Information processing at work

On energy-aware algorithm design

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. In this paper we propose 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 this methodology is robust and can be used in many settings. We thus adopt it to study the impact of computation and pattern of memory accesses onto the energy-profile of an algorithm when executed on different processors and architectures, thus achieving some surprising insights on green algorithm design.

Keywords: power consumption models, Energon, energy-profiling of algorithms.

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. [18]). But 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 can only associate an average power consumption to complex entities such as the intermediate language opcodes, because they can trigger quite (energy-)costly side effects within both the VM and the hosting OS. 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.*"

Currently there are two main approaches to power-consumption modeling:

- Accurate analytic models based on the explicit measurement of the energy consumption of a set of benchmarks at the hardware level (see e.g. [10,12]). 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.
- Modeling of power at the simulation level (see e.g. [13,14]). Here power-consumption of the underlying virtual machine [11] or of the entire system is taken into account. This approach is limited to be used on simple codes to be profiled.

Certainly the energy consumption of programs 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 [19], 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. Since 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. So it is not surprising that [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.*" But literature yet misses an "abstract" model, akin to the Random Access Model (RAM [17]), and a methodology, akin to the asymptotic analysis of computational complexity, that accounts for energy issues in algorithm design and implementation.

In this paper we try to contribute to these issues by presenting a *theory* and a *methodology* for energy-aware algorithm design which is independent of the underlying system. We pursue our goal via the following 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!).
- Second, 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, 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). 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 then validate the Energon by evaluating the energy-profile of some well-known algorithms (binary search and a few sorting algorithms), showing that it is robust and reflects what it is predicted by the RAM model when data fits in *one single* memory level.
- 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. In this case it should be possible to envelop experimental data between a function f multiplied by two constants (we will indicate $\Xi(f)$). 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. (For space constraints this issue is detailed in [20].)
- We finally extend our study to other types of processors, architectures and patterns of memory-accesses thus illustrating, as claimed in [3], that the relation between time- and energy-efficiency is much intricate. We quantify some of these differences and draw preliminary, nevertheless interesting, conclusions that speculate onto possible applications of *our theory and methodology* to energy-aware algorithm design.

II. 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 select 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 system [1] 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 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%. It features several programming libraries targeting almost all OS and programming environments. 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 rate of at least 10Hz and a comparable error range.

Given this HW, we proceeded to measure the current provided to the motherboard by acting on 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 AC into DC, would reduce the ability of detecting current variations of our ammeter. Moreover, we thought (following [10]) 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, 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.

We then designed two software systems (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 (data-collecting) software.

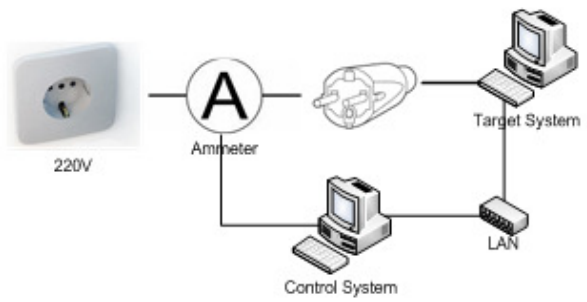

Figure 1. The structure of the experimental setup

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

The two programs used for performing measurements are open source and available on the CodePlex online site [9] (one of our main requirements is reproducibility).

The testing software performs a sub-sampling of the data and computes the average of the samples received from the ammeter sensor every 100ms (10Hz rate). We compute other values from the raw current readings and we are thus able to estimate the power consumption (in Watts and Joules) by using well known relations among the quantities involved. (We disregard the power factor $cos(\Phi)$ in the conversion, which leads us to over-estimating the power by a constant factor.)

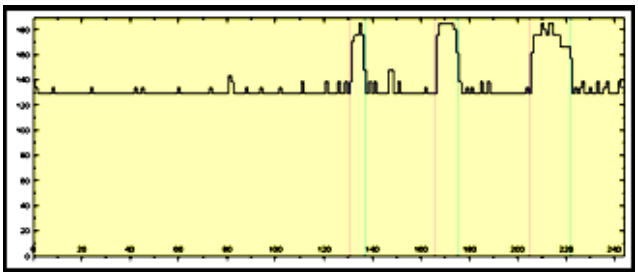

Figure 2. Idle operating system, and with a program running.

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 the total power consumption. As observed by others [6], idle computers consume a lot of energy!

