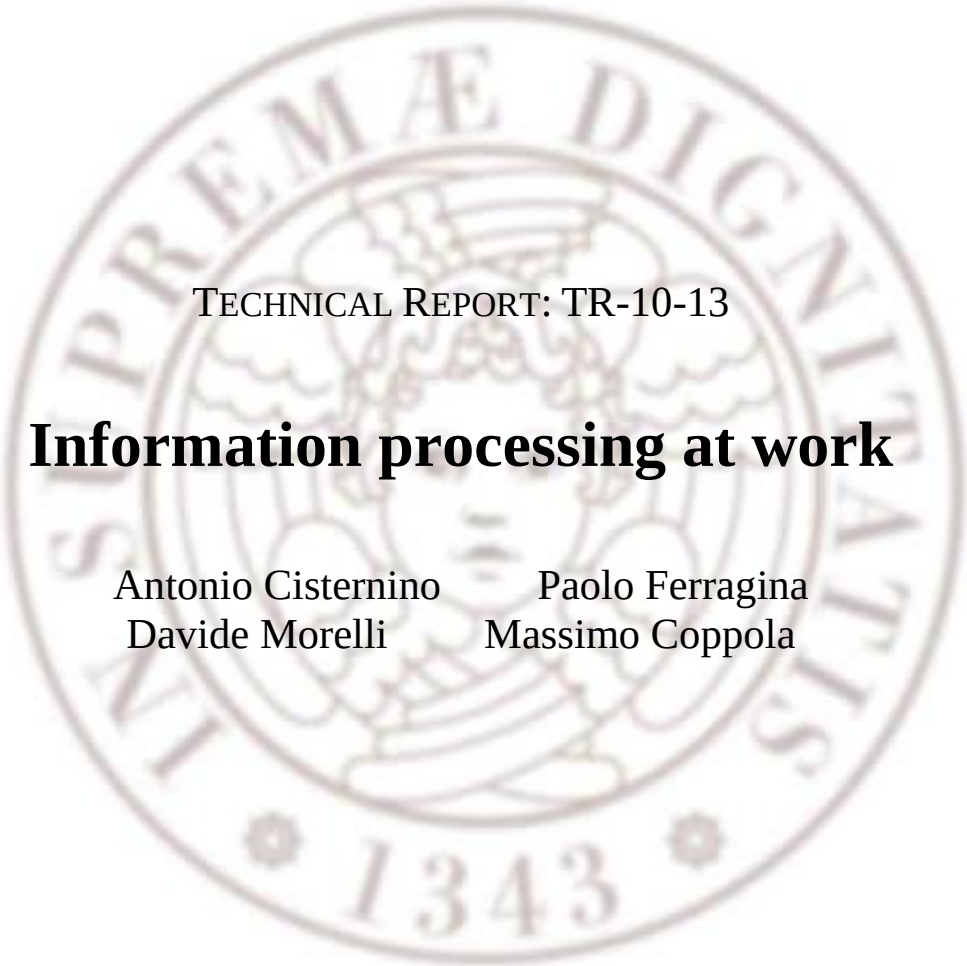
UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA



TECHNICAL REPORT: TR-10-13

Information processing at work

Antonio Cisternino Paolo Ferragina
Davide Morelli Massimo Coppola

July 7, 2010

ADDRESS: Largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700, FAX: +39 050 2212726

Information processing at work

On a theory for experimental algorithm complexity

Antonio Cisternino, Paolo Ferragina and Davide Morelli
Dipartimento di Informatica
University of Pisa, Italy

Massimo Coppola
ISTI "A.Faedo"
CNR Pisa, Italy

***Abstract*—It is common experience to upgrade firmware of mobile devices and obtain longer battery life, living proof of how software affects power consumption of a device. Despite this empirical observation, there is a lack for models and methodologies correlating computations with power consumption [3-5]. In this paper we propose an experimental approach to computational complexity and a methodology for conducting measures which result independent of the underlying system running the algorithm/software to be tested. Early experimental results are presented and discussed, showing that our methodology is robust and can be used in many settings. We also introduce the foundations of a theory for experimental algorithm complexity, which mimics what is predicted by the classic theory of computational complexity (big-O or Theta notations), except for some notable exceptions that we highlight and comment. This theory is validated in many scenarios, by considering several architectures and algorithms. Because of the relation between time complexity and energy consumption, we may suggest that our work measures the "information work": namely, the energy required for performing information processing.**

Keywords- energy-profiling of algorithms, power consumption models, RAM model, Energon, experimental algorithm complexity, green computing.

I. INTRODUCTION

Programs are made of instructions that manipulate system resources to achieve a goal. These instructions are executed by a processor (CPU), which uses electric signals to change bit configurations according to their semantics. Copying bits in a system also involves certain amount of physical work. Therefore, for each instruction of a specific processor, it is in principle possible to give an estimate of its power absorption that is roughly related to the micro operations performed to fulfill the instruction semantics. In these cases, researchers typically consider embedded systems, where computational-units are simpler than general purpose processors, often based on a RISC and missing many of the power hungry functionalities (multimedia and floating point instructions, high performance memory controllers) that are commonplace in the world of full-fledged computers. It is therefore not surprising that for such simple devices authors come to the conclusion that all instructions have the same power cost, and derive regular behaviors as well as detailed system models (see e.g. [21]). However, modern CPUs are much more sophisticated than that, in terms of complexity and sheer number of transistors. They are intrinsically parallel and concurrent at the micro-architecture level, with the actual amount of energy required to execute a single instruction being suitably estimated on average at the best. The variance of these costs may be large, and it worsens as we move higher in the *software stack.* Take for example virtual machines, such as the Java Virtual Machine: here we want to associate an average power consumption to complex entities such as the intermediate language opcodes. These can actually trigger quite (energy-)costly side effects within both the virtual machine and the hosting operating system.

It goes without saying that things get even more complicated when we need to define power cost models for programs, which thus involve a compilation step. This is the reason why [4] asked for "*models to be developed at all abstraction levels and granularities. […] These models will provide a solid baseline for higher level models, and many intermediate levels that can be of use to various layers in the abstraction hierarchy.*"

As a result, we can currently devise two main approaches to power-consumption modeling:

1. Accurate analytic models based on the explicit measurement of the energy consumption of a set of tests at the hardware level (see e.g. [13,15]). This approach is restricted to be used onto a narrow class of simple processors whose energy-consumptions features are well known and defined, such as embedded systems and/or micro controllers.
2. Modeling of power at the simulation level (see e.g. [16,17]). Here power-consumption of the underlying virtual machine [14] or of the entire system is taken into account. Unfortunately, this approach is limited to be used on simple codes to be profiled.

