Power-awareness and smart-resource management in embedded computing systems [*]

M. D. Santambrogio
Politecnico di Milano
DEIB
Milano, 20133, Italy
marco.santambrogio@polimi.it

José L. Ayala
Universidad Complutense de Madrid
DACYA
Madrid 28040, Spain
jayala@ucm.es

Simone Campanoni
Northwestern University
EECS
Evanston, IL, USA
campanoni@eecs.northwestern.edu

ABSTRACT

Resources such as quantities of transistors and memory, the level of integration and the speed of components have increased dramatically over the years. Even though the technologies have improved, we continue to apply outdated approaches to our use of these resources. Key computer science abstractions have not changed since the 1960's. Therefore this is the time for a fresh approach to the way systems are designed and used.

1. INTRODUCTION

Recently, multicore processors have become prevalent in the whole spectrum of computing systems, ranging from embedded solutions like mobile handheld devices to warehouse-scale systems like Google's datacenters. The growing amount of on-chip processing elements and their heterogeneity, coupled with the increasing number of concurrent applications, exponentially inflates the design-space of efficient computing systems calling for innovative performance- and power-aware resource management techniques. A deep redefinition of the hardware/software stack is needed to manage the fundamental trade-offs between maximizing performance under a power cap. Computer architectures could leverage reconfigurable fabrics to dynamically specialize and support the performance/power requirements of fluctuating loads; compilers and runtimes should automatically tune code generation to better exploit the underlying computer architecture to reach the sweet-spot in terms of performance per Watt; operating systems should implement smart resource management techniques leveraging the dynamic knobs provided by both computer architectures (e.g., dynamic voltage and frequency scaling) and compilers (e.g., degree of parallelism, accuracy of the resulting results).

As commodity processors acquire increasing numbers of cores, program performance depends more and more on creating or discovering thread-level parallelism. Because developing a multi-threaded program is hard, and because mainstream compilers lack effective parallelization strategies, the processing power of multicore processors is often wasted. Section 2 presents HELIX, a parallelizing compiler that extracts threads automatically, even for sequentially-designed, irregular programs. The parallel code that it generates can adapt at run time to make use of available cores. Moreover, when the output of a parallelized program only needs to approximate that of the sequential original, HELIX enables the user to control the trade-off between performance, energy consumed, and acceptable blurring of results.

Moreover, the actual power consumption of digital systems is strongly affected by their current *working state*, which is often a consequence of the external interactions they are subject to. As their power consumption is poorly described as the mere sum of contributions of the components they are composed of, it is difficult to make accurate predictions on the power consumed by the whole system over time, when it is subject to constantly changing operating conditions: this makes the definition of energy saving policies not trivial in most of real world scenarios. Because of this, Section 3 presents an holistic power modeling framework that can be used to profile the energy consumption of a wide range of energy-constrained systems: the same high-level workflow is

[*] Additional Authors:
R. Cattaneo, Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, *riccardo.cattaneo@polimi.it*
G. C. Durelli, Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, *gianlucacarlo.durelli@polimi.it*
M. Ferroni, Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, *matteo.ferroni@polimi.it*
A. Nacci, Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, *alessandro.nacci@polimi.it*
J. Pagán, DACYA, Universidad Complutense de Madrid, and CCS - Center for Computational Simulation (UPM), *jpagan@ucm.es*
M. Zapater, DACYA, Universidad Complutense de Madrid, and CCS - Center for Computational Simulation (UPM), *marina.zapater@ucm.es*
M. Vallejo, Automation Group of the UNAL, Dpt. of Electric Energy, National University of Colombia, *mavallejov*@unal.edu.co

978-1-4673-8321-9/15/$31.00 ©2015 IEEE

tailored on the actual system's features, extracting a specific power model able to describe and predict the future energy behavior of the observed entity.

Finally, in Section 4 the energy-optimization techniques that can be applied to optimize an e-Health application in an MCC scenario is presented. In wireless body sensor networks (WBSNs), the human body has an important effect on the performance of the communication due to the temporal variations caused and the attenuation and fluctuation of the path loss. This fact suggests that the transmission power must adapt to the current state of the link in a way that it ensures a balance between energy consumption and packet loss. On the other hand, the Mobile Cloud Computing (MCC) Scenario imposes a huge amount of acquired data that sometimes imply the usage of data centers to process such information. The proposed techniques cover the provision of transmission power level policies (reactive and predictive approaches) to minimize the energy consumption of the wireless monitoring nodes, as well as a workload assignment technique, based on heterogeneity and application-awareness, that redistributes low- demand computational tasks from high-performance facilities to idle nodes with low and medium resources in the WSN infrastructure.

2. HELIX-UP

Program performance in today's multicore processors mainly depends on the amount of thread level parallelism (TLP) available in a program. TLP is also important for obtaining energy efficiency because idle cores of today's multicore processors still consume power, and it is more energy efficient to rely on TLP rather than Instruction Level Parallelism (ILP) exploited by a single core to gain performance.

Some computing problems often translate to either inherently parallel or easy-to-parallelize numerical programs. However, sequentially designed non-numerical programs, which are hard to parallelize because of their complicated control and data flow, are much more common. Non-numerical programs offer no TLP when traditional compilers are used; this leads to low performance and low energy efficiency in today's multicore processors. To obtain high performance per watt in today's systems, we need to enable a high amount of TLP even for non-numerical programs that are notoriously difficult to parallelize either manually or automatically.

Contrary to common assumptions, there is considerable latent TLP even in non-numerical sequentially designed programs, e.g., between the iterations of a loop [5, 3, 4, 20]. When such iterations can run unfettered on multiple cores in a modern processor, performance adjusts with the number of cores. Unfortunately, loop iterations often depend upon one another and require strict ordering; thus, compilers that strictly adhere to program semantics generate slow sequential code to guarantee correct results. For applications that can tolerate bounded distortion of results, however, there is an exciting opportunity to ignore some dependences and liberate parallelism. We propose extracting TLP from programs (including non-numerical) by relaxing sequential consistency constraints in a disciplined manner, doing so only when it increases performance and energy efficiency while incurring little or no output distortion.