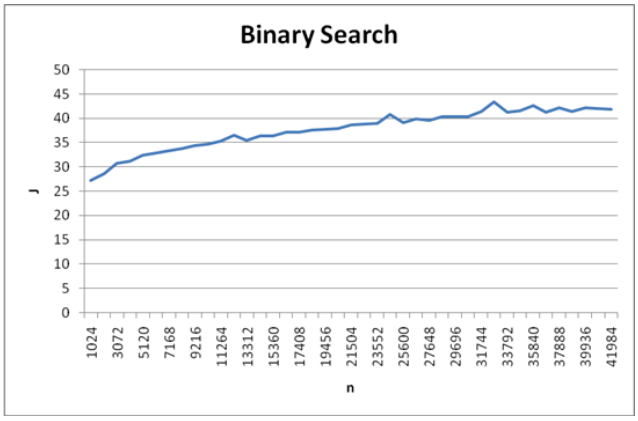

Figure 3. Energy consumption of a (log n)-time algorithm.

Then we examined the energy-usage of a more interesting algorithm: binary search. Figure 3 reports Joule-estimations which are obtained by iterating each search 2^{20} times, hence between 0.4 and 0.75 seconds. It's remarkable that the behavior of this algorithm, having computational complexity $\Theta(log\ n)$, can be measured through the power line, and with our simple methodology and HW-tools. The logarithm function fits the energy-consumption plot. In the subsequent sections, we will present other energy-plots, showing even more precise trends.

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 with respect to the underlying system running the experiment, thus playing the role of the unit-step in the RAM model, but in the energy-profiling setting. This is what we introduce in the next section.

III. THE ENERGON

It is well known that measurements taken in clean environments often differ from those obtained in the real world. In our case, 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 running our program, distinguishing them from the background noise caused by the eco-system itself. Two different runs of the same program may well lead to different results, so we need a method that distinguishes the program's contribution from the noise.

Our 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. Moreover sometime occur spikes that are related to some services running, and 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* [16]. 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. The energy-cost of this reference-program is then used as a meter to normalize 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), right before the real experiment is run.