However, a model to be useful must be complex enough to capture the most important factors in the application (whichever it is), while being simple enough to allow abstract reasoning. The purpose of a good abstract model is not to model exactly all aspects of reality, but to be used as a heuristic tool by an engineer when searching for engineering solutions. The establishment of a good model can be the difference between success and failure of a science as a foundation of an engineering discipline [4]. Historically the success of algorithmics as a science underlying software engineering relied on the deployment of the Random Access Model (RAM) as a model for designing and analyzing algorithms. While the RAM model ignores issues such as memory hierarchies, it captures *enough reality* to be an extremely useful model for establishing the efficiency of an algorithm or for comparing algorithms based on their time performance. Here the analysis of algorithmic performance concerns with some computational resources and takes into account only *asymptotic trends* by ignoring constants, lower-order terms, as well as the fine details of the underline physical machine (architecture, compiler, language,…) on which the algorithm is executed. Primary resources are typically time and space.
Nowadays, PCs got more sophisticated, so that the RAM model became inadequate to estimate algorithm performance on large datasets. More sophisticated models have then been introduced in order to measure other (computational) resources such as I/Os (external-memory model [6]), parallelism and/or communication (logP, BSP, etc. [7]), hierarchy of memory levels (cache-oblivious model [8]), etc. etc.

Given that [5] "*some eight orders of magnitude separate the energy efficiency of conventional computers from what is theoretically possible*", energy-consumption became a primary resource to be optimized in systems and algorithm design, and thus many researchers as well as companies asked for models and metrics for evaluating the energy-profile of algorithms/architectures in advance to allow comparisons and novel designs. Yet, these issues are in their infancy although some notable efforts have been pursued [3-5]. Certainly a program's energy consumption strongly depends on the number of instructions executed and the number of accesses to the memory hierarchy --- these are the same factors that determine a program's completion time [3]. Also, an instruction that takes fewer clock cycles to execute generally also consumes less energy. However, as it was made obvious by the RISC/CISC debate a few years ago [22,23], simpler instructions lead on the one hand to a higher instruction count for the same algorithm, and on the other hand to systems whose clock cycle is possibly shorter. Beware: as power dissipation is related to the square of the clock frequency, the tradeoff between simplicity and power of the instructions, which has been

thoroughly explored in recent years as the market strived for faster and faster CPUs, turns out not to be easily transferred to the energy consumption. As stated in [3], "*the correspondence between completion time and energy consumption is not one-to-one. […] The average power consumption and computation rates are intricately tied together, making it difficult to speak of power complexity in isolation. […] This also indicates that models for the study of energy-computation tradeoffs would need to address more than just the CPU.*" However, literature still misses an "abstract" model, akin to the RAM, and a methodology, akin to the asymptotic analysis of computational complexity, that accounts for energy issues.

In this paper we try to contribute to these issues by presenting a *theory* for experimentally analyzing algorithms using power consumption as a measure of their behavior, and that results independent of the underlying system used to execute the algorithms/programs to be tested. We pursue our goal in four steps:

- First, we introduce an *experimental methodology* to measure program behavior in a robust yet simple way, so that comparisons can be made across different architectures, system configurations and algorithms, and by anyone without any particular HW/SW skills. Our methodology consists of some easy-to-build hardware tools and a public software to use them; they will be described in fine details in order to make our experiments reproducible (being this a key step of any scientific result!). Our HW is plugged between the power supply and the PC-case (on the alternating-current side), and still allows robust estimations without being invasive.
- Second, unlike other papers that use code profiling and measure the absolute energy consumption (see above), we introduce the notion of *Energon*, which is a sort of "unit of power consumption" for the system in use. The Energon in some sense mimics the role of the "algorithmic step" of the RAM model (unit of computation), here adapted to take into account the energy-profile of an algorithm and being as much independent as possible of the HW-SW specialties of the underlying system (as for the RAM-step). Energon is computed as the energy consumed by the CPU when running at 100% peak performance (because of the execution of some simple *registry-based* code, see Section IV). The Energon is therefore a relative value, that should allow comparison of algorithms/programs in a way that is independent on the underlying system which runs them.
- We validate the Energon by evaluating the energy-profile of some well-known algorithms (e.g. binary search, few sorting algorithms). We show that indeed our evaluation is robust and reflects what it is predicted by the RAM model, even if it concentrates only on the energy-profile of the algorithms and works on the alternating-current.
- Given these experimental results we introduce the notion of Ξ, the *experimental algorithm complexity*, inspired by the more popular Θ-notation of classic computational complexity. We use it to characterize the energy-performance of some well-known algorithms. Interestingly, Ξ is *compositional* and thus can be used, as it occurs for Θ, to compare CPU-bounded algorithms based on their energy-profile in a system-independent way.
- Finally, we extend our study to other processors and architectures thus concluding, as claimed in [3], that the relation between time- and energy-efficiency is much intricate and although Ξ somehow mimics Θ, they are much different when measuring the execution of parallel algorithms on many-core systems and over a hierarchy of memory levels. We quantify these differences and thus draw some novel, preliminary, interesting conclusions that speculate onto possible applications of *our theory and methodology* to green computing.

It goes without saying that our work is still in its infancy, but nonetheless we believe that our results are interesting in that they relate "information-space" properties to "physical-space" properties by reasoning about algorithms and programs on the ground of *whole* real systems. Moreover, the theory we propose, although confined to CPU-bounded computations aims, with its simplicity, at addressing the issue drawn in [4] for "*models to be developed at all abstraction levels and granularities. […]"*.

The paper is organized as follows: in section II we introduce our methodology and the goals of our work; section III describes the measurement setup for reproducing the experiments and discuss some early results; section IV introduces the notion of *Energon*, a meter we use to normalize measurements; section V discusses the analysis of few basic algorithms with the goal of correlating our experimental theory of algorithm complexity to the classic theory of computational complexity, and validate the Energon-based energy-profiling of those algorithms; section VI introduces some definitions and the notion of Ξ, our complexity measure, inspired by the more popular Θ; sections VII-VIII discuss how our measure adapts to different processors and architectures; section IX speculates possible applications of our theory with respect to benchmarking, black-box complexity, and green computing; section X draws some final conclusions and suggests future works.

II. A METHODOLOGY FOR EXPERIMENTAL ALGORITHM COMPLEXITY

The way we write programs affects their power consumption; so our primary goal is to build a methodology for empirically measure this effect in a way that is as much independent as possible from the experimental setting, namely the system running the experiment. Our further goal is not only in reporting our own experimental results, but also in clearly defining an *experimental framework* that our community can use for its own experiments or for validating ours.

Thus we are looking for an experimental setup with the following characteristics: Reproducibility, Robustness, and Universality.
As we commented in the introduction, most of the existing power-modeling results are specific to a particular architecture, so their setup is difficult to be generalized to other HW-SW settings. We instead aim at a universally applicable measure that results agnostic of the underlying HW/SW components. The first contribution of this paper is therefore a robust and easy-to-implement experimental methodology for forecasting expected consumption and energy savings deriving from architectural as well as software choices. It will be based on an *ammeter*, and a (public) software that reads the ammeter and performs the measurements over the tested algorithm. We verified that our methodology is *sufficiently sensible* in comparing energy-consumption of algorithms at a fine scale. Moreover, it allowed measurements on standard PCs without any particular OS configuration, and indeed they successfully extend to other architectures and settings, as we will comment in Sect. VII-VIII.