Our work is based on the empirical observation that not all dependences are created equal. Some will slow execution more than others. Other dependences have little to no impact on the outputs of interest. Lastly, these characteristics change depending on inputs and at run time. Therefore, the challenge lies in identifying these dependences that bottleneck parallel performance, understanding impact (if any) on the outputs by relaxing sequential requirements and relaxing them at runtime, all guided by performance goals and tolerable limits set by the program's user (not the programmer). To this end, we implemented the *unleashed parallelizer* (HELIX-UP), a co-design of profilers, a loop-parallelizing compiler, and a runtime system. The HELIX-UP compiler, built on top of the HELIX parallelizing compiler [3], selectively relaxes strict adherence to language semantics to increase parallel scalability at run time. HELIX-UP's user provides a sequential program, representative training inputs, and a function that measures distortion of the program's outputs. The profiler uses this information to assess how much the program semantics can be relaxed and the impact of doing so. Finally, the runtime system applies the relaxations judiciously and automatically based on the user's request.

2.1 The HELIX Parallelizing Compiler

HELIX is a conservative parallelizing compiler that automatically generates parallel threads from a sequentially-expressed program, and allows parallel loop iterations to run concurrently on separate cores in a commodity multicore processor. This approach resembles traditional DOACROSS parallelism [9]. To satisfy data dependences between loop iterations (i.e., *loop-carried dependences*), HELIX identifies segments of a loop's body—called *sequential segments*—that must execute in iteration order on the separate cores. Synchronization operations that surround each sequential segment guarantee loop-iteration order. HELIX also guarantees that different sequential segments are independent so that dynamic instances run in parallel to gain more performance [4].

Three main sources of overhead hinder performance improvements in traditional parallelizing compilers like HELIX that strictly adhere to program semantics. First, strict observance of a program's original sequential order leads to bottlenecks resulting from a long-running iteration that stalls all other cores. Second, slow communication between cores amplifies the cost of inter-core data sharing and communication required to satisfy dependences. Lastly, distribution of shared data across multiple cores degrades locality. By carefully relaxing otherwise strict adherence to program semantics, HELIX-UP alleviates all three of the aforementioned overheads with little to no impact on output correctness.

2.2 The HELIX-UP Solution

Figure 1 provides an overview of HELIX-UP, which builds on the HELIX compiler to include semantics-relaxing code transformations in addition to the conventional

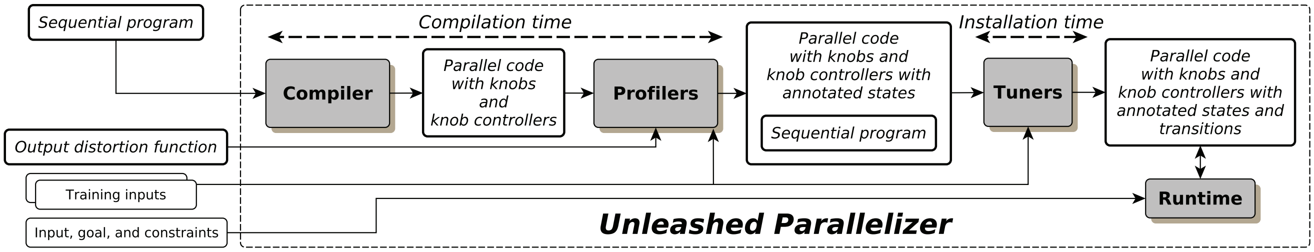

Sequential program
Compilation time
Installation time
Compiler
Parallel code with knobs and knob controllers
Profilers
Parallel code with knobs and knob controllers with annotated states
Sequential program
Tuners
Parallel code with knobs and knob controllers with annotated states and transitions
Output distortion function
Training inputs
Input, goal, and constraints
Runtime
Unleashed Parallelizer

Figure 1: User provides a sequential program to parallelize, a function to measure program's output distortions, representative inputs, and goal and constraints. The compiler parallelizes the code, including in it knobs and knob controllers. The profilers annotate knob controllers by using the training inputs. The tuners customize knob controllers for the target platform. While the program runs with the reference input, the runtime interprets knob controllers, adjusting knobs to alleviate performance bottlenecks, while respecting the constraints and goal set.

semantics-preserving ones [6]. Information inferred by the HELIX-UP compiler through traditional code analyses (e.g., data dependence analysis or induction variable analysis) is coupled with information collected by HELIX-UP profilers. Each profiler identifies dependences that have little or no impact on workload outputs when left unsatisfied. The HELIX-UP runtime uses this information while monitoring parallel execution. When the observed execution does not meet expectations (e.g., performance is too low), the runtime selectively relaxes program order based on profiler data to boost performance.

The HELIX-UP knobs. To control relaxation of program order performed by semantics-relaxing code transformations, HELIX-UP relies on a set of *knobs*, which are defined by the compiler, calibrated off line by profilers, and then used by the HELIX-UP runtime. Each knob corresponds to a potential source of parallelism-dampening overhead for a parallelized loop. For example, a knob may describe how often a dependence between instructions is satisfied, or whether the order of loop iterations is being maintained. The setting of a knob starts from the most conservative configuration (e.g., program order is completely preserved, with some loss of performance) to the most aggressive (e.g., program order is totally sacrificed to maximize performance).

Setting a knob might affect the impact of other knobs, so the design space created by knob configurations grows exponentially with the number of knobs. To limit this space, we empirically observed that a knob that changes the code of a given loop has no effect (as a first approximation) on knobs included in other loops. Exploiting this empirical observation, we perform an exponential reduction of the knob space.

All knob configurations of the reduced knob space are characterized in terms of performance, energy, and output distortion. These metrics are measured after whole program executions using training inputs.

