Are smartphones really useful for scientific computing?

No Author Given

No Institute Given

Abstract. Smartphones are a new kind of mobile devices that allow users to take their office anywhere and anytime with them. The number of smartphones is rapidly growing. For instance, almost 400,000 new Android smartphones are activated every day. Most of the time their capabilities are underused, therefore several authors have studied how to exploit smartphones for assisting scientific computing. However, to the best of our knowledge, there is no previous study aimed at determining whether smartphones can do a significant contribution to this area in terms of providing aggregated computing resources. This paper shows that conventional smartphones are not that slow when compared to standard mobile devices, such as notebooks. Furthermore, a notebook running on battery only performs 8 times more work than a low-end smartphone before their batteries run out. However, the low-end smartphone is 145 times slower than the notebook, while the smartphone's battery has less capacity than the notebook's battery. Since smartphones can effectively execute an interesting amount of work running on battery, and as a result of their large and ever increasing availability, we think that smartphones can have a major role in building the next-generation HPC infrastructures.

1 Introduction

Mobile devices have recently evolved from simple mobile phones and Personal Digital Assistants (PDAs) to small "computers" with everywhere Internet access using wireless technology, such as 3G or 802.11 wireless LANs (commonly called WiFi). In addition to this evolution, mobile devices are probably the commonest form of technological device in the world with more than 2 billion people owning at least one of them. Furthermore, people in established markets frequently own two or more mobile devices [12].

Nowadays, mobile devices have a remarkable amount of computational resources that allows them to execute complex applications, such as 3D games, and to store large amounts of data. Besides, the capability of mobile devices to be connected everywhere makes it possible to do on-the-way tasks that traditionally required a desktop computer, like checking and sending e-mails. This kind of mobile devices that, aside from placing phone calls, allow users to access the Internet, play games, and listen to music, among other features, are known as *smartphones*. Another kind of mobile devices focused on gaming and Internet browsing are called tablets. Tablets have similar computational capabilities

to smartphones, but bigger screens to improve the user's experience. However, smartphones and tablets can be treated as equal concerning the goal of this paper.

Due to their new capabilities, which are not present in traditional devices, there are emergent research lines that aim at integrating smartphones and other mobile devices into traditional distributed computational environments such as clusters and Grids [14]. Since smartphones have a wide variety of sensors, such as GPS, microphone and accelerometer, they are usually seen as providers of a new kind of context-dependent information, which was previously unavailable [1]. Another common role of smartphones in distributed computational environments is as resource consumers. Offloading computational work to fixed servers and accessing the results through smartphones allows end users to perform complex computational tasks without draining their smartphones' batteries [7].

Although the capabilities of smartphones have increased at an exponential rate [11], little research has been carried out to study the viability of using them to contribute to solving complex computational problems from the engineering or scientific communities. This is due to the fact that practitioners tend to disregard smartphone capabilities [14] because they are very limited when compared to desktop computers or servers. However, they somehow fail to consider the stunning amount of mobile devices currently available that together can represent an interesting pool of computational resources. These limited capabilities combined with the large amount of available smartphones are aligned with the concept of cluster computing, which is basically using a large number of low/mid-end devices to emulate the performance of a high-end computer. In this sense, an interesting use of smartphones might be to increase the computational capabilities of clusters, and eventually Grids at a low cost.

This work studies the viability of using smartphones for performing complex computations to contribute to scientific clusters and Grids. Concretely, this paper presents a comparison among different Android-based smartphones/tablets and Linux netbooks/notebooks in terms of their computational capabilities. In addition, this work analyzes how different types of operations typically used in scientific computing, such as integer and floating-point arithmetic and array management, perform in the different devices. This analysis goal is to have prelimminary hints on which scientific applications smartphones might run effectively.

We have two main reasons for choosing the Android platform as the base for our study. The first reason is that, since Android is widespread, there are plenty of available Android-based devices. According to Google I/O 2011 announcements[1], there are 310 Android-based smartphone models produced by 36 manufacturers. In addition, there are more than 100 million devices active, with 400,000 devices activated daily. This makes our results significant to a broad user community. The other reason is that Android is an open platform that does not restrain developers from using any available feature, such as multi-tasking.

[1] Google I/O 2011 announcements: `http://www.google.com/events/io/2011/announcements-archive.html`

It is even possible to change the default applications, for example the phone dialer, which is a very uncommon feature [2]. As a result of this flexibility, a developer can tune a device to perform different (user) foreground and (scientific) background tasks.