To validate our methodology, we tried to relate classic algorithmic complexity (based on the RAM model) to power consumption, as the input grows. As we discussed in the introduction, it is reasonable to assume that power consumption is related to the number of operations performed by an algorithm so, if our methodology is sound, then its energy-evaluations should resemble what is predicted by the asymptotic time-complexity on CPU-bounded computations (given the limits of the RAM model). And indeed this is the case for binary-search and various well-known sorting algorithms. However, in order to make a *theory* we cannot rely our algorithmic comparisons onto their *absolute* energy-consumptions, since this would be dependent on the specific PC on which the experiment is executed. Therefore, in order to mimic the role of "algorithmic step" in the classic time-complexity analysis on the RAM model, we introduce a normalization factor that allows us to

obtain data-independent energy–consumption evaluations (rather than using absolute Watts and/or Joules). This is called the *Energon* and allows us to relate pretty well the complexity expected from algorithmic theory to our energy-aware experimental complexity, independently on the executing system. In few words (for details see Sect. IV), the Energon relates the energy-profile of the tested software with the one of a simple assembler-code that executes 1G register increments. As a result, the Energon does not measure energy-consumption but it provides system-independent measures for algorithm comparison (à la RAM, but now for the energy resource).

A nice property of our theory will be that it is *compositional*: in fact, we will *experimentally* show that it allows to evaluate energy consumption of a program by summing up the costs of each one of its constituting algorithms, in an arithmetically predictable way. Moreover it is *robust* enough that we can use it to evaluate the power efficiency of whole programs on different processors and/or architectural settings (e.g. Atom vs. AMD CPU, single- vs multi-core computation). Surprisingly we will show that there are significant savings when an Atom CPU is in use, and also that exist a wasting of up-to 10% for a quad core mostly idle (see Sect. VII for details). These, as well as other features (e.g. how the pattern of memory accesses impact onto the energy-profile of an algorithm), will help us in deriving some interesting *invariants* that could be deployed to design novel energy-efficient algorithms.

III. TEST ENVIRONMENT

The test environment has been redesigned several times because of many obvious choices do exist, each with its own pros/cons (e.g. think to the many positions an ammeter can take among the modules of a PC motherboard).

Our first choice has been to choose an ammeter that is commercially available, cheap and that can be easily programmed, in order to automate the reading of experimental data and the transformation of the readings into Joules. We opted for the Phidgets [1] system, that features a plug-and-play approach to electronics: a microcontroller with a USB interface can be plugged with a number of sensors with a standardized interface. We used the Phidgets ammeter 1122 sensor [2] that has a range of 30A and 0.04A of resolution on AC. The typical error is between 1% and 2% with a maximum error of 5%. The Phidgets system features several programming libraries targeting almost all operating systems and programming environments nowadays available. We used the .NET interface and developed the software using C#. Other ammeters could be used in place of the one we selected, provided that they offer a sampling frequency of at least 10Hz and a comparable error range.

Given this HW, we proceeded to measure the current provided to the motherboard by instrumenting the power lines running inside the case from the power supply. We made this based on the belief that the circuitry of the switching power supply, which transforms alternating current into direct one, would reduce the ability of detecting current variations of our ammeter. Moreover, we thought (following [13]) to be able to test single contributions of different components fed by separate power lines, such as hard drive, CPU and memory. Although sophisticated and precise (potentially useful for digging into our results into the near future), we decided to drop this approach in favor of the simpler one that detects current variations on the alternating current side of the power supply. Several experiments showed that this was enough to obtain reliable measurements. Thus we decided to use a single ammeter connected to the AC line of the target computer, with increased ease of instrumentation.

Like [13], we designed a two-parties software system (see Figure 1): one is responsible for collecting data from the ammeter; the other runs on the target machine and is responsible for

starting the tested software (on that same machine) and signaling its start/end to the former (data-collecting) software.

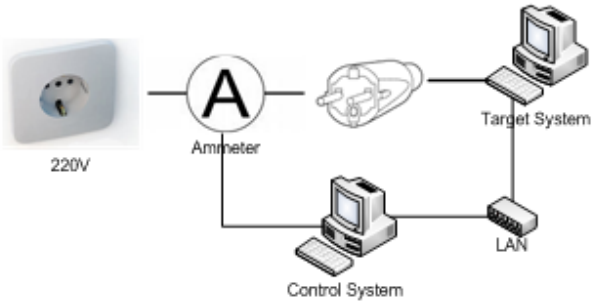


Figure 1. The structure of the experimental setup

One may wonder whether the use of a network affects the measures. We verified with several experiments that such an approach is indeed robust enough to ignore micro variations due to the operating eco-system, provided that experiments are conducted when the system is idle and no *significant* programs are running (i.e. CPU is essentially idle). We will discuss further this analogy with eco-systems in the next section.

Figure 2 shows the power consumption (in Watts) on an 8-core system, which was first idle and then executing a program that performs a linear scan of a huge array (marked by green and red lines). The OS exhibits a stable behavior when idle, but it is also interesting to notice that the spike due to a full use of the 8-cores accounts only for around 40% an increase in power consumption. Idle computers consume a lot of energy! (see also [9])

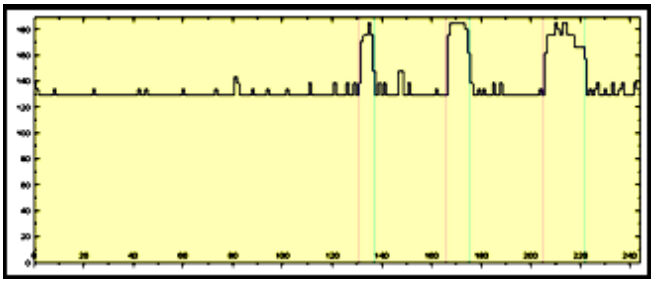


Figure 2. Idle operating system, and with a program running.

The two programs used for performing measurements are open source and available on the CodePlex online site [12] (remember one of our main requirements--- reproducibility).

The testing software performs a sub-sampling of the data received from the ammeter sensor by computing the average of the samples received each tenth of a second, thus reducing jittering effects due to the environment and possible errors in the readings. The rest of the system relies on a 10Hz measurement rate. This simple strategy will be subject to improvements and refinements over time, though so far it has been able to effectively show an appropriate amount of details in the data. We computed several values from the raw current readings taken from the sensor. We are able to estimate the power consumption (in Watts and Joules) by using well known relations among the quantities involved. Namely, power is obtained by multiplying the current and the voltage together (220v AC in Italy). We disregard the power factor $cos(\Phi)$ in the conversion, which leads us to slightly over-estimating the power (by a constant factor that can be easily reinserted). The energy expressed in Joule is then obtained by discrete integration (multiplying power and time together for each time slot and then summing up the results).