Figure 4 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 the same way, 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 the following sections, where we study the impact of the pattern of memory-accesses onto algorithmic energy-consumption. The case of disk-accesses is deferred to subsequent studies.

```
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++.

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 Greek letter ε 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 from 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 about its *stability*. We proved that iterating the Energon program *h* times, the energy consumption increases correspondingly (with 0.01 standard deviation).

Given the Energon, we can again consider the binary–search algorithm, and evaluate its energy-profile in terms of

[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 observed 177.25*j* absorbed in avg with a stddev of3.46*j*.

Energon. The obtained plot (not shown) is smoother than the one in Figure 3, still retaining its logarithmic shape.

IV. 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.

We considered three well-known sorting algorithms: merge-sort, heap-sort, and quick-sort [17]. 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 statistical significance. 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 these issues are in the scope of our work, in this early stage of our validation process we prefer to stick onto the more regular measurements of C++ programs.

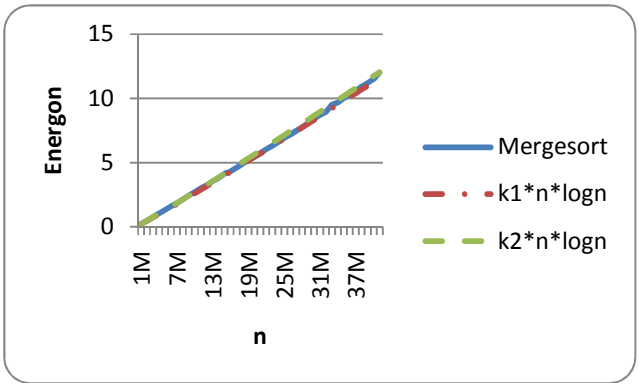

Figure 5. Merge-sort energy consumption in Energon

Merge sort is an optimal comparison-based sorting algorithm that requires $\Theta(n \log n)$ time in the worst and average case. As shown in Figure 5, the energy consumed by the algorithm as a function of its input size fits the classic asymptotic time-complexity.

We also tested heapsort and noticed that it requires more than twice the energy of merge-sort. This is due to its *random* pattern of memory-accesses, which implies more waiting time by the CPU because of cache-misses. Given these results, the next section will address the energy-issues concerning the non-negligible impact of the pattern of memory-accesses onto the energy-profile of an algorithm.

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 6 shows the energy-profile of Quicksort expressed in Energon. We fitted those data using the least square method, and the best fit was $f(n) = n \log n + \frac{n^2}{512}$.

This function is different from the one obtained for Merge-sort and heap-sort, as it includes both terms predicted by the worst- and average-case asymptotic analysis.

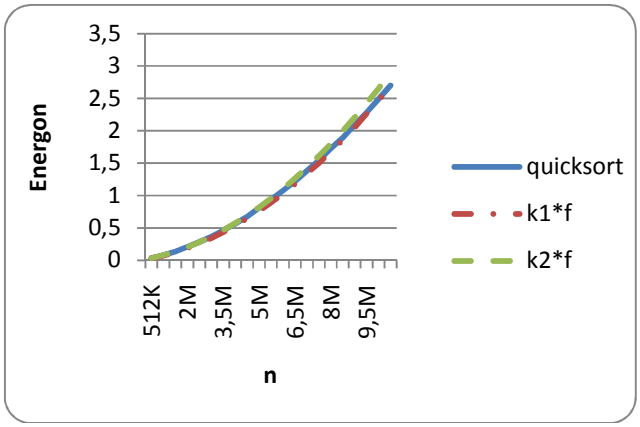

Figure 6. Quicksort energy consumption in Energon.

V. Experimenting different architectures

We tested two (radically different) architectures both with 2Gb RAM: an AMD Athlon 64 X2 Dual Core 4200+ 2.20GHz, and an Atom N230. We have executed all algorithms of the previous sections with the twofold goal of verifying that Energon results are confirmed across different architectures, and showing that the Energon allows for data comparison between different systems.

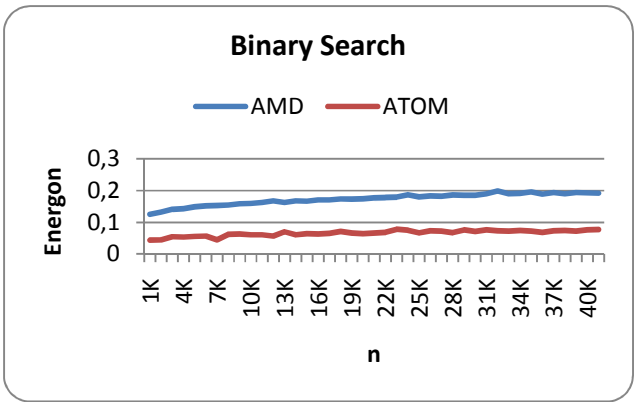

Figure 7. Binary search tested on two different platforms

First of all we compared the 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. Figure 7 reports the Energon-plots of binary search, results on the sorting show even larger savings. It can be noted that

data follow the same trend on the two architectures but their plots do not overlap. A possible explanation is that the Energon program measures only the CPU, and thus it does not take into account the energy-efficiency of the whole architecture.

