Survey: Techniques for Efficient energy consumption in Mobile Architectures

Sean Maloney
University of California, Santa Barbara
sean@cs.ucsb.edu

Ivan Boci
University of California, Santa Barbara
bo@cs.ucsb.edu

March 16th, 2012

Abstract

As the world becomes more dependent on mobile technologies, battery life becomes an important factor in the success of a system. We survey several different solutions that have been proposed to increase battery life by optimizing the system architecture and software of mobile devices. We then comment on the lack of performance testing done by the papers and present a conclusion.

1 Introduction

Mobile computing has become integrated into everyday life – from a teenager texting with friends, to a high powered CEO working on a laptop in a transcontinental flight. The success of these products depends on how well they can fulfill the users needs but they wont be able to do that if they run out of batteries. So, to support mobile users we must address power management when designing and implementing the system architectures for these devices. This is a complicated task because extending battery life is a delicate balance of give and take between longer battery life and more functionality. For example, you can extend the battery life by disabling all network communication, but then you cannot check email or connect to a server or make calls.

This survey will look at several different solutions that have been proposed to help lower energy consumption without a huge loss of functionality. Before we present these solutions for building mobile architectures, we must first understand where all the energy is going in todays systems.

A study published in 2010 [1] measured various key points of a smartphone. Each hardware component was measured separately, with greater detail than shown in Figure 1. For example, they measured each component, such as the GPS, all under different power modes and usage levels. Figure 1 only shows the averages over what the authors deem regular usage. The power drain of each component varies greatly on usage: for example, calls drained $834mW$ on the average, GPS $143 - 166mW$, both of these to be used only infrequently but ultimately depending on user. Screen backlight amounts to between $7.8mW$ to $404mW$ depending on brightness. It should also be noted that the testing device is a 2.5G phone and 3G drains more power[2].

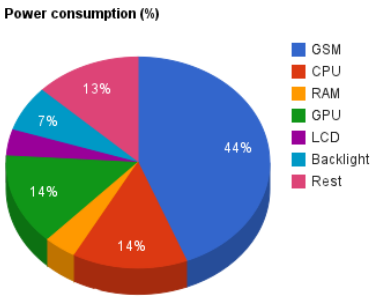


Figure 1: Average power consumption per component for a modern smartphone

Their findings suggest that the way to more power is in optimizing the GSM module. However, we as computer scientists can barely grasp all that signal processing stuff and leave that to our ECE friends. Instead well focus on the other most important culprits. In Section 2 we look at how to optimize the CPU with new operators and new caching architectures. We also look at how to reduce the energy consumption within the LCD display controller. In Section 3 we look at techniques for conserving battery life when using location sensing like GPS or WiFi. Finally we will summarize our findings in a conclusion (Section 4).

2 Low Power Mobile Architectures

2.1 Low Power Operators

One method to reduce power consumption is to rework how components of the CPU function. [3] proposes a new architecture for multiplication operators after studying statistics on how they are used. In order to gather these statistics of typical multiplier operands the authors test with four benchmark programs. These are: go (a go-playing game), ijpeg (a standard JPEG compression/decompression program), compress (a file compression program) and vortex (an object-oriented database). After looking at 33 million instructions involving the multiplication operator they found some interesting facts:

1. The statistics show the distribution of positive and negative operands is highly imbalanced, The great majority of inputs are positive.

2. The 'Significant Bit Count' (SBC) is the number of least-significant bits at the bottom of a binary number ignoring all of the most-significant bits which constitute a series of '1's or '0's. The SBC distribution is not balanced. Most inputs have a low SBC of between 0 and 16 bits.

With these statistics in mind a new asynchronous, iterative multiplication operator is suggested. Normal multiplication operators (list or tree multipliers) function by 1.) generating partial products 2.) adding the partial products together to get one partial sum and one partial carry 3.) sending the sum and a partial carry to a final adder to get the final result. This process involves extensive overhead while working with negative numbers. These operators are very fast but also very power hungry because they involve a significant amount of bit transitions, from 1 to 0 or 0 to 1. The low power asynchronous multiplier minimizes this issue with the structure in Figure 2 and a sign changing algorithm.