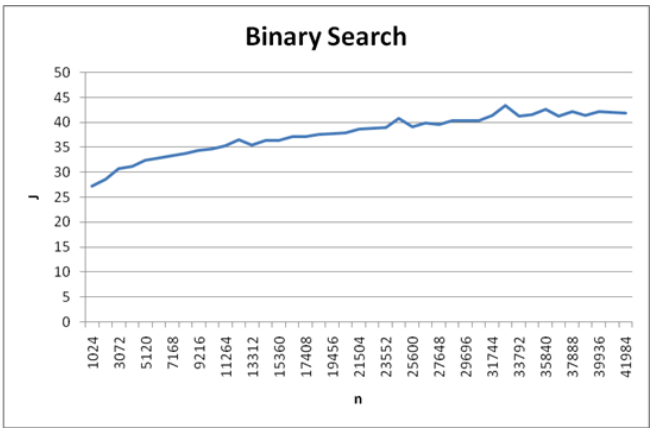


Figure 3. Energy consumption of a (log n)-time algorithm.

As an example, Figure above shows the energy used to perform a binary search on an array of increasing size. In order to obtain measurable durations we iterated each search 2^{20} times, which on the particular machine we used to perform the experiment took between 0.4 and 0.75 seconds to

complete. The whole test has been iterated 30 times and the results averaged. We actually employed an iteration factor of 30 for all tests described in this paper, with the only exception of those reported in section VIII where a factor of 3 has been used.

We find amazing that the behavior of an algorithm with computational complexity $\Theta(\log n)$ can be measured through the power line, and thus our simple methodology and HW-tools. The logarithm function fits the energy-consumption plot, as expected. In the subsequent sections, we will present energy-plots of more intensive computations, showing even more precise trends that match the curves predicted by the classic (asymptotic) computational complexity, and thus regarding so called CPU-bounded computations.
However, as observed before, Joule is an energy measure useful for speculating about power consumption of a particular system, but it cannot be used to *compare measurements* across different systems, as any *"reasonable" model* should allow to. We need a more robust measure which is invariant wrt the underlying system running the experiment, thus playing the role of the RAM model but in the energy-profiling setting. This is what we introduce in the next section.

IV. THE ENERGON

It is well known that often measurements taken in clean environments differ from those ones obtained in the real world. For this reason, or because a "clean" environment is sometimes impossible to define, industrial benchmarks like SysMark 2007 [18] perform system testing empirically by starting several times a set of productivity programs installed on a "clean" system, trying to reproduce typical usage patterns.
From our perspective the system we observe is a sort of eco-system, including OS processes and services triggered by external events and interacting with other applications. We want to measure changes that happen when we alter the situation by running our program, distinguishing them from the overall picture and from the noise caused by the mentioned eco-system. Two different runs of the same program may well lead to different results, so we need a method that distinguishes the contribution of the program we want to observe from the contribution due to the underlying system.

The first attempt has been the obvious one: measure the energy absorption of the idle system in Joules or Watts for a given amount of time, and then subtract this "milieu" from the measure acquired when our tested-program runs. This approach is *ill-conditioned* because the power consumption of the idle system is usually several times bigger than both the signal we want to measure and the additional noise, and moreover there occur sometimes spikes that are clearly related to some service running that jeopardize the interpretation of the measurements.
To circumvent these drawbacks, we have drawn inspiration from approaches taken by biomedical engineers for measuring *cell eco-systems* [19]. When biologists measure cells activity on a terrain, for instance for drug testing, they first take measurements in a reference system, and then, after performing the change they want to observe, they take additional data. Results are then normalized with respect to the first reference that *acts as a meter*. There are two aspects in this approach worth of notice: reference measurements are made because it is virtually impossible to reproduce exact environmental parameters (such as temperature, humidity, etc.); and the reference system *includes* cells, and not just the terrain, because cells that interact with their environment may alter the terrain parameters even before the desired stimulation is performed.

We found strong analogies between our framework and the cell eco-system. In particular the fact that an idle OS may change the execution patterns of its services (the eco-system) when a new program runs and asks for resources. We thus introduce a *reference program* in our experimental methodology that is run in the same environment where the experiment should be

executed. Then the energy-cost of this reference-program is used as a meter for normalizing the results of our experiments. There are several choices for such a meter, taking for instance into account different architectural features and environment services, and we expect over time to study several other meter definitions. In this paper we propose the *Energon* program, which is designed to use the CPU (or a single core of it) at full speed for a certain amount of cycles (namely 1G), and before the real experiment is run.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int a=0;
    int i = 0;
    std::string line;
    std::cout << "begin\r\n";
    std::getline(std::cin, line);