The rest of the paper is organized as follows. Section 2 surveys previous works aiming at integrating smartphones and traditional mobile devices into distributed systems. Section 3 discusses the motivations behind our vision of using smartphones in the HPC scientific computing. Section 4 presents an experimental comparison of different devices in terms of their computing capabilities. Finally, Section 5 concludes the paper and delineates future research opportunities.

2 Background

From the beginings of the mobile Internet, mobile devices have been seen as administrative tools for distributed computing infrastructures [6]. This idea is followed even in recent approaches, such as [7], in which the authors discuss a fluid simulator implemented on a super-computer that offers a result visualizer application that runs on Android. In particular, this visualizer was tested on a Nexus One and uses OpenGL ES –a reduced version of OpenGL for embedded systems– to draw tri-dimensional graphics. The fluid simulator solves several differential equations in the super-computer. When the simulation completes, a user can access the results through his smartphone. Although this work shows that Android smartphones can be used to access the results of this kind of applications, there is no study about using Android smartphones to actually execute computing intensive parts of scientific applications on it. Furthermore, this particular work does not provide empirical evidence about battery consumption when accessing to simulation results. Intuitively, this is important since, in this context, users do not want the visualizer to drain their smartphones' battery [11].

Other recent studies have proposed using mobile devices (including smartphones) as partial or full members of Grid Computing infrastructures [14]. Several roles for mobile devices in distributed computing were analyzed, which vary from sensors to nodes that perform computations. These new roles were proposed because today's mobile devices have more features, and their hardware capabilities have increased at an exponential rate [11]. However, this is not the case with energy density in batteries. As a result of the disparity between computational power and energy density increase rate, most existent research is focused on minimizing battery use.

As suggested, some works have tried to integrate mobile devices as working nodes in Grids. In particular, several researchers [5, 9, 13, 14] have studied task scheduling algorithms for mobile Grids. In this context, a large computational problem is split into several parts, which are computed independently by using many mobile devices. In particular, both [9] and [5] propose scheduling algorithms in which assigning resources to tasks involves solving large equations systems with variables that are difficult, if not impossible, to determine. In contrast, the work presented in [13] uses simple equations and easily estimable

variables. Although these works have their differences, these schedulers have not been evaluated with real mobile devices. Instead, all of them have been evaluated through simulation, thereby there is no real assessment of using mobile devices for HPC computing viability.

Another research line strongly related to using smartphones in distributed computing, regardless their role, is the reduction of network usage [15]. This is important because mobile devices tend to be connected to slow and unreliable networks. Furthermore, some mobile devices, particularly smartphones, are usually connected to expensive data networks such as 3G networks. Interestingly, 3G data transmission needs to actively send radio signals that requires significant battery power. Therefore, sending large amounts of data might result in a fast battery depletion.

3 Smartphones for Scientific Computing

Traditionally, research efforts based on conceiving mobile devices as being part of an HPC environment have been focused on moving expensive computations from devices to fixed servers, thus saving energy at the expense of potentially having greater communication cost. In this sense, *computational offloading* [8] has been recently proposed as a way of running resource intensive computations originated in a mobile device (e.g. a rendering algorithm integrated with the camera) to a Cloud infrastructure. Once the computation is done, the associated results are transferred back to the device. Particularly, this work addresses Clouds but is arguably also applicable to other distributed enviroments.

Alternatively, there are a number of research efforts that conceive mobile devices as an active processing element within a distributed enviroment [14, 10]. Under this approach, devices do not only harness the computational power offered by fixed computing infrastructures such as super-computers, clusters and Clouds, but also act as resource providers. However, works in this line are still under development, since there are in principle two important open questions that need to be answered before materializing this approach:

- *Do real smartphones have appropriate resources to be exploited for executing computing intensive tasks?*
- *If so, what are the best techniques for scheduling tasks on real smartphones?*

With respect to the former issue, particularly, the amount of CPU cycles that can be delivered and eventually donated by a single device intuitively depends on its battery availability. This latter is in turn subject to smartphone usage profiles. For illustrative purposes, Table 1 shows the fraction of time spent by four real smartphones in performing common activities. Collected data spans over several months[2]. Clearly, although generalization is not possible, the data suggests that there is a fraction of resources available in everyday smartphones that remain unused.