When correlating the two architectures, we found that the two plots overlap perfectly if multiplied by a constant k. For binary search, we found k=2.8; for quick-sort k=3; for merge-sort k=6. 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. Thus we investigated whether there are constant contributions of the architecture which are 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 multiplicative k across the two different CPUs (2.8 versus 3) in spite of having quite a different time complexity and real running times; they do share, however, a *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 they incur into a different value for k (3 versus 6). Indeed, Mergesort produces the sorted output with 1/6 of the Energon amount on the Atom architecture wrt AMD one, while Quicksort only saves 1/3 . 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 results, the next section will dig into the effects of memory-access patterns on our Energon–based methodology.

VI. Experimenting Patterns of Memory Accesses

So far we have overlooked the cost of the memory accesses incurred by the executed algorithm, which however may deeply affect its time performance because current memories are hierarchical and consist of many levels (L1, L2, DRAMs, HDs, SSDs, etc.) each with its own capacity, bandwidth, latency. Accessing data in the memory levels closer to the CPUs is orders of magnitude faster than accessing data at the farthest levels. In this section we explore this issue in terms of energy-usage of the algorithm.

We designed a benchmark that, taken two parameters a and b, accesses an integer array I[1,n] in blocks of size b, skipping a blocks for each one read, and wraps around whenever I's end is reached. We experimented n and b power of two, and we took a as an odd number so to guarantee that the benchmark accesses all integers in I. It is clear that by varying <a, b> we can explore the issues of: sequential vs random patterns of memory accesses, and the single/blocked access to data. In addition to this, we instantiated the benchmark in three different ways to estimate the impact that FPU can have on the energy-cost of the algorithm: one (called DL) uses a doubly-nested loop (one for a and the other to sum the b-block integers) using mult/div to compute the positions of the items to be summed; another (called DLint) equals DL but avoids the use of the FPU unit and thus proceeds with just sums and bit-masks; finally a third approach (called SL) uses a single loop by adopting if-statements, sums and bit-masks.

Figure 8 shows the Energon cost of our tests. First we observe that as b increases the energy cost reduces, thus showing that block-wise access to data is worth of more than an order of magnitude energy-saving. Second, increasing a, and thus the *randomness* of the memory-accesses, determines an increase in the energy costs especially when working on large datasets. Surprisingly, all costs depend on the *implementation* of the algorithm, and thus on whether we use single or double loop, and if we deploy the FPU or not. The change in Energon may increase of 25%, but vanishes as b increases.

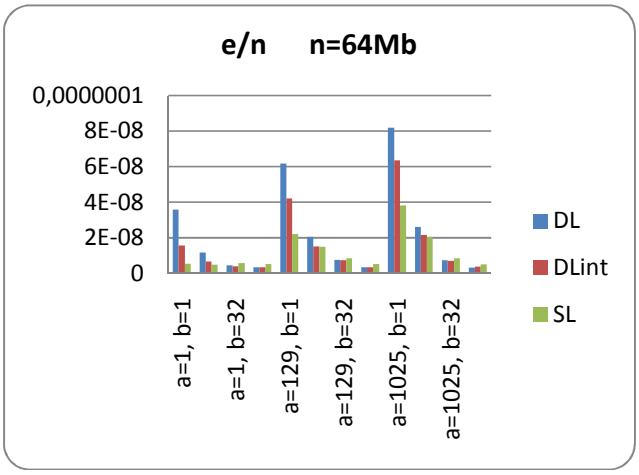

Figure 8. Energy required for accessing data in memory.

Figure 9 shows the Energon per time unit used by SL for various array sizes. We notice that we have fluctuations in the bars and they increase as the input size increases, thus the Energon subtly takes into account the cost of the memory-accesses which is obscured by the time complexity alone. This chart therefore shows that two algorithms may have same time efficiency but different energy costs because they process data of different size and/or they deploy different patterns of memory accesses.

We believe that our results provide some insight into the claim in [3] that "the relation between time- and energy-efficiency is much intricate".

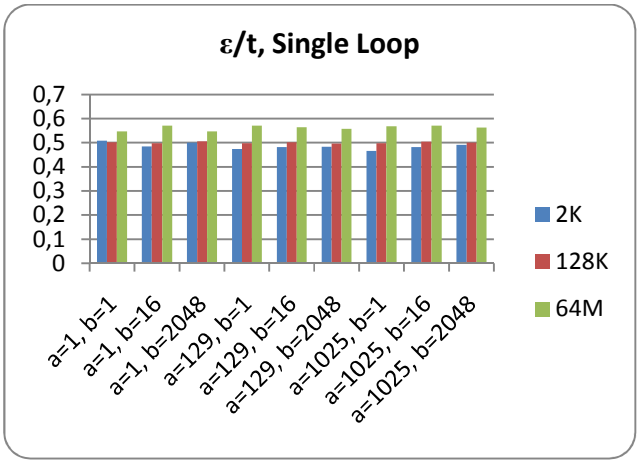

Figure 9. Energy required for processing arrays with different sizes.

VII. Experimenting Many-Core computations