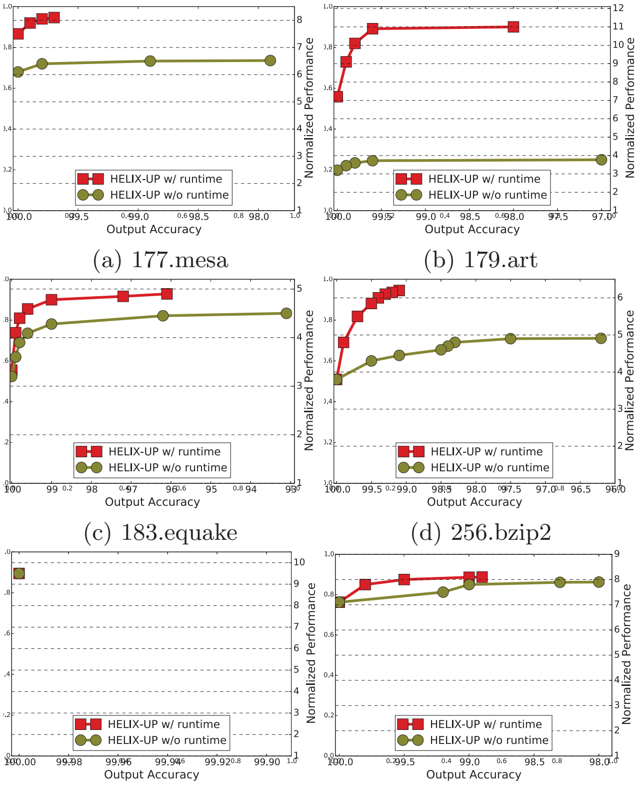
Assumptions and limitations. HELIX-UP assumes that the training inputs yield representative performance, energy, and output distortion. If that assumption is incorrect, HELIX-UP does not guarantee restriction of output distortion or avoidance of unrecoverable faults. In that case, the HELIX-UP user risks unexpected output distortion while gaining substantial performance or energy benefits. To overcome this problem, other systems rely on calibration periods at run time [24].

2.3 Evaluation

HELIX-UP makes the performance of parallelized code robust and scalable while limiting output distortion. We now demonstrate the resulting benefits on today's commodity platforms, and we show that each component of HELIX-UP is important to its success.

Experimental setup. Using sequentially-designed benchmarks, ranging from those considered difficult to parallelize (SPEC CPU suite) to more regular ones (PARSEC), we evaluated HELIX-UP on today's multicore processors. We label the platforms with their microarchitecture names. The first, *Nehalem*, includes six Intel® Core™ i7-980X cores, each operating at 3.33 GHz. The second, *Haswell*, includes four Intel® Core™ i7-4770K cores, each operating at 3.5 GHz. We disabled Turbo Boost for both platforms. Each has three cache levels. L1 (32KB) and L2 (256KB) are private to each core. With HyperThreading enabled, two threads can run per core, sharing the first two cache levels. In both Nehalem and Haswell, the last level cache (12MB and 8MB, respectively) is shared by all cores.

We used the second version of the HELIX compiler, HCCv2 [5], which is based on the ILDJIT compilation framework [2]. The sequential programs used as baseline were the unmodified versions of benchmarks, optimized (O3) and compiled by ILDJIT with LLVM 3.4.1 as the back end.



(a) 177.mesa (b) 179.art

(c) 183.equake (d) 256.bzip2

(e) blackscholes (f) swaptions

Figure 3: A runtime is needed to choose the best performance-accuracy settings. HELIX-UP is called UP in these figures. Platform used: Nehalem

Both semantics-relaxing and semantics-preserving code transformations are needed. Figure 2 evaluates HELIX-UP on the Nehalem platform when either 0% or 10% output distortion is acceptable. The important difference between HELIX-UP and when semantics-relaxing or semantics-preserving transformations are disabled suggests that using semantics-relaxing code transformations to *enhance* semantics-preserving ones is better than replacing them. These results also suggest that conventional parallelizing compilers like HELIX are severely limited because they include only semantics-preserving transformations.

Parallelized code needs to be dynamically adjusted. Figure 3 plots the trade-off between performance and output accuracy for HELIX-UP with and without a runtime.[1] To generate this data, we swept the constraints given as inputs to HELIX-UP while keeping performance as the goal. The difference between the two scenarios shown in Figure 3 highlights the value of sparingly sidestepping strict semantics only when semantics-preserving optimizations fall short. In other words, there is great value in dynamically adjusting the parallelized code.

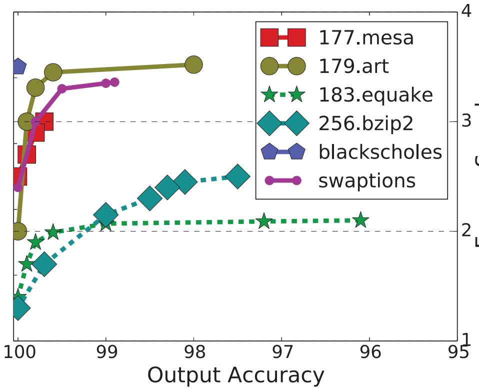


Figure 4: Energy-accuracy trade-off of HELIX-UP. Platform used: Haswell

HELIX-UP provides an energy efficient solution. Figure 4 shows the trade-off provided by HELIX-UP between energy and output accuracy on the Haswell platform. This platform is the only one of our experimental processors with the RAPL hardware performance counters which were used to compute energy consumption.

Note that without semantics-relaxing transformations, a parallelizing compiler like HELIX cannot provide better energy efficiency than the results obtained by HELIX-UP with 100% output accuracy shown in Figure 4.