[2] Data was collected by using the Android System Info application, which is available at `http://www.appbrain.com/app/android-system-info/com.electricsheep.asi`

	Star A3000	Samsung I5700	Samsung I5500	Nexus One
Running	50.8	13.5	8.3	23.2
Screen on	8.3	7.5	5.1	2.3
Phone on	0.2	0.6	0.4	0.5
WiFi on	93.3	55.0	66.7	100.0
WiFi running	33.2	26.2	19.1	32.3

Table 1. Real-life smartphone activity (in percentage with respect to total usage time)

In addition, even when smartphones still have limited hardware capabilities, they are very energy-efficient. Therefore, we believe that answering this issue boils down to determining whether smartphones *individually* provide the necessary computing capabilities. Our main hyphotesis is that smartphones offer a good balance between the computing capabilities and battery depletion rate tradeoff inherent to mobile devices making them suitable for HPC computing. In the following paragraphs we focus on providing empirical evidence to test this hyphotesis. Methodologically, the followed experimental approach consisted in comparing smartphones and standard mobile devices, namely netbooks and notebooks, in terms of hardware capabilities by employing classical scientific benchmarks. As mentioned earlier, the experiments targeted smartphones running Android 2.2 (codenamed "Froyo").

It is worth mentioning that answering the second question is out of the scope of this paper. Nevertheless, several authors [5, 9, 13, 14] have pointed out that adapting and modifying traditional distributed scheduling algorithms to make them battery-aware might be a good starting point for answering this question.

4 Experimental results

To assess the computing capabilities versus energy capacity tradeoff of Android-powered smartphones compared to traditional mobile devices, we have used a notebook, a netbook, a tablet that has similar hardware compared to a high-end smartphone, a mid-end smartphone, and a low-end smartphone. Table 2 specifies the devices used during this evaluation, which were selected because they are good referents of currently available mobile devices and smartphones. Note that the benchmarks were implemented in a single thread for fairness reasons.

To compare the computational capabilities of the different devices, we executed several well-known scientific benchmarks in every device. We selected benchmarks written in Java because the notebook and the netbook can run Java applications, and the base programming language for Android applications is also Java. Although Android applications are usually written in Java, they are compiled to a different virtual machine, called Dalvik, that is optimized for smartphones. Therefore, before executing the experiments, we compiled the source code of the benchmarks using either Java compilers to ensure fairness.

Device name	CPU	RAM	Storage	Battery
Dell Inspiron	Intel Core i3 M-380 (2.53 GHz)	4 GB	250 GB	6 Cells 4600 mAh
Samsung N150	Intel ATOM processor N270 (1.60 GHz)	1 GB	160 GB	6 Cells 5900 mAh
ViewPad 10s	NVidia Tegra 250 T20 (1 GHz)	512 MB	512 MB up to 32 GB	Lithium Ion 3300 mAh
Nexus One	Qualcomm QSD 8250 Snapdragon (1 GHz)	512 MB	512 MB up to 32 GB	Lithium Ion 1400 mAh
Samsung I5500	MSM7227-1 ARM11 (600 MHz)	256 MB	170 MB up to 16 GB	Lithium Ion 1200 mAh

Table 2. Devices features

Firstly, we compared the different devices by using the Linpack [4] and the SciMark 2.0 [3] benchmarks, which are very popular within the HPC community and provide an estimation of the number of Megaflops (million of operations per second) a machine can perform. Figure 1 depicts the results of the benchmarks

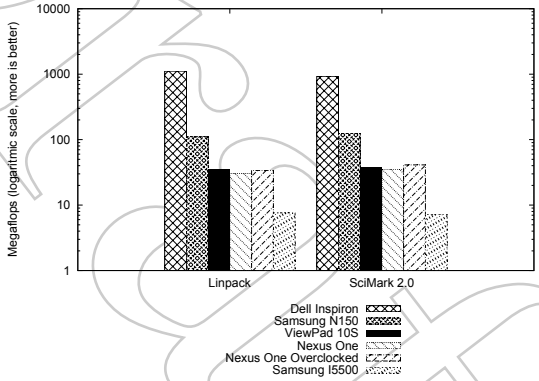

Fig. 1. Linpack and SciMark 2.0 benchmarks: Results

executed in the different devices. The Linpack benchmark results indicate that the netbook, the ViewPad 10s, the Nexus One, and the Samsung I5500 are 10, 31, 36, and 145 times slower than the notebook, respectively. In contrast, the SciMark 2.0 benchmark, which emulates real-life numerical applications, resulted slightly better for the smartphones. In this case, the ViewPad 10s performed 24 times slower than the notebook, while the Nexus One and the Samsung I5500 performed 26 and 127 times slower, respectively. Finally, we executed the benchmarks using the same Nexus One with its CPU overclocked to 1.113 GHz. Although it performed marginally better, this configuration resulted in a battery

overheating when the benchmarks were run several times in a row. Consequently, we were unable to complete the experiments.