From this architecture you can see that instead of using one large register, the operator uses split registers. This is because large registers put a heavy load on the control wires. By splitting the large register into two smaller ones we can decrease the capacitance of the registers and wires and improve the power efficiency greatly. The proposed multiplier also uses an asynchronous latch controller that shuts off all current to the datapath circuits when the operator has completed its job. This control may stop unnecessary switching activity, which saves even more power.

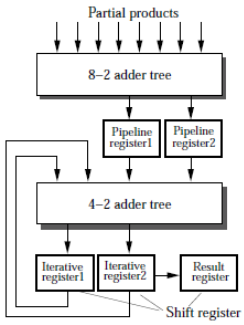


Figure 2: Pipelined iterative multiplication operator

Multipliers	**SNN**	**ABN**	**ANN**	**ANS**
Power (mW)	16.17	9.56	7.35	6.47
Ratio	22	1.3	1	0.88

Table 1: Power comparisons among 4 multipliers

The power savings of this suggested architecture was was tested using simulations with `HSPICE` by comparing several different multipliers. The multipliers tested were: a synchronous non-Booth's multiplier with unified registers (SNN), an asynchronous Booth's multiplier with unified registers (ABN), an asynchronous non-Booth's multiplier with unified registers (ANN) and an asynchronous non-Booth's multiplier with split registers (ANS which is what the authors proposed). Table 1 shows the results to be quite conclusive that the proposed solution is more energy efficient.

2.2 Power Efficient Cache Designs

Caches consume a significant amount of energy in mobile CPUs. So in order to extend the battery life of the devices we must optimize these caches by considering the best architecture with minimal energy use. Having a better hit-rate will minimize need to go to off-chip memory and save power but there have already been many papers focusing on how to improve cache hit rates. The traditional cache architectures are direct-mapped and set associative. A set associative cache usually has a better hit rate than a direct-mapped cache of the same size. Although the amount of bit line switches in the set-associative cache is usually more than that in the direct-mapped cache, the energy use of each switch in a set-associative cache is less than that in a direct-mapped cache of the same size. With that background the authors of [4] present three new types of architectures then analyze their strengths and weaknesses.

1. Vertical Cache Partitioning: The main design goal of vertical cache partitioning is to increase the number of on hierarchical chip caches in order to reduce the amount of energy used. This is done because accessing a smaller cache has a lower power consumption since it requires less energy to pass through each capacitor and retrieve the bits. One example of vertical cache partitioning offered by the authors is called block buffering.

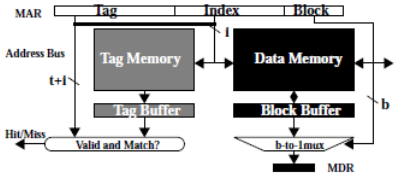


Figure 3: Block Buffering Architecture

The block buffer is an additional on chip cache that lives closer to the processor than regular on-chip caches. Figure 3 shows describes how the block buffer works: The processor checks if there is a block hit. If it is a hit, the data is directly read from the block buffer and the cache is not operated. The cache is operated only if there is a block miss.

2. Horizontal Cache Partitioning: Like Vertical Cache partitioning, Horizontal Cache Partitioning focuses on decreasing the size of the chip caches in order to reduce the energy used. But it differs in that it segments the caches and each segment can be powered individually. By doing this, only the one segment that is being accessed will consume energy. One example of this architecture is called cache sub-banking (Figure 4)

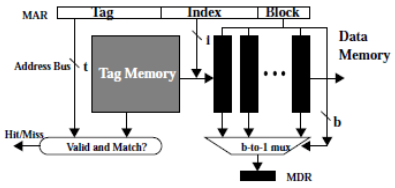


Figure 4: Cache Sub-banking architecture

In cache sub-banking, the data array is broken apart into several banks called cache sub-banks. Cache sub-banks save power by eliminating unnecessary accesses. The more sub-banks there are, the more power you save. Determining which sub-bank data is stored in is trivial because it is indicated in the index block included in the memory address.