2.4 Conclusion

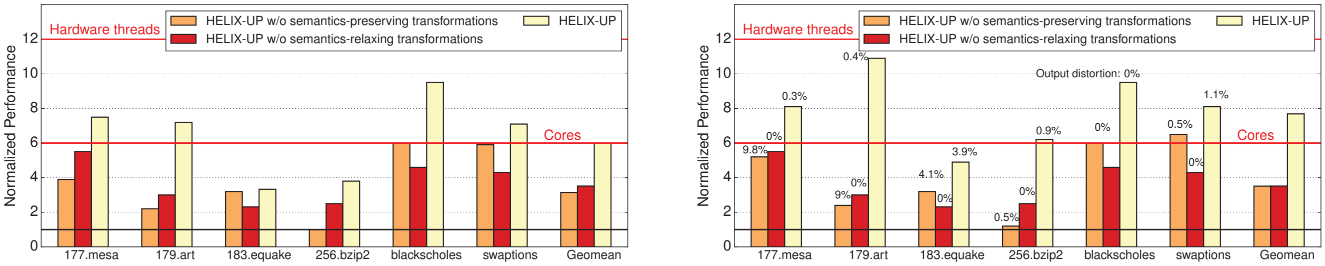
Thread Level Parallelism (TLP) is necessary for both performance and energy efficiency for today's multicore processors. HELIX-UP automatically extracts TLP even from those programs notorious for being difficult to efficiently parallelize either manually or automatically. To obtain TLP, HELIX-UP enhances a parallelizing compiler with semantics-relaxing code transformations and controls them at run time. This provides a trade-off between performance (or energy) and the output distortion generated by these added code transformations.

3. HOLISTIC POWER MODELS AND MANAGEMENT

Power consumption has become a major concern for almost every digital system: from the smallest embedded circuit to the biggest computer cluster, an energy budget is always constraining the performance of the system. Moreover, the actual power consumption of these systems is strongly affected by their current *working state*(e.g., from idle to heavy-workload conditions, with all the shades in between), which is often a consequence of the external interactions they are subject to. As their power consumption is poorly described as the mere sum of contributions of the components they are composed of, it is difficult to make accurate predictions on the power consumed by the whole system over time, when it is subject to constantly changing operating conditions: this makes the definition of energy saving policies not trivial in most of real world scenarios.

3.1 An holistic power modelling approach

[1] Accuracy is computed as 100 minus the output distortion.



(a) No output distortion

(b) Output distortion ≤ 10%

Figure 2: Both semantics-relaxing and semantics-preserving transformations are needed to extract TLP.

We not want to show an holistic power modelling tool that can be used to profile the energy consumption of a wide range of energy-constrained systems: the same high-level workflow is tailored on the actual system's features, extracting a specific power model able to describe and predict the future energy behavior of the observed entity. More specifically, we observed that the energy behavior of a computing system is generally piecewise linear. On the one hand, there exist some variables of the systems that determine a strong difference in its power consumption (i.e., a significantly different slope of the energy trace over time): these variables determine the *configuration* of the system in a certain instant and are often related to the current *working state* of its internal hardware components. These variables are often controllable, and these are the *knobs* we can use to tweak the system's power consumption. On the other hand, other variables can not be controlled (e.g., user-defined workload conditions) and have to threaten as an *exogenous input* of the modeled system. This methodology is provided in an *as-a-service* fashion: at first, the target system is instrumented to collect power metrics and workload statistics in its real usage context; then, the collected measurements are sent to a remote server, where data is processed using well known techniques (e.g., Principal Components Analysis, Markov Decision Chains, AutoRegressive statistical models, etc.); finally, an accurate power model is built as a function of the metrics monitored on the instrumented system, which will then be able to take advantage of these information to optimize its behavior with respect to its own performance-per-watt goals, being that on an hourly or a daily basis.

3.1.1 The MPower case study

We validated the aforementioned approach in a mobile device scenario: we developed an Android application, codename MPower (available on the Google Play Store), to collect data about smartphones and tablet in their real usage scenarios [11, 17]. The system produced power models for more than one thousand of devices, showing how the proposed methodology is able to perform better than the one implemented by Google Historian for recent releases of the Android operating system, thus allowing the definition of more effective power management policies able to save as much power as possible with respect to the user's goals. The proposed methodology observes the device's behavior for a certain period of time in its real operating context, in order to learn how its battery lasts in different conditions. This knowledge is then used to make accurate predictions of the Time-To-Live (TTL) of a generic mobile device in its real usage context. The estimated TTL is provided in terms of minutes (and not just in term of remaining battery percentage) and it gives an important information to the user, because it allows him/her to choose how to use his/her device, with respect to his/her current needs. These predictions are based on data directly retrieved from devices without any dedicated hardware tool, laboratory measurements or modification of the operating system: as a consequence, the adopted modeling system may apply to the burden of devices currently available on the market, with the only constraint of having a set of APIs to easily access measurements coming from available sensors. To validate the proposed methodology, we implemented a real system currently working on Android devices, since it is the most widespread open source mobile operating system, which provides the necessary set of APIs (an Android application is available for free on the Google Play Store [13]). Differently from the already existing approaches for hardware power modeling, our method starts from everyday usage data (all the data provided by the mobile app listed in Tab. (1)) coming from devices in real context [11], to achieve a high level of flexibility: consequently, it is able to adapt both to new devices and new OS versions.

3.1.2 From mobile to datacenters

We now want to show how it is possible to exploit the same methodology to account and profile power consumption of virtual machines in a multi-tenant server infrastructure. During our collaboration with University of California Berkeley, we instrumented the current prototype of the Tessellation operating system [8], based on the Xen Hypervisor: we monitored the processor's Hardware Performance Counters (HPC) to get per-core metrics of CPU utilization and the Running Average Power Limit (RAPL) interface to get per-socket energy measurements. In this scenario, data showed us that resource allocation and virtual machines placement strongly influence the system's power consumption, thus defining the *configuration* of the system. Then, HPC metrics, related to the instantaneous workload condition of a virtual container, can be threaten as an *exogenous input*. The holistic power modeling tool will then exploit this information to learn the power model of the system, enabling the development of an energy-aware scheduler: given a certain hourly or daily energy budget, the system will be able to estimate the best tradeoff between global performance and power consumption, still providing the required Quality-of-Service to all the guests applications towards an adaptive and power-aware multi-tenant server

Table 1: Data gathered from the device.