Secondly, we compared the power consumption from resolving various complex computations. To do the associated experiment, we selected five benchmarks of the JGrande section 2 [3]:

- EPBench: Is a Java implementation of the NAS Embarrisingly Parallel benchmark, which generates pseudo-random numbers with a Gaussian probability distribution.
- PrimeBench: Verifies whether a large number is prime.
- FFTBench: Is a Java implementation of the NAS Fast Fourier Transform (FFT) benchmark, which computes both a 3D FFT and inverse FFT. FFTs are commonly used in scientific computations, and require extensive floating-point arithmetic and data shuffling.
- HanoiBench: Solves the 25-disk Tower of Hanoi problem, which is a well-known combinatorial sorting puzzle.
- SieveBench: Calculates prime numbers using the Sieve of Erasthosthenes. It consists in integer arithmetic operations with a lot of array accesses.

The experiment consisted in executing the five benchmarks in a round robin fashion repeatedly until the battery of each individual device was empty. Basically, the idea was to measure how many computational tasks could be completed using the standard battery. To ensure fairness, this experiment was performed with all the devices connected to the Internet through a WiFi network. However, the benchmarks did not need Internet access, and no standard system application that requires Internet was deactivated, such as update managers. Figure 2

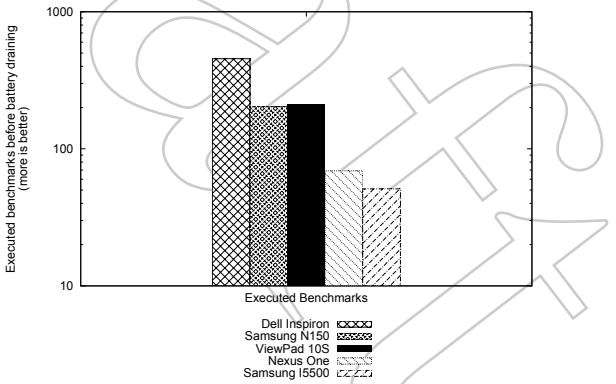

Fig. 2. Executed benchmarks

presents how many benchmarks were executed by each device before they run out of battery capacity. Basically, the notebook executed 2.22, 2.14, 6.50, and

8.90 times more benchmarks than the netbook, the ViewPad 10s, the Nexus One, and the Samsung I5500, respectively.

Although both Linpack and SciMark 2.0 benchmarks pointed out that these smartphones are considerably slower than the notebook, the amount of work that they can perform using only one full charge of battery power does not depend lineally on their computational capabilities. Despite being slower and having less battery, the ViewPad 10s executed more benchmarks than the netbook. This fact suggests that any smartphone can execute complex computational calculations even when they are running on battery, which is the most likely smartphone state [8]. On the other hand, this extrapolation is sound since the Samsung I5500 is an entry-level smartphone which is very less powerful and power-efficient compared to other smartphones available today in the market.

In addition to measuring how many benchmarks were performed before battery depletion, we also measured the average time that each benchmark took to run. Figure 3 presents the result of these measurements. From this Figure, it is

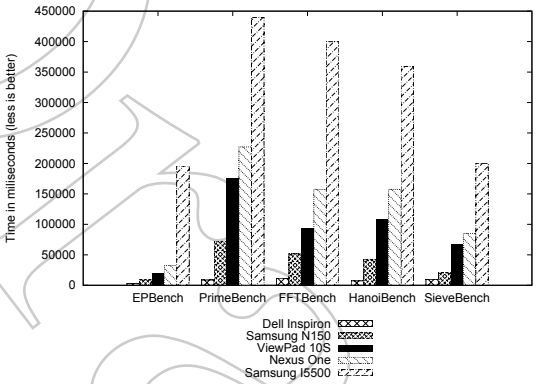

Fig. 3. Average benchmark time

possible to observe that the different benchmarks performed quite different in the employed devices. In this regard, Table 3 presents a comparative result regarding how slower the different devices, including a notebook, a tablet, and two smartphones, executed the benchmarks compared to the notebook. This table shows that there are noticeable differencies among the benchmarks' slowdown within the same device. These variations can be appreciated in the obtained standard deviations that are significant with respect to the average in each case. For example, the largest standard deviation is delivered by the Samsung I5500, which is 42% of the average. Since the implementation used in the different benchmarks was the same, the difference in these results suggests that these devices performed some operations faster than others. This does not contradict the results obtained when executing Linpack and SciMark 2.0 because both benchmarks consist in two similar sets of operations (mostly related to floating point