3. Gray Code Addressing: Memory addressing used in a traditional processor design is usually in a 2s complement representation. The bit switching of the address buses when accessing sequential memory space is not optimal. Since there is a significant amount of energy consumed on the address buses and sequential memory address access are often seen in an application with high spatial locality, it is important to optimize bit switching activities of the address buses for low power caches. Gray code addressing has been proposed to minimize the number of bit switches of the address buses when a consecutive memory space is accessed. A Gray code sequence is a set of numbers in which consecutive numbers have only one bit different. In the case of a sequence of numbers from 0 to 16, there are 31 bit switches when the numbers are coded in binary representation

The authors conclude by testing both the regular conventional set-associative and direct mapped caches as well as the solutions that they presented above. They found that set associative caches (for both instruction and data) consume less energy than direct-mapped caches when the caches are implemented in static logic. Caches with sizes from 4K to 16K bytes t consume the least energy of all other cache organizations. Results show that 33% and 12% of the bit switches on the instruction and data buses can be reduced by using Gray code addressing. When both Gray code addressing and cache sub-banking are applied on a four-way set associative cache, the overall cache energy consumption is only 23.11% of the total cache energy consumption when the cache without applying the technique.

2.3 Fuzzy Display Controller

Lighting a laptop or cellphone screen takes an immense amount of power in order to function properly. LCD screens are now the default configuration for handheld mobile systems. In most cases, the screens use a transmissive LCD which uses a backlight source so that the screen can be read in dim areas. Transmissive LCDs consume more power than reflective LCDs which reflect light back to the user but require operation in bright conditions [5]. Since most mobile devices contain transmissive LCDs we will focus on their architecture. A white LEDis used by the

backlight circuit in most mobile displays and the architecture of the display appears as Figure 5 below.

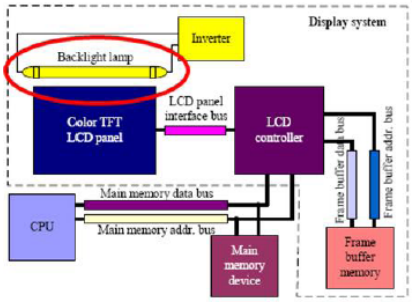


Figure 5: Block diagram of a mobile embedded system with LCD display

In the diagram above, we see that there is part of the mobile device called the LCD controller. This component interacts with the various different structures within the architecture such as the CPU, GPU (not shown), and the Frame buffer in order to display images. One paper [6] investigates the option of taking input from different layers of the protocol stack and user inputs to optimize the duration of current through the LED thereby improving the battery life. The proposed architecture in the figure below is a logical depiction of the LCD controller. The authors state that this architecture can be implemented for any cell phone with any technology.

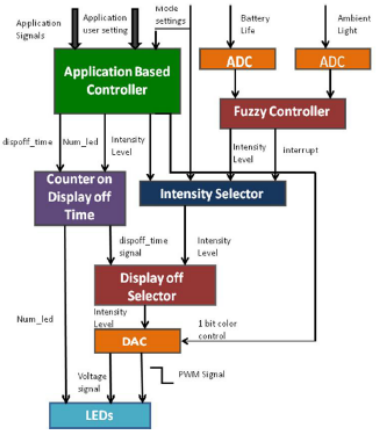


Figure 6: Multi-Layered Cross Functional LCD Controller

The controller depicted in Figure 6 takes in several different inputs including: the current battery life, the ambient light in the room, the current power-saving mode, customized user settings, and signals from the individual applications. This controller is implemented with both hardware and software protocols. Below we will go into greater detail about what each layer of this controller does.

1. Application Based Controller: finds the current apps running on the cellphone as well what the current power mode is. Turns the screen off or dims based on those inputs.

