context Awareness

In section 3.4, context awareness was touched upon in relation to handling sleep/wake-up transitions efficiently, but that doesn’t quite capture the entire scope of what it means. We, humans, are generally smart enough to quickly react to whatever changes happen around us, meaning that if it’s cloudy we know it might rain so we might take an umbrella with us if we are going out or that if you cut yourself you need to tend to that quickly. Computers should be the same way, in that they should know how to react to any changes in the form of power they are running on. If the computer is plugged into a wall outlet, then it should know it and run as it has to. But if it’s running on battery then it should dim the screen right away, or perhaps not allow certain program, such as those that require webcams, to run while on the battery.

These types of context awareness actions can be passive or active. A passive response would be one that asks the permission of the user before it can do its job. An active one acts right away without permission. Applications can take advantage of context awareness to save energy. Here are a couple of examples that can attest to this.
AC or DC?

In windows there is a GUID called GUID_ACDC_POWER-SOURCE. This GUID can tell an application if, for example, a laptop computer is plugged into a wall power outlet or operating on battery. Therefore, you may be able to adapt your program’s behavior and maybe extend the life of the battery.

Platform Power Policies

Microsoft Windows comes with three built-in power-policies- “High Performance,” “Balanced,” and “Powersaver.” The user then has the option to choose one of them while their system is operating on battery. Each one adjusts the behavior of applications in response to the chosen power policy.

4.2.4. Idle Efficiency

It has been mentioned many times that a great way to save energy and extend battery life is to let the machine idle as much as possible. But does it really work? A study was conducted to see if it did. The power usage of running the Office Productivity Suite on an Intel Core 2 Duo platform running Windows Vista was measured and here are the results (with minimal background activity). Here are the results

![Average power scenario on running MM07(Office Productivity suite)](image)

Figure 6- Power consumed while Office Productivity suite is running.

The spikes in the chart represent the times when the computer was at C0 state. As you can see, work was completed quickly so that it could go back to an idle state as soon as possible. The results was that the processor was in state C0 only about 5-10% of the time and in idle about 90-95 percent of the time, consuming only about 8 watts of energy at the time, compared to close to 40 at its highest peak. So, yes, it works.
4.2.5. Deep C-State Residency

For the energy savings shown in section 4.2.4 to come true, the processor must be in deep C-states (preferably C6 or lower) during at least 90% of the time. Also, C-state transitions must be kept to a minimum, as the opposite would not be energy efficient. To reduce C-state transitions, it is recommended that tasks are not split between processes or threads unless they can be executed in parallel. If they must be split, schedule the work so that the number of transitions is not as great.

4.2.6. Background Activity

This section mostly relates to operating systems, but it’s also similar to section 3.4.5 in saying that it’s important to turn off applications that are not going to be used. For example, don’t have Skype start automatically when your computer is turned on. Instead, open it as needed.

4.3. Summary

In our modern world Green technologies are needed more and more every day, and software can also be “green” as longs as it behaves accordingly to the specifications of the CPU. Keep the following in mind when writing software or programs:

- Taking advantage of performance features by emphasizing Computational efficiency
- Reduce data movements for better date efficiency
- Make your program “smart” and implement context awareness
- Remember that keeping the computer at an idle state saves energy and look for ways to improve idle efficiency.

5. Conclusion

As more and more people around the world have access to a computing device, whether it is a tablet or a smartphone, the energy consumption needs increase. As a result, our carbon footprint
grows in size, and affects our one and only planet. Just as the people in organizations like Green Peace do their part to protect it, programmers must also do their part by making their programs and applications more energy efficient. It may seem as if the energy savings of one little device doesn’t matter, but considering the number of them out there, it all adds up and will continue to do so over the years. Therefore I believe that it’s no longer an option whether or not to create energy efficient software, but rather, it’s our responsibility. In this paper I have summarized a few viable techniques that experts in this area have proven to be effective. If we all do our work right, companies with fancy computing centers and people who depend on multiple devices for their work, will not further affect our already-fragile planet. And although in the end we would not get a ton of gratitude, we can rest knowing that because of our work, our planet will be improved.

References