Screen	Battery	CPU(s)	Mobile	WiFi	Audio	Bluetooth	GPS
is on	on charge	max freq.	state	is on	music active	is on	is on
brightness mode	temperature	min freq.	activity	is connected	speaker on	state	status
brightness value	voltage	current freq.	net type	signal strength	music volume		
width	percentage	max scaling freq.	signal strength	link speed	ring volume		
height	technology	min scaling freq.	tx bytes				
refresh rate	health	governor	rx bytes				
orientation		usage	call state				
		cpu id	airplane mode				

infrastructure.

3.2 Power Model Estimation

Experimental setting Since the goal of the proposed system is the TTL prediction, we want to provide evidence on how the models computed by the system are effective in predicting the device TTL. We used the data collected within the remote server as the test bed for the prediction task. These data come from 188 devices which nowadays are continuously sending data (this set is composed of more than 30 different devices models, which are mainly Samsung, HTC and LG). We took into account 11638860 samples coming from 5 months long recording (spanning the period October, 15 2013 - March, 15 2014) with a minimum of 1443 and a maximum of 299205 samples per device). Around 75% of the data were used for training, while the remaining was saved for testing purposes.

We decided to use the linear time-invariant ARX model family. We selected the ARX orders by basing on the validation error and exploring models with $n_a = \{0, \dots, 3\}$ and $n_b = \{1, \dots, 5\}$, using as input variables $x(t)$ CPU frequency and screen brightness (25% of the training set was used for model selection). After this phase, batches of $M = 100$ samples (selected as described in Section **??**) were extracted and used to compute the parameter vectors α_cs. For each training batch, we computed the parameter vector through a *non-positive least square* algorithm[2], which guarantees to reach the least square solution under the constraint of having only negative parameters. We opted for a model with only negative coefficients since, within this context, if a variable increases its value we expect an higher discharge rate, e.g. the more the screen brightness is high, the more the battery level will drain. At last, in a real scenario, we do not have information about the future value of the uncontrollable inputs u, hence, we substitute, for prediction purpose, future inputs with their estimated average value.

To evaluate the performance of the proposed prediction methodology, we used the *absolute error* described in [26]:

$$M(t) = \beta|e(t)| + (1 - \beta)M(t - 1)$$

where $M(0) = 0$ and $e(t) = \hat{y}(t) - y(t)$ is the error at time t. For a fixed configuration c, we computed $M_c = M(360)$, i.e., the smoothed error made by our system after 1 hour of prediction. We used the value of $\beta = 0.1$, as suggested in [26]. Since the data used for estimation were limited compared to those needed for the estimation of the distribution of the coefficient vectors, we decided to apply a 10 fold cross-validation approach, considering the average of M_c for each configuration. The total error for each device is then computed by taking a weighted average:

$$M_{d_i} = w_1 M_{c1} + \ldots + w_{|C|} M_{c|C|}$$

where d_i is the device index and weights w_i are computed as the fraction of time the i-th configuration is reported on a given device.

Furthermore, we analyzed the predicted configuration models per device and the predicted TTL for 400 models, different in configuration and device.

Results Figure 5 shows the performance of our methodology in making a complete forecast on unseen data. In this example, when a configuration without a model is found, it is replaced by the real discharge behaviour for visualization purposes. The graph illustrate how our methodology is able to follow the real discharge of the device (line in green): clearly a better result is obtained by using real inputs (line in red) of the traces analyzed, while a still reasonable prediction is performed by the model with the chosen euristic for input reconstruction (line in green).

The averaged error M_d distribution for the considered devices is presented in Figure 6. In the proposed system, by considering the real input, the average absolute error is $\bar{M}_d = 0.036$, around 90% of the devices has $M_d \leq 0.065$ and the maximum error computed on a device is $M_{\max} = 0.87$, which allows us to infer that there is only a slight difference in the behavior of our system on different devices. Thus the methodology we presented is flexible w.r.t. the wide range of Android-based devices.

Another interesting statistics is that the average number of estimated configurations is 9, while the total amount of observed ones is 21. This justifies the effort of the presented methodology, which is able to compare devices and reuse estimated models.

At last, to underline the variability of the discharge curves predicted on different configurations, we computed the TTL of each estimated configuration, stating by a 100% charged battery. The results provided a TTL spanning from 2 to 22 hours: this high variability of the predicted TTL justifies

[2] This algorithm is a trivial modification of the non-negative least square presented in [14]

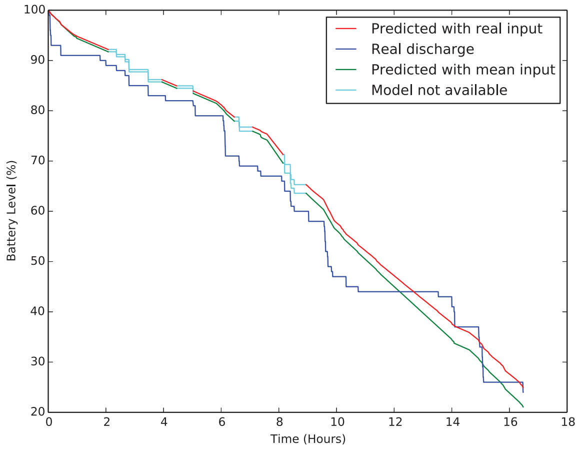


Figure 5: Example of a real discharge curve for a device, coupled with the one predicted by our methodology

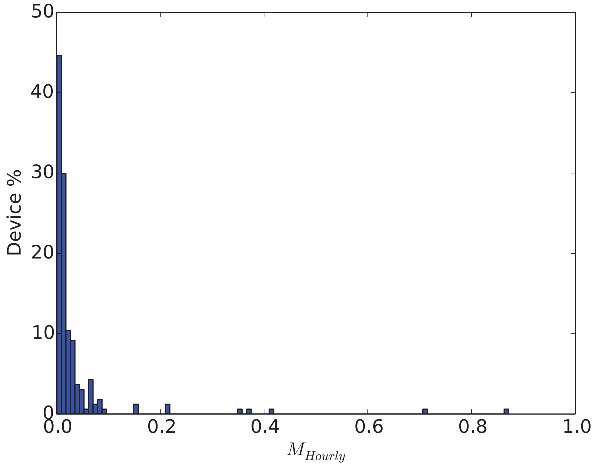


Figure 6: Distribution of M_d on the device population

the use of multiple models crafted for each configuration.