     __asm
     {
      mov ecx, 40000000h
      mov eax, 0
loop0:
      inc eax
      loop loop0
      mov a, ebx
     }

    std::cout << "end\r\n";
    return 0;
}
```

tested algorithm

repeated 1G times

Figure 4. The energon definition in C++.

Figure above shows the simple C++ source code of the Energon. It consists of assembler code explicitly inserted in a loop to ensure that no compiler could change it and thus render the measurement compiler-dependent. The surrounding C++ code is used to communicate with the testing software and signals when to start and stop the measurement. In this way more complex algorithms can exclude their setup phase from the measurement.

The code can be easily adapted to different architectures. We deliberately avoided at this stage a (Energon) program involving memory, since communications between CPU and memory are asynchronous and may introduce unwanted idle cycles in the CPU which heavily depend on the hierarchy of memory levels. We will discuss further this issue in section VII, and defer the study of the impact of the pattern of memory-accesses onto algorithmic energy-consumption to the final version of this paper.

While the Energon test can be still thought as an idle power consumption plus a "100% computation overhead", it is nevertheless measured as a whole. When we test the real algorithm, its measure in Energon (we use the ε Greek letter to denote 1 Energon) is computed as the ratio between its overall power consumption and the overall power consumption of the Energon program. This makes the measurement more stable and robust, because it includes the "constant" consumption incurred by the PC during the test.

As already mentioned, measures are repeated several times (in our tests 30 times) to compute the average energy used by the Energon program and the program tested. We obtained a standard deviation of 2.8% of the average energy required by the Energon[1]. It is worth noticing once again that the Energon does not depend on time: it simply measures the energy required for incrementing a register one billion of times. Of course it will take more or less time depending on the particular processor used, but in some sense it captures the notion of "computational power per electrical power" of the processor. And in this sense, it is mimicking the notion of "algorithmic step in a RAM model" that represents the computational unit independent on the HW-SW features of the PC that will run the algorithm; here the Energon plays the same role but with respect to the energy-profile of an algorithm.

The first question we faced after defining the Energon was: what about its *stability and composability*? We expect that if we iterate the Energon program *h* times, we observe a corresponding increase in its energy consumption. We call this new program ENERGON(*h*) (and the

[1] We avoid referring to specific architectures because we expect that these numbers will change as the method will be used on different systems. In this particular case we tested the energon on an AMD Athlon 64 X2 Dual Core 4200+, 2.20GHz 2 GB RAM, running Windows Vista H.P. We obtained an average of 177,2516*j* of energy used with 3,464587*j* of standard deviation.

original is Energon(1)). Experiments show that the ratio ENERGON(h)/h*ε is correctly close to 1 (0.97 in average with 0.01 of standard deviation).

Given the Energon, we can evaluate the energy-profile of the binary search, now scaling the energy-consumption by the Energon measure. The obtained plot (not shown for the lack of space) is smoother, still retaining its logarithmic shape.

V. SORTING ALGORITHMS

In this section we validate the robustness of our measurements by investigating the *energy trends* of few sorting algorithms over increasing input sizes. This shows that our methodology is able to predict algorithm trends in energy consumption quite carefully. In the next section we will elaborate on these results by introducing a *theory of experimental algorithm complexity*, useful to compare algorithms based only on such observed energy-trends. Finally, in section VIII we will show that energy consumption is not always related to completion time, especially when considering thread-parallel computations.

We considered three well-known sorting algorithms: merge-sort, heap-sort, and quick-sort [20]. They vary either in their worst-case running time, or in the amount of working space, or in the pattern of their memory accesses. In all cases we measured only the sorting phase, ignoring the setup of the input array with random data. We repeated the test 30 times in order to get empirically sound input distributions. All the algorithms have been implemented and tested using C++. Tests using C# were also performed, obtaining similar results, but showing less clear trends due to C# virtual machine services (such as the garbage collector) coming into play and creating irregularities and spikes in the data. While the situation is in the scope of our work, in this early stage of our validation process we preferred to use real measurements which are more predictable and easily handable.

Merge sort is an optimal comparison-based sorting algorithm that requires Θ(n *log* n) time in the worst and average case. As shown in Figure 5 below, the energy consumed by the algorithm as a function of its input size follows exactly that prediction obtained with the asymptotic time-analysis in the RAM model.

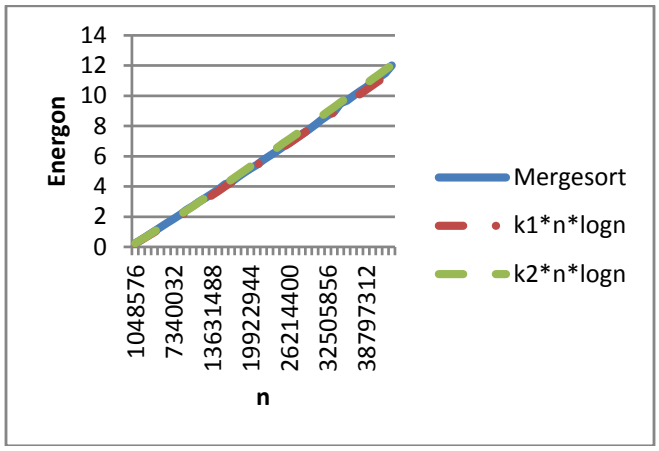


Figure 5. Merge-sort energy consumption in Energon

Heap sort is another optimal sorting algorithm which works in-place, and thus uses no additional working space, unlike Mergesort; however it induces a *random-pattern* of memory accesses. Figure 6 below shows an asymptotic energy-consumption trend towards the expected Θ(n *log* n), though at small input sizes there is a deviation with respect to the curve expected from theory.

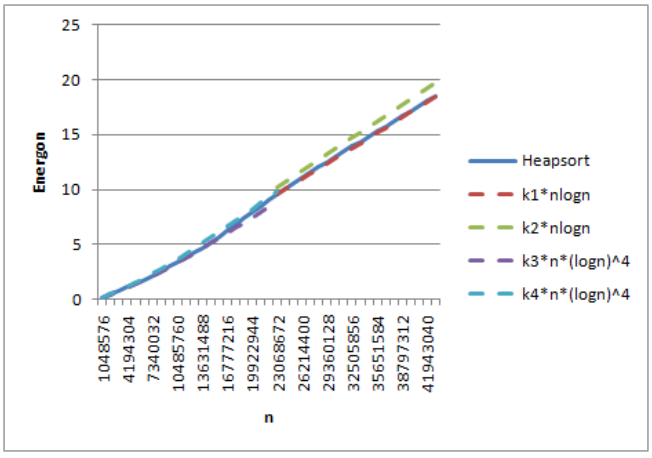


Figure 6. Heapsort energy consumption in Energon.

Overall heap-sort is less efficient than merge-sort, even if the asymptotic behavior is the same. Using heap-sort requires more than twice the energy of merge-sort. This is probably due to the pattern of memory-accesses that makes merge-sort cache-friendly: time-efficiency implies less waiting time by CPU, hence less power consumption. Given these results, in the final version of this paper we will deeply present our investigations about the non-negligible impact of the pattern of memory-accesses onto the energy-profile of an algorithm, and thus onto its green-design.

Quick sort is well known to be $\Theta(n^2)$ in the worst case and $\Theta(n \log n)$ in the average case, with lower constants than the other sorting algorithms. We were curious to see how our experimental results would have recorded this non-trivial trend. Figure 7 shows the energy-profile of Quicksort expressed in Energons. The curve is plainly different of the ones shown before. We tried to fit data against different functions using the least square method. Both n^2 and $n \log n$ functions fitted nicely, though the best fit was obtained by:

$$f(n) = n \log n + \frac{n^2}{a}$$

with a constant value (the best fit has been found for a value of a=512). It is in our opinion very interesting that the overall cost includes both terms predicted by the worst- and average-case asymptotic analysis.

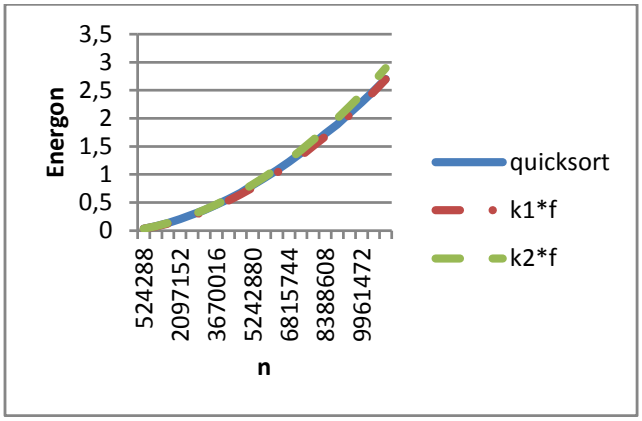


Figure 7. Quick sort energy consumption in Energon.

VI. EXPERIMENTAL COMPUTATIONAL COMPLEXITY

Even if at first sight the correlation between time and energy may suggest that an experimental theory of program complexity can be based *just on time-performance*, we will find that this is not the case when we consider other processors, the hierarchy of memory levels and parallel CPUs.
However we cannot neglect *such correlation*, which comes out because obviously the longer an algorithm executes, the highest will be the power demand of the CPUs. This correlation is useful when considering CPU-bounded computations because of our simple methodology to measure the energy-profile of a *sequential* algorithm.

We introduce a notation for our experimental complexity theory inspired by the traditional Θ-notation. In our case, however, we cannot rely on asymptotic behavior since by definition our measurement system is finite.

Definition (Ξ-notation). Let us given a set of energy-measurements A expressed in Energon (i.e. ε) and representing the energy-profile of an algorithm with respect to a set of inputs whose size is in the range $[a, b]$. The data set is said to belong to $\Xi(f(n))$ over that interval, if there exist two constants k_1 and k_2 such that:

- $\forall x \in [a, b]. data(x) \in A \Rightarrow k_1 f(x) \leq data(x) \leq k_2 f(x)$
- $\exists m_1, m_2 \in A, x_1, x_2 \in [a, b]$ s.t. $k_1 f(x_1) = m_1$ and $k_2(x_2) = m_2$

and we will write that $A \in \Xi(f(n))$ with respect to ε and [a,b].

The definition tries to capture the idea that data can be *enclosed* by a function, with the additional requirement that the weighted function must pass through two data samples in order to take the tightest among many possible functions.
As an example, Figure 11 shows another run of binary search with $f(n) = \log n$.

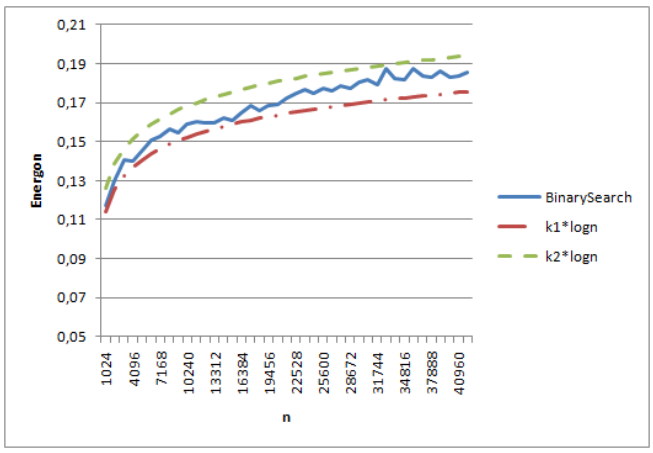


Figure 8. Binary search is Ξ(*log n*)

Since the interval is finite it is possible to create several functions *f* that state an experiment to be Ξ(*f*). To avoid these ambiguities, we introduce notions that characterize relevant behaviors of Ξ.

Definition (Ξ-preferred). Given a set of data *A* over an input size interval [*a, b*] and two functions *f* and *g*. We say that *f* is Ξ-preferred with respect to *g* if $A \in \Xi(f)$ and $A \in \Xi(g)$ but *f* is a better least-square approximation than *g* on *A*'s data.

Definition (Ξ coherence). A function *f* is said to be Ξ coherent with respect to an algorithm if for each interval [*a,b*] of input sizes the resulting data set $A \in \Xi(f)$.

These definitions attempt to capture common patterns we found in the data. For instance we found that merge-sort is Ξ(*n log n*) coherent. We quickly find useful to have a precise meaning to speak about experimental data, but these definitions do not only address description needs. Actually, consider the following law:

Law of composition. If two algorithms are respectively Ξ(*f*) coherent and Ξ(*g*) coherent, then the algorithm obtained by executing them in sequence (i.e. one after the other) is Ξ(*f* + *g*) coherent.

This law has been verified in all our tests, and we are currently working onto verifying functional composition of algorithms: for instance if a sorting algorithm receives a comparison function that is not constant what happens to its experimental complexity and Ξ?

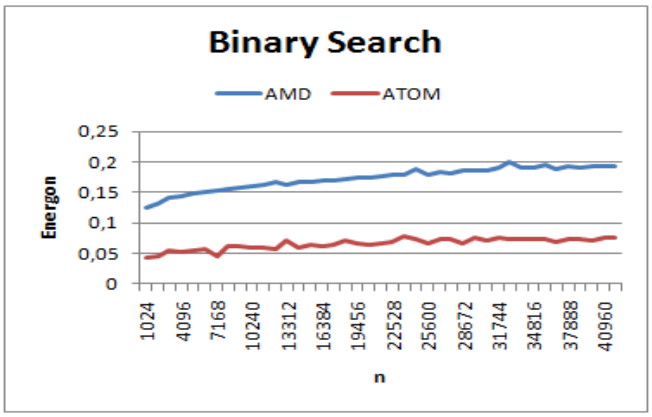
It is important to notice that even if we mimicked the definition of Θ, in our case there is no way to simplify composition. If you combine a *quadratic* algorithm with a *linear* algorithm, then you obtain a quadratic algorithm in Θ. With Ξ we can only say that the algorithm is made of the sum of quadratic and linear energy consumption. This is again due to the lack of asymptotic reasoning.
Overall we have verified the following relation: if an algorithm has Θ(*f*) time-complexity then it is Ξ(*f*) coherent, and thus our methodology is robust for CPU-bounded computations. The converse in not necessarily true, as we will show in the next section.

VII. Experimenting different architectures

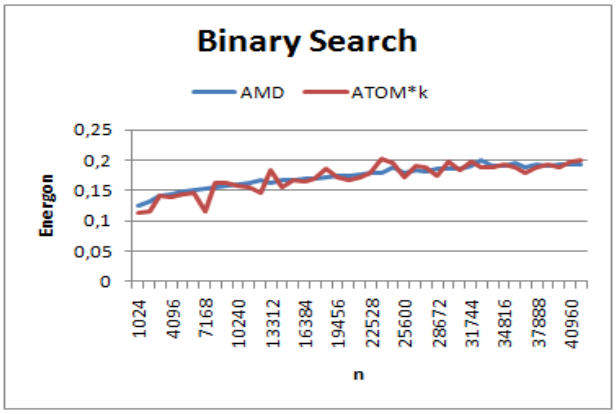
We tested two architectures: an AMD Athlon 64 X2 Dual Core 4200+ 2.20GHz based system with 2Gb of RAM, and an Atom N230 based system with 2Gb of RAM. The two architectures have been chosen because they are radically different. We have executed all algorithms of the previous sections (i.e. binary-search and sorting) with the twofold goal of verifying that Energon results are confirmed across different architectures, and thus that Energon normalization allows for data comparison between different systems.

First of all we compared the two Energon computed by the two architectures, in order to get an idea of the overall efficiency of the two platforms. Results confirm the power efficiency of Intel Atom: it absorbs *about half* the energy taken by AMD. Of course this measurement includes not only the CPU-cost but also the energy-cost of the whole system.
Then we applied our experimental methodology to binary search and the three sorting algorithms above. Because of lack of space Figures below

refer only to the Energon-plots for binary search (results on the sorting are even better). It can be noted that data follow the same trend on the two architectures (i.e. they belong to the same class of Ξ) but their plot do not overlap.



A possible explanation is that Energon measures only the CPU, not taking into account the energy-efficiency of the whole architecture which therefore introduces some "energy-gaps" which are not negligible especially with the increase of the input size. (In the final version of this paper, we will account on these issues.)



We tried then to correlate the measurements of the two architectures, and found that the two plots overlap perfectly if multiplied by a constant *k* (see pictures below). For binary search, we found k=2.8; for quick-sort k=3; for merge-sort k=6. This fact is a further confirmation of the robustness of our experimental method: we can still predict the energy-profile of algorithms on different architectures, and show that they have the same behavior when energy consumption is scaled using the Energon.