2. Fuzzy Controller: When no user input provided, takes in the ambient light measure and the battery life to set the intensity level of the LEDs. This is called a fuzzy controller because these inputs are integers from 0-255 and are offset then converted to 3 membership functions of low, medium, high based on the range they fall into. These two values of low/medium/high are applied to a set of rules which determines an output LED intensity of low, medium, high. Then this intensity value is defuzzified and converted back in to a real digitized value of intensity

3. Intensity Selector: Takes the input from the Application Based Controller and the Fuzzy Controller and gives the appropriate value based on the power mode. If no power mode or specific user settings are set, then the intensity selector will take the fuzzy intensity value as its output. Otherwise it take a ratio of 60% of the application based intensity and 40% of the fuzzy controller intensity.

4. Counter on Displayoff Time and Display off selector: Display off time is typically 5-10 seconds in a cell phone. Counter counts on this signal and indicates display off time selector when count completes. These two blocks together decide on the number of LEDs to be turned on and the intensity value (either zero or the one coming from intensity selector) of LEDs at a particular time based on the running application.

In order to test this Fuzzy Controller architecture [6] implemented it with Hardware Description Language (HDL) and simulated it on a Xilinx10.1 emulator. While the control itself consumes little power and time, they did not give any percentage of saving in power consumption stating that power will be saved depending on real data, users choices and current application. The authors of [6] could have assumed a set of static user choices and applications to benchmark but they did not.

3 Sensor Solutions

Mobile devices are equipped with various sensing devices that make the device more versatile and powerful. However, these sensors significantly contribute to the total power consumption of the device. In this chapter well overview these sensing devices, see which are the most costly and how to mitigate their cost.

3.1 Background

Modern mobile devices are equipped with many sensors that serve various system and third party applications in abstract ways. The most common today are the accelerometer, GPS, compass, a sensor that measures distance from the screen to the closest obstacle etc. The accelerometer is used by the system to determine the orientation of the handset, but is also used as input to third party applications such as games. GPS is used to determine the location of the device, on which even background running processes may be dependent.

There are also sensors which run only when directly invoked by the user. These are the microphone and/or camera, or multiple cameras; sometimes multiple audio sensors, the bluetooth signal detector, even a barometer and/or a thermometer. These devices are to be run when the user needs them because of their big power cost.

Because these input devices vary not only in support by different devices and OS versions, but also in quality. They are usually hidden behind a carefully designed API that lets applications access these features in a uniform manner. Features that are not supported are marked as such by the API, but when possible, applications can still read the inputs that the missing feature would have supported; read values being the best approximation that can be made by using other sensors that the device does have.

In 2009, a survey [7] was done on how different sensors affect battery life. They used a Nokia N95 smartphone and measured the battery life when only one sensor was continually on from a full charge to a shutdown. The results can be seen on Table 2.

Judging from those findings, it is obvious that sensors use a significant amount of power. The battery life we get by running most of these sensors uncontrollably is much shorter than the generally acceptable norm of about 1 day of usage per charge.

We will present two papers that focus on this problem, in particular location sensing. Location sensing is a costly matter, as shown in the above table. Yet many applications depend on it: not only different map and navigation applications, but also applications that keep track of your movement distance for fitness purposes, the Twitter Android app, applications that run in the background and post and update your location on social networking sites.

Location sensing is provided through a simple API, along the lines of Give me the current location and precision of that measurements. This information can be provided by multiple sources: GPS being standard, but relative locations of mobile radio transmitters and WiFi hotspots can help out as well for a much smaller energy cost. On a software architecture level, the accelerometer may be employed to make the more costly location sensing less frequent.

3.2 Less is More: Energy-Efficient Mobile Sensing with SenseLess

In order to keep location sensing power cost down the authors of [7] suggest a system that will utilize less costly sensors more frequently, without compromising the precision of read data. The inspiration for this system is that location sensing is unnecessary if the subject is not moving (where data needs not be updated).