4. POWER TRANSMISSION AND WORK BALANCING POLICIES IN THE EHEALTH MOBILE CLOUD COMPUTING SCENARIO

The Internet of Things (IoT) holds big promises for healthcare, especially in proactive personal eHealth. To fulfill these promises major challenges must be achieved, especially in areas of non intrusive monitoring devices, predictive models and applications. These scenarios place a major concern in the amount of data to be processed. 24-hours monitoring studies generate a huge amount of data that require a high computing capability only available in state-of-the-art Data Centers. However, these facilities have a large energy consumption and strong impact on carbon footprint, leading to unsustainable electricity bills and environmental costs.

Following sections drive through a real case of prediction in the eHealth scenario, devoted to neurological disorders. The presented case study focuses on the migraine headache. Migraine is one of the most wide spread neurological disorders, with a higher prevalence in women than men. This mostly hereditary disease represents a cost in Europe of €1222 per patient per year [15]. A Wireless Body Sensor Network (WBSN) for non intrusive ambulatory monitorizations has been used to monitor four hemodynamic variables. These are: electrocardiogram (ECG) signal for heart rate (HR), surface skin temperature (TEMP), electrodermal activity (EDA) and peripheral capillary oxygen saturation (SpO2). Changes in these variables are regulated by the autonomic nervous system (ANS) and some clinical literature have related changes in some of these variables with migraines attacks [19, 22]. Recent studies demonstrated the capability of predicting migraines with these variables [21].

Data obtained using the WBSN is communicated to an embedded processing element, i.e. a coordinator (a PDA or smartphone), and sent to the Cloud. In order to predict migraines, huge data sets must be analyzed. To deal efficiently with such computationally intensive tasks, part of the processing and storage will be local, in the node, while another part will be communicated and processed in the Data Centers. This computing paradigm is generally called *Mobile Cloud Computing* (MCC) [12]. As the target population is large, the key challenge in this scenario is how computation can be off-loaded and distributed efficiently to Data Centers. However, Data Centers consume a huge amount of power and generate a tremendous amount of heat.

The aim of this section is to present a workload balancing proposal in order to optimize the energy consumption in each point of the monitoring and prediction framework (from the WBSN to the Data Center). Three energy consumption scenarios are presented, according to the amount of energy used for gathering data, pre-processing them and make the predictions. The first scenario represents a dummy situation where data is gathered and transmitted in streaming. The second scenario applies energy aware off-loading techniques, and the third one applies Data Center energy minimization policies on top of the previous off-loading techniques.

4.1 Experimental set up

In order to monitor and predict migraines, each patient needs to carry two sensor nodes wirelessly connected to a coordinator node, which on its turn transmits data to the Data Center via a 3G link. For our experiments, we consider a target population of 10000 patients being monitored at the same time.

Two phases are considered in this framework of migraine prediction: i) model training and ii) real-time prediction. During the training phase, data are gathered and sent to the Data Center. In the Data Center, data is pre-processed and models are created and validated. Training is made offline. In addition to gathering hemodynamic variables, patients mark in the smartphone their subjective pain. With these points a normalized two semi-gaussian symptomatic curve is generated. During real-time, the generated models are applied in runtime to the input data. Real-time prediction can be performed in the Data Center or in the coordinator node. Data can be pre-processed in the sensing nodes, or in the coordinator, or even in the Data Center. All possible combinations will be studied in a future work. In this paper two extreme situations are considered: i) executing all the processing in the Data Center or, ii) using the WBSN and coordinator (PDAs, smartphones) to off-load computation, under certain constraints, and minimize overall energy consumption.

The N4SID state-space algorithm has been used for migraine prediction. Training and validation codes have been developed using the System Identification Toolbox of the MATLAB software; however, the compiled version (from Matlab Compiler for standalone applications) has been computed in servers, improving the speed-up. N4SID matrices and states have been implemented in C-code for servers and the coordinator node.

4.1.1 The WBSN

The WBSN is composed of three nodes in a star topology. Two of them are peripheral sensor nodes placed in the right arm (Node 1, link L1) and in the right knee (Node 2, link L2). These are connected to the third one, the hub or coordinator node (a smartphone with radio interface[3]). The two sensor nodes sense the four aforementioned hemodynamics variables. The coordinator collects data from sensor nodes and forwards them to the Data Center as needed.

The sensor nodes are well known Shimmer [25] devices. Shimmer devices are based on the MSP430F1611 16-bit microcontroller. These microcontrollers use a 10 kB RAM and 48 kB flash memory at a maximum frequency of 8 MHz. The CC2420 chip radio performs the radio interface, implementing the IEEE 802.15.4 radio standard. Node 1 senses ECG signal on the chest and EDA in the arm with electrodes, and surface skin temperature near the armpit with a NTC thermistor. Node 2 senses SpO2 and photoplethysmography (not used in this work) in the capillarity zone near the groin, through the NONIN devices (8000R sensor and the OEM-III board [18]). The data acquisition sampling frequencies are 250 Hz for ECG and 0.2 Hz for TEMP and EDA, whereas the maximum data allowed by the OEM-III device is 3 kbps. Shimmer uses the operating system for embedded devices TinyOS; nevertheless, the FreeRTOS has been used instead, being more energy efficient due to the most recent development.

We consider the sensor nodes working in two different scenarios listed below:

- Scenario 1[4]: Node 1 samples ECG and transmits the data immediately (streaming). Node 2 reads data from NONIN device every second and transmits the data immediately (streaming).
- Scenario 2: Node 1 computes the HR (sampling + processing) and transmits it. Node 2 reads data, stores it for a while and transmits the data (burst mode). Data from Node 2 does not need conversion.