	Samsung N150	ViewPad 10s	Nexus One	Samsung I5500
EPBench	3.34	7.61	12.04	72.99
PrimeBench	8.40	20.49	26.38	51.07
FFTBench	4.46	8.17	13.64	34.69
HanoiBench	5.84	15.10	21.89	50.01
SieveBench	2.10	6.96	8.74	20.62
Average	**4.83**	**11.66**	**16.54**	**45.88**
Standard deviation	**2.42**	**5.92**	**7.33**	**19.63**

Table 3. Average benchmark execution times with respect to a notebook

operations and loops). On the other hand, the JGrande section 2 benchmark has algorithms that solve inherently complex problems, thereby they excercise other operations, for example object creation or array manipulation.

The final experiment therefore consisted in analyzing the performance differences of several common statements, such as adding integers, creating objects or arrays, etc. To do this, we used the JGrande section 1 benchmark, which measures the performance of these kind of micro-operations. We have used the following statement-level micro-benchmarks:

- Arith: Execution of arithmetic operations.
- Assign: Variable assignment.
- Create: Creating objects and arrays.
- Math: Execution of commonly used mathematical library functions.
- Method: Method invocation in the same or another class/object.

There are some other micro-benchmarks in JGrande section 1, but we did not use them for different reasons. The benchmarks designed to measure casting, looping and exception throwing did not provide useful information because the notebook performs these operations too fast and the benchmark informs that the notebook can perform infinite operations of this kind. Another benchmark that were left out of the analysis was the serialization benchmark because it needs a lot of memory, which Android applications cannot request without using native code.

As a result of this experiment, we have empirically confirmed that, comparatively, some operations performed better than others in the smartphones. For example, the notebook can perform approximately 4 times more integer additions per time unit than the Nexus One, but the notebook, according to the benchmark results, could create 62,051 more java.lang.Object objects per time unit than the Nexus One. This does not necessarily implies that the Nexus One is 62,051 times slower than the notebook when creating one object because, when many short-lived object are created, the garbage collector is forced to run intensively, which is not the case when creating only one object or even few objects. Therefore, the garbage collector is probably introducing an important noise in this case. Since the test consist in creating thousands of objects, it is

likely that the garbage collector in Android is launched many more times than the garbage collector in standard Java because Android applications have rigid memory constraints. Despite this, creating a large number of short-lived objects is not a good practice for Android applications.

	Samsung N150	ViewPad 10s	Nexus One	Samsung I5500
Arith	3.45	6.43	9.07	134.07
Assign	87.79	154.19	164.14	456.11
Create (Object)	7575.30	42,009.04	90,931.63	114,570.52
Create (Array)	3.95	221.87	435.31	526.79
Math	8.34	36.02	36.87	134.47
Method (Same)	189.93	23,983.98	9,362.91	14,532.00
Method (Other)	1,490.04	33,002.55	14,546.75	20,852.60

Table 4. Comparative operation performance

Table 4 shows how slower, in average, the different used micro-benchmarks were processed in the netbook and the smartphones compared to the notebook. The slowdown of an algorithm mostly depends on the necessary operations or how the algorithm is implemented, thereby using Android smartphones for scientific computing might require to slightly re-write the algorithms to minimize the associated slowdown.

Interestingly, the micro-benchmarks shows that the ViewPad 10s is faster than the Nexus One for Arith, Assign and Create, but the ViewPad 10s is slower than the Samsung I5500 for calling methods. However, the benchmarks are designed to evaluate the performance of a single-thread, single-process application and the ViewPad 10s has a dual-core processor. Therefore, it is expectable the ViewPad 10s to perform better when executing parallel applications.

Another result of the experiment is that different operations of the same kind have different behavior depending on where they are executed. For instance, adding two long values in the notebook is almost (takes 0.006% more time) as fast as adding two integers, but, in the Nexus One, adding two longs takes 33% more time than adding two integers. In consequence, badly choosing the data types of an application might not affect the performance in some platforms, but in others the impact can be very important and subject of serious consideration.