A natural question now is why the constant factor changes? We would have expected the same constant for all the experiments, thus depending only on the two architectures and not on the tested algorithm. This would have allowed us to model

- the energy efficiency of the algorithm (and in perspective, of complex applications) expressed by Ξ relations of complexity functions,
- the relative characteristics of two or more system architectures (these would be classified in terms of a constant factor expressing their energy efficiency).

Instead, we found that the constant *k* varies with each test. We thus started investigating if there were constant contributions of the architecture which were not properly normalized by the Energon, and thus we first took into account the overall running time of the tested algorithm.

It turns out that Quicksort and Binary Search need similar values for the multiplicative *k* across the two different CPUs (2.8 versus 3) in spite of having quite a different running time (the largest instance of Binary Search test is about 12 times faster to complete than the largest one of Quicksort), different algorithmic complexity and different Ξ class. They do share, however, an *random*-pattern of memory accesses.
Conversely, if we compare Quicksort and Mergesort, we notice that they have very close completion time and share the same asymptotic time-complexity on average, but do generate a completely different pattern of memory accesses, and indeed they need a significantly different value for *k* (3 versus 6).
The Mergesort algorithm produces the sorted output with 1/6 of the Energon amount on the Atom architecture wrt AMD one, while the same architecture only saves 1/3 in the Quicksort case. We argue that this is due to the fact that

Mergesort triggers a cache-friendly pattern of memory accesses, and this seems to better exploit the energy-efficiency design choices of the Atom processor.

Given these preliminary results, we are now starting an in-depth study of the effects of memory-access patterns on our Energon–based methodology, to refine its definition and allow more robust comparison of results across architectures. We plan to include these results in the final version of this paper.

VIII. MANY-CORE COMPUTATIONS

So far we have discussed measurements of sequential algorithms, mainly to validate our results against time complexity theory (and thus CPU-bounded computations), relying on the strong correlation of energy consumption and completion time.

In this section we are interested in studying how energy consumption of algorithms is affected by parallel execution, and in particular what can we expect from many-core architectures that are becoming ubiquitous in nowadays computing.

To understand this we measured the energy-profile of a simple linear-scan over a long array on a quad-core system with hyper-threading (a total of eight cores for the operating system). We compared the sequential version of the linear scan against several parallel versions using two, four, and eight cores. Parallel scanning has been implemented by partitioning the input array in p segments with p the number of cores used.

Theoretically this algorithm scales optimally since there is no access to shared data and each thread can run at its full speed. Ideally we expect that the energy consumed stays the same no matter how many cores are used, since the same total number of bits should be processed/accessed. But surprisingly, this is not the case.

Figure 9 shows the *average power consumption* (W) for the executions of linear scan using different number of cores. If, however, we measure the same linear-scan in Energon (plot not shown), it turns out that the most efficient way to use the CPU is by deploying all cores together, with sequential execution requiring three times more the Energons of 8-cores execution.

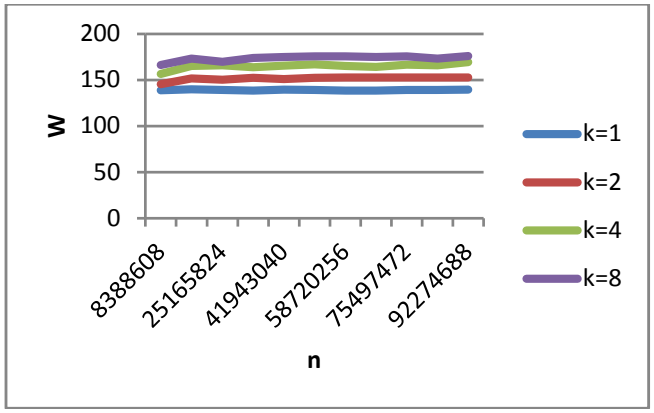


Figure 9. Average power used in the four experiments.

To understand this phenomenon we assumed that an idle core would consume a certain amount of energy that we set to be constant during the execution. Under this assumption we have been able to estimate the energy used by an idle core solving a simple system of equations. Waste due to idle cores has proven to be quite significant with respect to the overall computation as shown in Figure 10, and it is up to the 10% of the overall computation in the single core execution case.

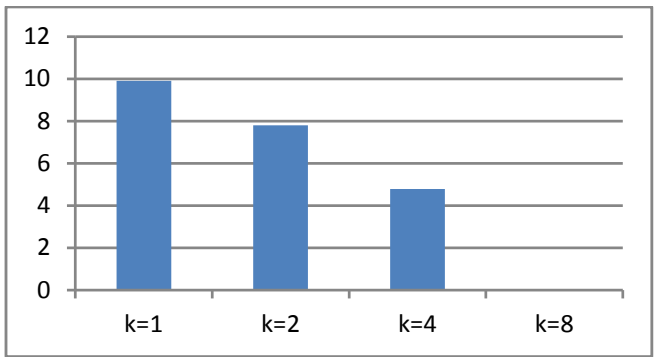


Figure 10. Percentage of energy wasted by idle cores

An important fact is that completion time and energy consumption *are not* so strictly related as it was in the sequential case. This is witnessed by the change in the average power absorbed by the processor depending on the number of idle cores. In particular we found that if C_k and T_k are the energy consumed and the completion time of

linear scan using k cores, the following relations holds:

$T_2 = 0{,}53\ T_1$ $\quad C_2 = 0{,}58\ C_1$ [ideal is 0.50]
$T_4 = 0.35\ T_1$ $\quad C_4 = 0.43\ C_1$ [ideal is 0.25]
$T_8 = 0.28\ T_1$ $\quad C_8 = 0.36\ C_1$ [ideal is 0.12]

These formula are NOT surprising because they show that, although intrinsically parallel, the linear-scan does not achieve ideal-parallelism (the best speed-up is 3.81 due to resource contention among the threads competing for the memory bus); nonetheless, these formula are interesting because the energy consumption of 1 core (i.e. C_1), expressed in Energon, is larger than the other C_i which decrease as the number i of cores increases. This is counterintuitive because few active cores consume more than many active ones; the reason is that C_i is energy-consumption (properly normalized) and we recall that Joule = W*T. So by increasing the number of cores the completion time decreases but the total consumed watts increase, with the balance being in favor of time reduction.

Further investigation is still required to improve our model and dig into the specialties for the many-core case. These experiments indicate that underuse of computational resources can be very expensive from an energy efficiency standpoint. We expect that our simple methodology of energy-profiling can be used to evaluate the efficiency of a system with respect to core-usage leading to a more savvy strategy in computing resources acquisition than simply "let's have as many cores as possible".

IX. PROGRAM ENERGONOMY