Table 2: Data transmission properties for sensors

	Scenario 1		Scenario 2	
	Node 1	Node 2	Node 1	Node 2
TX_r (ms)	277	1000	60000	20000
D (Bytes)	128	471	30	9252
TX_t (ms)	4	15	0.1	296
Duty cycle (%)	1.42	1.56	0.0016	1.56

For the 802.15.4 radio standard, a 104 Bytes data payload has been used, and 128 Bytes correspond to 1 transmission packet (including headers). The radio data rate is 250 kbps. Table 2 summarizes the transmission rate (TX_r), the amount of data sent (D, including headers in packets), the transmission time cost (TX_t), and the duty cycle (transmission time related with the transmission rate) for nodes 1 and 2 in each scenario.

The energy consumption of the microcontroller and external devices has been measured using a high precision digital amperimeter, whereas, the consumption of the radio has been simulated, due to the complexity in measurement of energy consumption in the real scenario. Simulation has been carried out with the open source simulator Castalia [7], which is designed specifically for WBSNs and that includes a channel model based on data measured empirically, and a model of radio CC2420, used for our experiments with the patient. Also, the simulation capabilities provided by Castalia have been extended to include the path loss values calculated from RSSI measurements obtained with the sensor nodes in the experimental stage. It is assumed for our experiments that radio switches on for every transmission, going off after finishing. Simulations for this work have been executed at the maximum power available in the radio CC2420 chip, 0 dBm. A retransmission rate has been simulated as random retransmissions along the time, simulating the channel effects (data collisions, medium accesses, etc.), using a 802.15.4 MAC with two transmission attempts and temporal variation model to recreate the dynamics of the path-loss.

The coordinator node is an smartphone. To simplify energy characterization, a BeagleBone Black platform [1] has been used, as it uses the same processor as the Samsung Galaxy S smartphone. The processor is an ARM Cortex-A8 at 1 GHz with 512 MB DDR3 RAM. A 2 GB SD memory card has been used to store data. As radio device, the same CC2420 as in the Shimmer devices is supposed to be used in the platform or smartphone. We plan a case study to apply energy-aware off-loading policies that balance computation

[3] For simulation, the coordinator is supposed to be placed in the waist (just over the navel).

[4] TEMP and EDA data are considered negligible for energy consumption calculation

between the computing elements and the Data Center. In this sense, the coordinator node may perform two actions: i) data pre-processing, in case it was not executed in the Shimmer nodes, or ii) run-time testing of the algorithms to off-load computation from the Data Center.

4.1.2 The Data Center

The Data Center setup comprises two clusters: i) a High-Performance Computing (HPC) cluster to train and validate the models, as this are CPU and memory intensive tasks, and ii) a virtualized Cloud cluster for model testing. The HPC cluster is composed of Quad-core Intel Xeon RX300 servers with 16GB of RAM, and the Cloud cluster consists on Intel SandyBridge Decathlete servers with six cores and 32 GB of RAM.

We characterize in terms of power and performance the training and validation tasks by taking real measurements in these servers. The training and validation phase of 1 patient runs for approximately 3.5 hours. Training and validation for the 10000 patients is performed in a simulated HPC SLURM cluster [16] consisting on 20 servers. We consider that, in average, models need to be re-trained once per month per patient.

Run-time prediction time is computed every 1 minute. The time horizon of prediction is always 30 minutes forward, i.e., a migraine is predicted 30 minutes in advance. Run-time prediction is a light-weight process that can be computed at the coordinator node or at the Data Center. If computed at the Data Center, a cluster of 10 servers virtualized using OpenStack over KVM is used. Several testing instances are packed together in the same virtual machine, until a high utilization is reached. When this happens, OpenStack automatically scales up resources, launching as many virtual machines as needed to execute the workload.

We assume all 30 servers are placed in an air-cooled data room equipped with a Daikin FTXS30 unit [10], with a nominal cooling capacity of 8.8kW and a nominal power consumption of 2.8kW.

4.2 Results

Results have been calculated for a young, middle-sized and middleweight female migraine patient without treatment. 15 migraines have been used for training the models. Energy results have been generalized for a population of 10000 patients. The microcontroller energy consumption for Node 1 in both scenarios has been obtained from [23]. The consumption in Node 2 has been measured with a HAME HM8012 digital multimeter. Radio consumption has been calculated in simulation with Castalia for both of the nodes.

Table 3: Energy consumption for Shimmer nodes

	Node 1 (mJ)			Node 2 (mJ)		
	uC	Radio	Total	uC + NONIN	Radio	Total
Scenario 1	396	13526	13922	3635	3600	7235
Scenario 2	430	9	439	3635	74	3709

Table 3 shows the energy consumption for the Shimmer nodes (microcontroller and radio for Node 1, and microcontroller, 8000R oximetry sensor, OEM-III board and radio for Node 2), calculated for the execution time of 1 minute and only for 1 patient. As previously commented, radio switches on and switches off in every transmission, what explains the high consumption of Node 1 in Scenario 1. In Scenario 2, Node 1 switches on just one time per minute, while in Scenario 1 it does for 217 times, (see Table 2). A similar situation occurs with Node 2; in Scenario 1 radio switches on 20 times more than in Scenario 2. Energy savings for computing HR in Node 1 reaches the 3171% and 195% in Node 2 due to delaying transmissions.

Table 4 shows the energy consumption for the overall application framework of migraine monitoring and prediction under three extreme scenarios, during 1 week of execution and 10000 patients:

- Scenario A: Shimmer nodes streaming data (Scenario 1), coordinator node only executing data pre-processing, all training performed in the HPC cluster, and all testing in the virtualized cluster.
- Scenario B: Shimmer nodes pre-processing data (Scenario 2), coordinator nodes executing testing 70% of the time (when battery power allows), all training performed in the HPC cluster.
- Scenario C: same off-loading techniques than in Scenario B, but we also use virtual machine consolidation, turning off unused servers in the Cloud cluster, and applying cooling control techniques.