Judging from the low cost of using an accelerometer, the authors created an application that read accelerometer data every 10 seconds and logged it into a file. This application was installed on several phones and was given to multiple users who were to simulate normal behaviors: walking, sitting on a couch, biking etc. Their measurements show that euclidean distance between successive measurements is much greater while moving. They also show occasional exceptions: for example, even while biking it is possible to stop at a red light; while people do suddenly move while sitting on the couch.

After this they implemented SenseLess, a location system that bypasses GPS while stationary. A device considers itself stationary when 3 successive accelerometer measurements show little to no change. Note that this logic makes the system very cautious to deem a device stationary; the reason being, a false negative (detecting movement while stationary, like changing the couch seat or reaching for the remote) implies a slightly higher power consumption at that particular moment. A false positive, on the other hand (detecting little accelerometer dynamic while actually on the move) has a great cost of imprecise location measurement which may be crucial in cases of car navigation. Not detecting movement for some time after a red light, for example.

In addition they implemented a switch between GPS and WiFi based location sensing. GPS is more precise and power efficient, but is useless indoors. In-

Sensor	Approx. Battery Life (hrs)	Average Power Consumption (mW)
Video Camera	3.5	1258
IEEE 802.11	6.7	661
GPS (outdoors)	7.1	623
GPS (indoors)	11.6	383
Microphone	13.6	329
Bluetooth	21.0	221
Accelerometer	45.9	96
All Sensors Off	170.6	26

Table 2: Approximate battery life of cell phone censors for a Nokia N95

doors, WiFi is a more costly location source but that should be turned off when outdoors. So the authors keep GPS on at all times to detect of the device is indoors/outdoors; when a GPS signal is detected, the device is considered outdoors and location sensing by WiFi is turned off.

After implementing this system they gave it to multiple users who are supposed to simulate everyday activities over about an hour of walking, sitting, indoors and outdoors. Some devices had SenseLess while some were on stock GPS. Their results can be seen on Table 3. In the paper they also show that location fidelity is not compromised with SenseLess.

While this system shows significant power savings, they do not show if those are contributed by detecting if the stationary state or by GPS/WiFi switching, and in which ratio. There is also the question of estimating movement by the accelerometer and extrapolating location from the previous measured location using the accumulated movement vector and amount of time that has passed; this would make accelerometer readings more frequent but hard location readings more rare.

3.3 Improving Energy Efficiency of Location Sensing on Smartphones

Another paper also tries to lessen the overall cost of location sensing through middleware. It starts by listing references to common Android apps, showing how popular they are and the tricks they employ to preserve battery life, without which they would chase the customers off.

One popular trick is to reduce the frequency of location sensing for apps that dont need up to date information. For example, the Twitter app only needs to know the general area when a Tweet is being sent. It need not be precise, only knowing the town is precise enough; and also, it need not be updated for tweets that are made shortly one after another. However, the authors point out that, even though multiple apps employ this trick, they do so independently of each other; making overall GPS sensing much more frequent.

They also did experiments measuring the power spent on an idle phone vs GPS enabled, on using network-based triangulation for location sensing vs GPS, and on the actual GPS sensing frequency by running multiple location-based apps vs a single. They show that GPS is about twice as costly as network-based location sensing, but also more precise; net-based location sensing gave an accuracy within 30-100m of their true location vs GPSs 10m precision. While relatively imprecise, location-based sensing is more then precise enough for many purposes; weather location being an example.

The paper presents four independent systems that, run as middleware together, reduces the number of GPS invocations by up to 98%, thus improving battery life by up to 75% [8].

* Sensing Substitution: The Android location API allows location sensing after the application states it's required precision, if it needs direction or not, altitude, speed, etc. After this query the OS picks the most-efficient location sensor that suffices the criteria.

  The problem with this system is that it is not dynamic or environment aware. Sometimes GPS gives worse results than network-based triangulation, such as when indoors. This lack of flexibility goes unnoticed; and an application asking for a precise location will keep using GPS, draining battery unnecessarily. In rural areas network-based triangulation is less precise, and in those running GPS very briefly and using partial data may give better and cheaper results then scanning for cell towers over a long period of time.