Even if we decoupled the notion of completion time from that of energy consumption, at least in the many-core case, the two quantities are still related. Writing efficient programs usually leads to energy efficient systems; this is well known, of course, but now we have a simple methodology to quantify it or compare programs according to their energy-efficiency under several architectures. This is important also because completion time is becoming less critical for ordinary applications, as witnessed by the increasing number of codes written via interpreted languages such as Python. From a user standpoint a tenth of a second or half second does not really matters, and this explains the popularity of these coding approaches. However it is also well known that Python is more than fifty times slower than C in the average (see Debian shootout [11]). Thus we expect that Python programs will consume more energy than their C/C++ equivalent. Of course scripting is easier and more flexible, but it must be clear that even if performance is acceptable there is a hidden energy-price to pay. Same arguments hold for virtual machines, such as Java or Mono (.NET implementation for Linux). They are flexible, but their average performance is twice slower than C, and thus we expect such wasting of energy when using that programming framework. We will perform more precise evaluations in the near future to further quantify these hypotheses, even if they are clearly suggested by the results found in this paper.

In summary, a greener computing infrastructure is achieved not only by optimizing the IT infrastructure, but also by writing better (i.e. more efficient) software. And, as witnessed by our work, the impact can be as significant as much as hardware improvements (or even more!). Our experimental methodology and protocol offers a clean and simple way to measure software improvements and benchmarking. In particular we have three dimensions: input size, algorithm, and architecture. We can perform energy efficiency benchmark by fixing two out of three dimensions and observe the variations. If we fix the input size and the algorithm, we can benchmark architectures as we did in section VIII. When input size and architecture are fixed, we can benchmark the algorithm efficiency as we did in section V for sorting. If architecture and algorithm are fixed, we can see how well an algorithm scales with respect to its input, as we did in many examples thorough the paper.

The ability of practically measuring the energy-profile of an algorithm, or even an entire software, is likely be the basis for what we call *energonomy*: the attempt to write energy efficient programs.This will be a key issue in the next years because, as observed in [3] *"Algorithmics offers benefits that extend far beyond TCS into the design of systems."* The simple methodology for energy-profiling of algorithms, described in this paper, can be adopted to quantify these benefits is a robust yet simple way. In particular, in the final version of this paper we will address the energy-issues concerned with the pattern of memory-accesses deployed by an algorithm in its execution; some of our preliminary results (to be included in the final version) already show that the impact of these accesses cannot be neglected and may even be of order of magnitudes. So any programmer should take them into account when designing a software.

X. CONCLUSIONS AND FUTURE WORK

In this paper we introduced an experimental methodology for measuring energy consumption of programs in a robust way, largely independent of the particular environment used to perm the experiments. Energon, our unit for energy-profiling of algorithms, has proved to be a useful meter for our investigations.

Philosophically, our theory follows naturally by the consideration that at the very core of computation there is the notion of physical work required for moving electric charges from one register to another, or changing a circuit state. So we may suggest that we are measuring the "information work": the energy required for performing information processing.

We hope that this preliminary work may contribute to set a community of researchers that use our methodology to investigate other combinations of algorithms/architectures. We published our testing software on the Web [12] and we will start a web site where researchers and practitioners may upload the results of their experiments.

REFERENCES

[1] Phidgets system Web site, available at http://www.phidgets.com/, last access: April 5, 2010.

[2] Ammeter sensor data sheet, available at http://www.phidgets.com/documentation/Phidgets/1122.pdf, last access: April 5, 2010.

[3] K. Kant, Toward a science of power management, IEEE Computer, 42(9): 2009.

[4] Workshop on the Science of Power Management, NSF, April 2009.

[5] Disruptive solutions for energy efficient ICT, EU Expert Consultation Workshop, Brussels, February 2010.

[6] J.S. Vitter, *Algorithms and Data Structures for External Memory*, Series on Foundations and Trends in TCS, now Publishers, 2008

[7] T.H. Cormen, and M.T. Goodrich. A bridging model for parallel computation, communication, and I/O. *ACM Comput. Surv.* 28, 1996.

[8] R. Fagerberg: Cache-Oblivious Model. Encyclopedia of Algorithms, Springer, 2009.

[9] L.A. Barroso, U. Hölzle. The Case for Energy-Proportional Computing. IEEE Computer 40(12): 33-37 (2007).

[10] Computational Intelligence in Scheduling (SCIS 07), IEEE Press, Dec. 2007, pp. 57-64, doi:10.1109/SCIS.2007.357670.

[11] Debian language shootout, available at http://shootout.alioth.debian.org/u32/benchmark.php?test=all&lang=all.

[12] Energon software web site, available at http://energon.codeplex.com/, last access April 5, 2010.

[13] D. Economou, S. Rivoire, et al. Full-system power analysis and modeling for server environments. Workshop on Modeling, Benchmarking, and Simulation (MoBS), 2006.

[14] S. Lafond, J. Lilius. An Energy Consumption Model for Java Virtual Machine. TR 597, Turku Centre for Computer Science, 2004.

[15] C. Seo, G. Edwards, D. Popescu, S. Malek, N. Medvidovic. A framework for estimating the energy consumption induced by a distributed system's architectural style. ACM International workshop on Specification and verification of component-based systems, 2009.

[16] D. Brooks, V. Tiwari, M. Martonosi, M. Wattch: A framework for architectural-level power analysis and optimizations. Annual International Symposium on Computer Architecture (ISCA), 2000.

[17] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, M. Kandemir. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. International Symposium on High Performance Computer Architecture (HPCA-8), 2002.

[18] The Sysmark 2007 benchmark. http://www.bapco.com/products/sysmark2007preview/index.php

[19] J.V. Castell, M.J. Gmez-Lechn . In vitro methods in pharmaceutical research. Academic Press, 1997.

[20] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms*. MIT Press, Third edition, 2009.

[21] J.T. Russell, M.F. Jacome. Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors, IEEE ICCD 1998.

[22] D.A. Patterson, and D.R. Ditzel, The case for the reduced instruction set computer. SIGARCH Comput. Archit. News 8(6), 25-33, 1980.

[23] J.L. Hennessy, and D.A. Patterson. Computer Architecture, Fourth Edition: a Quantitative Approach. Morgan Kaufmann Publishers Inc, 2006.