Results are aggregated per device, i.e. each column shows the aggregated energy value for all 10000 patients under test. For instance, for Scenario A, 20000 sensors (10000 ECG and 10000 SpO2) consume 592kWh, and 10000 coordinator nodes consume 1262kWh. As can be seen, in an scenario with so large number of sensors and coordinators, the impact of these devices on the overall energy consumption of the application is large. In this sense, reducing the energy consumed by the sensor devices implies significant savings. Data Centers are an important contributor to overall energy, as only 30 servers consume almost 40% of the overall energy for the application.

Table 4: Energy consumption for various workload off-loading scenarios and 1 week (kWh)

	Sensors	Coordinators	DC IT	DC Cool.	Total
Scenario A	592	1262	579	231	2664
Scenario B	116	1264	550	220	2150
Scenario C	116	1264	464	197	2041

For space reasons, only three scenarios are tested and limited energy minimization policies are presented. The policies applied yield to savings of 23.4% in the overall application. All possible combinations and further energy-aware policies, including SLURM allocation optimization, and improved VM consolidation will be applied in future work, following the methodology in [27].

Acknowledges

The *Power Transmission and Work Balancing Policies in the eHealth Mobile Cloud Computing Scenario* research work has been partially funded by the Spanish Ministry of Economy and Competitivity under Research Grant TEC2012-33892

References

[1] BeagleBone. Beaglebone black. http://beagleboard.org/BLACK. Accessed: 2015-07-1.

[2] S. Campanoni et al. A highly flexible, parallel virtual machine: Design and experience of ILDJIT. *SPE*, 2010.

[3] S. Campanoni et al. HELIX: Automatic parallelization of irregular programs for chip multiprocessing. In *CGO*, 2012.

[4] S. Campanoni et al. HELIX: Making the extraction of thread-level parallelism mainstream. In *IEEE Micro*, 2012.

[5] S. Campanoni et al. HELIX-RC: An architecture-compiler co-design for automatic parallelization of irregular programs. In *ISCA*, 2014.

[6] Simone Campanoni et al. HELIX-UP: Relaxing program semantics to unleash parallelization. In *CGO*, 2015.

[7] Castalia. Castalia wireless sensor simulator. https://castalia.forge.nicta.com.au/index.php/en/. Accessed: 2015-07-1.

[8] J.A. Colmenares, G. Eads, S. Hofmeyr, S. Bird, M. Moreto, D. Chou, B. Gluzman, E. Roman, D.B. Bartolini, N. Mor, K. Asanovic, and J.D. Kubiatowicz. Tessellation: Refactoring the os around explicit resource containers with continuous adaptation. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–10, May 2013.

[9] R. Cytron. DOACROSS: Beyond vectorization for multiprocessors. *ICPP*, 1986.

[10] Daikin AC (Americas), Inc. Engineering Data SPLIT, FTXS-L Series. http://www.daikinac.com/content/resources/manuals/, 2010.

[11] Matteo Ferroni, Andrea Cazzola, Domenico Matteo, Alessandro Nacci, , Donatella Sciuto, and Marco D Santambrogio. Mpower: Gain back your android battery life! In *International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, September 2013.

[12] Abdul Nasir Khan, ML Mat Kiah, Samee U Khan, and Sajjad A Madani. Towards secure mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(5):1278–1299, 2013.

[13] NECST Laboratory. Mpower v2.0. https://play.google.com/store/apps/details?id=org.morphone.mpower, december 2013.

[14] Charles L Lawson and Richard J Hanson. *Solving least squares problems*, volume 161. SIAM, 1974.

[15] M. Linde et al. The cost of headache disorders in europe: the eurolight project. *European journal of neurology*, 19(5):703–711, 2012.

[16] Alejandro Lucero. Simulation of batch scheduling using real production-ready software tools. http://www.bsc.es/media/4856.pdf.

[17] A. A. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto, and M. D. Santambrogio. Adaptive and flexible smartphone power modeling. *Mob. Netw. Appl.*, 18(5):600–609, October 2013.

[18] NONIN. Nonin devices. http://www.nonin.com/Home. Accessed: 2015-07-1.

[19] Carlos M Ordás, María L Cuadrado, Ana B Rodríguez-Cambrón, Javier Casas-Limón, Náyade del Prado, and Jesús Porta-Etessam. Increase in body temperature during migraine attacks. *Pain Medicine*, 14(8):1260–1264, 2013.

[20] G. Ottoni et al. Automatic thread extraction with decoupled software pipelining. *MICRO*, 2005.

[21] Josué Pagán, M. Irene De Orbe, Ana Gago, Mónica Sobrado, José L. Risco-Martín, J. Vivancos Mora, José M. Moya, and José L. Ayala. Robust and accurate modeling approaches for migraine per-patient prediction from ambulatory data. *Sensors*, 15(7):15419, 2015.

[22] Jesús Porta-Etessam, María L Cuadrado, Octavio Rodríguez-Gómez, Cristina Valencia, and Sara García-Ptacek. Hypothermia during migraine attacks. *Cephalalgia*, 30(11):1406–1407, 2010.

[23] Francisco Rincón, Joaquin Recas, Nadia Khaled, and David Atienza. Development and evaluation of multilead wavelet-based ecg delineation algorithms for embedded wireless sensor nodes. *Information Technology in Biomedicine, IEEE Transactions on*, 15(6):854–863, 2011.

[24] M. Samadi et al. SAGE: Self-tuning approximation for graphics engines. In *MICRO*, 2013.

[25] Shimmer. Shimmer devices. http://www.shimmersensing.com. Accessed: 2015-07-1.

[26] DW Trigg. Monitoring a forecasting system. *OR*, 15(3):271–274, 1964.

[27] Marina Zapater, Patricia Arroba, José L. Ayala, José M. Moya, and Katzalin Olcoz. A novel energy-driven computing paradigm for e-health scenarios. *Future Generation Computer Systems*, 34:138–154, 2014.