  Sensing Substitution is a process that runs in

	GPS-only	SenseLess
Average power consumption (mW)	454	118
Average CPU load (%)	1.5	2.6
Electric charge (mAh)	131	54
Estimated battery life (h)	9.2	22.2

Table 3: Power comparison of GPS vs SenseLess

the background and monitors the precision and availability of GPS vs network-based location at geographical areas. Over time, it learns to use the better choice in those areas. They claim that this system is logical because users usually spend time in a few areas; home, work and shopping, for example. When a user enters a known location, learning need not happen because the quality of either sensor is known, so only the superior location sensor will be used for a given precision constraint.

* Sensing Suppression: this system tries to suppress location sensing when the device is not moving. They use the accelerometer and an orientation sensor to the same effect SenseLess does. In addition, they give the user a choice of turning this system off when needed, and add a verification system that will periodically check if the location has changed while in static mode. One other difference is that they monitor changes every 20 seconds and every 2 seconds independently, because they have observed that patterns of motion can be missed by either frequent updates or infrequent updates, but rarely both.This contribution alone overlaps and is more significant than the entire SenseLess paper.

* Sensing Piggybacking: by default, applications need location updates and make a request periodically. For example, lets say that one application asks for an update each minute and another each 40 seconds. Because these applications do not share information, they together run 2.5 location sensor invocations per minute.

  This can be optimized by having the sensor run each 40 seconds, only for the purposes of the second process, and just passing the most recent information collected to the first process. This first process will read locations old up to 20 seconds, which is more then acceptable giving that it asks for measures each minute. This would yield a total of 1.5 location sensor invocations per minute.

  Sensing Piggybacking is a system that implements this approach in a dynamic nature; the frequency of location requests are monitored on a per-application basis, from where the actual frequency of detections needed is estimated and actually done. It is also made more complicated by having two different sensors available, where the less precise one can piggy-back on the more precise sensor.

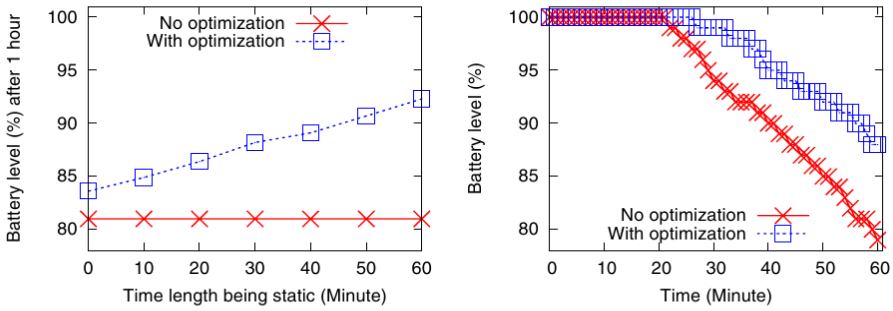
* Sensing Adaptation: when the battery is low, users may be fine with sacrificing location precision for some more life in their device. The user is then shown an option to preserve power; if he accepts to do so, the parameters of the above mechanisms are altered, and the more power efficient mechanism is being used even if the more precise mechanism is requested by the application.

The authors then proceed to explain the experiments done to measure the effectiveness of each of these solutions, discussing results in detail. This is omitted in this report for brevity. After they show experiments done on a system with all four systems integrated, where they show that location invocations become as 98% less frequent, yielding an improvement of up to 75% longer battery life, depending on location and sensor availability, using different apps and use cases (mobile vs static).

Figure 7 (a) shows battery life before and after these optimizations for a custom built location-based application, and Figure 7 (b) is done on some standard use case where the user was half walking, half standing still over an hour. The fact that the battery was at 100% for the first 20 minutes is an error in battery state readings that was consistent through their work.

3.4 Commentary

The SenseLess paper [7], although 1 year older and cited by [8], is not nearly as rich as [8]. [8] also mentions other approaches to the static/dynamic location sensor optimization, involving sound detection.