So far we have discussed energy measurements of sequential algorithms, comparing our results against the RAM model and using CPU-bounded computations with varying access patterns. However, proper exploitation of features like superscalar execution and multiple cores deeply affects modern CPUs performance and energy efficiency. There is a lack of theoretical models relating energy efficiency with parallelism exploitation. In this section we analyze the Energon measure in many-core architectures.

Ideally, if every Joule is useful to the computation, the energy E_a required for executing an algorithm on a given input should be constant independently of the number p of computing entities employed. This is analogous to the ideal model in which the completion time T_c is $1/p^{th}$ of the time T_1 required by a single processor. The Amdahl's law corrects this view by stating that the completion time must also take into account an intrinsically-sequential part of the computation:

$$T_c(p) = T_s + T_1 / p$$

A similar consideration can be made for energy consumption:

$$E_c(p) = T_c(p)(W_o + W_p\, p)$$

where W_o is the average power due to the infrastructure overhead (the communication network, the common hierarchy of memories and the peripherals which tie up the cores or processors into a system, cooling fans), and W_p is the average extra-power required by one core to perform the computation with respect to the idle state. In the ideal case we have

$$W_o=0,\ T_s=0,\ E_a=T_c(p)W_p p=T_1 W_p.$$

We define two reference abstract architectures, called CC and IC, both multicore CPUs with p cores. The CC (core-cost) architecture sets $W_o=0$; the IC (infrastructure cost) architecture sets $W_p=0$ (so there is no difference between idle and not-idle energy consumption for a core). This way the CC model is an ideal architecture from the energy viewpoint, it only uses energy to execute the algorithm, and thus it represents a strict lower bound to its energy consumption. The IC architecture models "monolithic" parallelism with all available cores/nodes always powered on, so it is an upper bound to the energy used by a program on a real CPU. The two models collapse to the same energy cost if the execution uses all cores available. Real multi-core/processor architectures behave in between of the two extremes. Figure 10 left reports the case of $T_s=0$, where "Real" refers to a synthetic plot obtained by weighting 50% both computation (CC) and infrastructure (IC) costs.

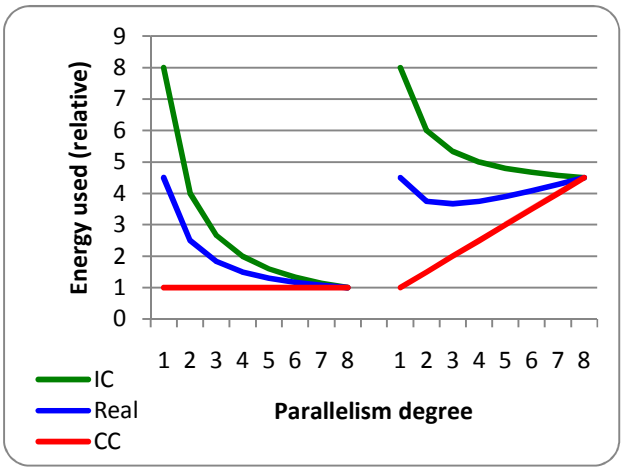

Figure 10. Relative energy consumption of the IC and CC models with respect to a realistic case; left assumes $T_s=0$, right assumes $T_s>0$.

If we consider the Amdahl's law with $T_s > 0$, we obtain that the IC and CC bounds behave as in Figure 10 right. Real architectures will have an energy cost with a decreasing term proportional to $1/p$, and an increasing one proportional to p. Here we ignore arbitrary overhead terms into the algorithm, and any variation in the infrastructure energy due to the computation. This tells us the existence of a *minimum* which should be determined in order to obtain the best energy-efficiency for a given algorithm running on a given architecture.

Given these considerations, we measured the energy-profile of a simple linear-scan over a large array on a quad-core system with hyper-threading (seen as an eight-core CPU by operating systems). We analyzed parallel versions of the linear scan using p=1,2,4,8 cores. The algorithm partitions the input array in p segments and scans them concurrently, so that its theoretical speedup is p. In practice, memory contention effects cause a slowdown from the ideal performance (measured speedup is 3.81 on 8 cores). The effect is well modeled with Amdahl's law as a sequential

overhead with T_s = 3.59e-8 and T_1=1.99e-7, as shown in Figure 11.

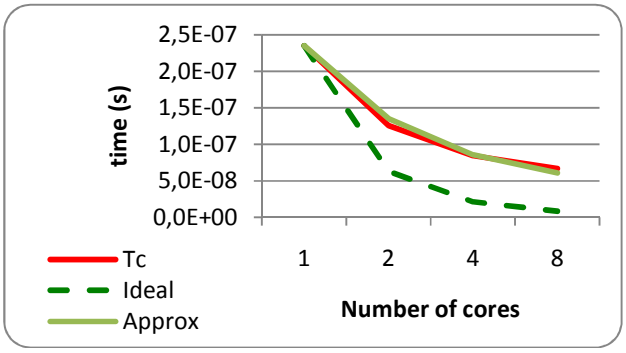

Figure 11. Linear scan ideal and real completion time (Tc(p)) with respect to the number of cores used, plus Amdahl –like approximation.

Figure 12 shows the *average power consumption* in Watts achived by the linear scan using different numbers of cores and array sizes. Exploiting parallelism increases the power consumption rate, but reduces the running time, and the overall result is a lower energy consumption.