5 Conclusions and future work

In this paper, through a number of rigurous experiments, we have empirically shown that smartphones are able to execute complex computational tasks even

when their implementations have not been specifically adjusted to smartphone resource restrictions. Moreover, these experiments have also shown that, when smartphones are runing on battery, they can perform an interesting amount of computing intensive code before the depletion of their batteries takes place.

In addition to this, preliminary results beyond this paper point out that optimizing a code for the Android platform might better exploit smartphones' capabilities when performing scientific computations. Improving code efficiency not only decreases response time, but also might improve battery usage. Therefore, scientific source code optimization for the Android platform is a prominent research line.

In future research, we aim to study the performance of multi-thread and multi-process applications. This is of vital importance because current mobile device CPUs have multi-core capabilities. Hence, smartphone applications will have better performance levels when using parallel programming paradigms to exploit new CPUs.

Another research line consists in analyzing the performance of Android platforms when using native code. Although native code is popular in the scientific community and executes faster, it requires special cares to properly deploy it. For instance, native code usage can introduce memory leaks. The Android platform native language is C. Therefore, we are analyzing different benchmarks written in C to reproduce our findings when using native Android application codes.

Although Android smartphones might not be suitable for all scientific computational problems, this paper provides evidence about smartphones utility for solving computing intensive computational problems on the grounds of employing established scientific performance benchmarks. We think that if the problems are carefully chosen and implemented, Android smartphones can be considered as useful processing elements. For instance, Android smartphones would be helpful for solving iterative algorithms with preponderance of integer operations. On the other hand, Android smartphones would not be useful if the algorithms are recursive or require a lot of double operations. All in all, as a result of their popularity, Android smartphones might actually have a positive impact on contributing with resources to large distributed systems, such as Grids. In this sense, we have developed a battery-aware task scheduler that is able to submit individual tasks to many smartphones for execution. At present, we have evaluated the scheduler through simulation, but an Android implementation is underway.

Bibliography

[1] Y. Anokwa, C. Hartung, W. Brunette, G. Borriello, and A. Lerer. Open source data collection in the developing world. *Computer*, 42(10):97–99, 2009.

[2] S. Blom, M. Book, V. Gruhn, R. Hrushchak, and A. Köhler. Write once, run anywhere - a survey of mobile runtime environments. In *International Conference on Grid and Pervasive Computing*, pages 132–137, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[3] J. Bull, L. Smith, M. Westhead, D. Henty, and R. Davey. A benchmark suite for high performance Java. *Concurrency: Practice and Experience*, 12:375–388, 2000.

[4] J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK benchmark: Past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[5] P. Ghosh and S. Das. Mobility-aware cost-efficient job scheduling for single-class grid jobs in a generic mobile grid architecture. *Future Generation Computer Systems*, 26:1356–1367, 2010.

[6] F. González-Castaño, J. Vales-Alonso, M. Livny, E. Costa-Montenegro, and L. Anido-Rifón. Condor grid computing from mobile handheld devices. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(1):117–126, 2003.

[7] D. Huynh, D. Knezevic, J. Peterson, and A. Patera. High-fidelity real-time simulation on deployed platforms. *Computers & Fluids*, 43(1):74–81, 2011.

[8] K. Kumar and Y.-H. Lu. Cloud Computing for mobile users: Can offloading computation save energy? *Computer*, 43:51–56, 2010.

[9] C. Li and L. Li. Energy constrained resource allocation optimization for mobile grids. *Journal of Parallel and Distributed Computing*, 70(3):245–258, 2010.

[10] D. Murray, E. Yoneki, J. Crowcroft, and S. Hand. The case for crowd computing. In *2nd ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds*, pages 39–44, New York, NY, USA, 2010. ACM Press.

[11] J. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–27, 2005.

[12] A. Rice and S. Hay. Measuring mobile phone energy consumption for 802.11 wireless networking. *Pervasive and Mobile Computing*, 6(6):593–606, 2010.

[13] J. M. Rodriguez, A. Zunino, and M. Campo. Mobile grid SEAS: Simple Energy-Aware Scheduler. In *3rd High-Performance Computing Symposium - 39th JAIIO*, 2010.

[14] J. M. Rodriguez, A. Zunino, and M. Campo. Introducing mobile devices into Grid systems: A survey. *International Journal of Web and Grid Services*, 7(1):1–40, 2011.

[15] C.-L. Tsai, H.-W. Chen, J.-L. Huang, and C.-L. Hu. Transmission reduction between mobile phone applications and restful apis. In *2011 ACM Symposium on Applied Computing*, pages 445–450, New York, NY, USA, 2011. ACM.