(a) Custom-built LBA (b) Real Time Traffic

Figure 7: Before and after power consumption

[8] has presented their case in a more precise manner, with better experiments supporting it.

One thing [8] did not show is the (most probably) increased CPU cost, focusing only on sensing frequency and total energy spent over an amount of time. Increased CPU cost is to be expected, but may be turn out to be a significant figure that worsens the entire user experience.

Given the impressive results, we wonder if improvements on this system are possible. First, altering Androids API to support periodic location evaluations would make piggybacking more efficient and less computation expensive as not only would tracking the periods per process become obsolete, but allow more precise scheduling of location sensings.

Neither of these papers discussed using GPS more cheaply when less precise results are needed. The way GPS works, after satellite position data is acquired, GPS reads incoming data from detected satellites to measure the distance to them. This may take some time if the noise level is all but minimal, in which case the device reads multiple successive signals and tries uses the average, but only after the signal has become consistent and close to that average. Maybe waiting for that average to get steady is not needed for less precise purposes - and using partial readings from GPS would be as costly as network-based location. Maybe, we dont know.

4 Conclusion

In order to help maximize battery life we presented summaries of several papers that propose solutions or study the problem in depth. In cell phones, the top two power hungry aspects are the CPU and the network. We look at the architecture of the CPU, the LCD Display, and the power usage of location sensors like the GPS or accelerometer. Networking was not focus on this paper

For CPUs we looked at a power efficient way of implementing a multiplication operator by using an asynchronous iterative approach. We then present three different approaches to implementing caches that consume less energy than traditional caching techniques: vertical cache partitioning, horizontal cache partitioning, and grey code addressing. The LCD displays in mobile devices also consume a lot of power. We presented a paper that proposes a fuzzy LCD controller, which takes in input like battery life, ambient light, and user settings to dynamically change the luminescence of the screen. In all of these papers, energy tests were performed but few performance tests were shown to illustrate that the low power solutions are comparable to the higher power solutions.

For location sensors, we first presented a paper that studied the effect of the different sensors on the battery life and found that GPS location sensing is more power efficient, but useless indoors, while WiFi location sensing can be used anywhere but takes much more battery. A paper then proposes a solution called SenseLess which helps reduce the GPS usage when the cellphone is not moving by taking measurements from the phones accelerometer. We also look at 4 different middleware techniques that are used to limit calls to the GPS hardware in order to preserve battery life: sensing substitution, sensing suppression, sensing piggybacking, and sensing adaptation. Unfortunately these papers do not present evidence that the CPU cost of calculating when to limit the GPS is also energy efficient.

References

[1] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, (Berkeley, CA, USA), pp. 21–21, USENIX Association, 2010.

[2] J. Akhtman and L. Hanzo, "Power versus bandwidth efficiency in wireless communication: The economic perspective," in *IEEE VTC'09 Fall*, September 2009.

[3] Y. Liu and S. Furber, "The design of a low power asynchronous multiplier," in *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, pp. 301 – 306, aug. 2004.

[4] C.-L. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: a case study," in *Proceedings of the 1995 international symposium on Low power design*, ISLPED '95, (New York, NY, USA), pp. 63–68, ACM, 1995.

[5] I. Choi, H. Shim, and N. Chang, "Low-power color tft lcd display for hand-held embedded systems," in *Proceedings of the 2002 international symposium on Low power electronics and design*, ISLPED '02, (New York, NY, USA), pp. 112–117, ACM, 2002.

[6] T. Srivastava and A. Narayan, "Dynamic fuzzy controller based multilayered architecture for extended battery life in mobile handheld devices," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pp. 3035 –3040, oct. 2009.

[7] F. Ben Abdesslem, A. Phillips, and T. Henderson, "Less is more: energy-efficient mobile sensing with senseless," in *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, MobiHeld '09, (New York, NY, USA), pp. 61–62, ACM, 2009.

[8] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, (New York, NY, USA), pp. 315–330, ACM, 2010.