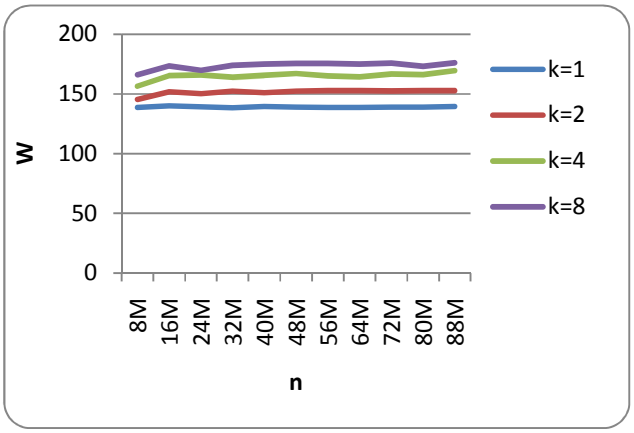

Figure 12. Average power used in the four experiments.

We can compute the power cost of a single core from the data, and see that the real CPU is quite close to the abstract IC model. The Energon measure of the linear-scan (figure 13) confirms that the most efficient way to use this CPU is by deploying all cores together. The speedup obtained over sequential execution (namely 3.81) induce a three times less Energons on the 8-cores execution. So there is a "dis-alignment" between time and Energon.

From the plot we get two important insights: current system architectures are overwhelmed by fixed energy overheads and underuse of available resources can be expensive from the energy standpoint. We further observe that minimizing the completion time alone does not optimize energy, but also the parallel work (time x processor product) may fail if the number of processors is unwisely chosen. The Energon count reflects these effects in a concise measure that allows tuning the system easily. With an increasing number of cores/processors, or when the real architecture gets closer to the CC model, the Energon measure can provide a robust tool to optimize the system's energy efficiency, still detecting any consumption trends increasing with parallelism (see Figure 10 right).

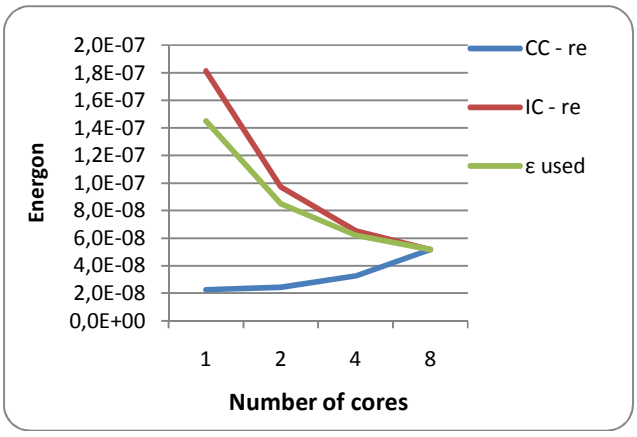

Figure 13. Energon measure of the linear scan compared to the CC and IC models built on the real completion time.

VIII. Program Energonomy and Future Work

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 important 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 [8]). Thus we expect that Python programs will consume more energy than their C/C++ equivalent. Same arguments hold for virtual machines, such as Java or Mono (.NET implementation for Linux) with respect to C coding. We will perform more precise evaluations in the near future to show that scripting is easier and more flexible, but it has a hidden energy-price to pay.

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!) 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, this methodology provided us with some quantitative hints about the impact of many-cores computations and the pattern of memory-accesses which may even be of some order of magnitudes. So any programmer should take them into account when designing a green-software (what we call program *energonomy*).

In the near future we plan to dig into the issue of the pattern of memory accesses by considering Hard and Solid-State disks, as well as several other multi-core architectures.

ACKNOWLEDGEMENTS

This work has been originated by several discussions with Antonio Calabrese and Maurizio Davini to whom we express our gratitude. We also would like Carmelo De Lauria and Daniele Mazzei for discussing with us the experimental method using their background in biomedical engineering, and in particular in cell cultivation.

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

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

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

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

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

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

[12] 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.

[13] 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.

[14] 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.

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

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

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

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

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

[20] A. Cisternino, M. Coppola, P. Ferragina, D. Morelli. Information processing at work: On a theory of experimental algorithm complexity. Technical Report # 10-13, Dipartimento di Informatica, University of Pisa, July 2010.