Energy consumption model over parallel programs implemented on multicore architectures

Ricardo Isidro-Ramírez
Instituto Politécnico Nacional
SEPI-ESCOM
M´exico, D.F.

Amilcar Meneses Viveros
Departamento de Computación
CINVESTAV-IPN
M´exico D.F.

Erika Hernández Rubio
Instituto Politécnico Nacional
SEPI-ESCOM
M´exico D.F.

***Abstract*—In High Performance Computing, energy consumption is becoming an important aspect to consider. Due to the high costs that represent energy production in all countries it holds an important role and it seek to find ways to save energy. It is reflected in some efforts to reduce the energy requirements of hardware components and applications. Some options have been appearing in order to scale down energy use and, consequently, scale up energy efficiency. One of these strategies is the multithread programming paradigm, whose purpose is to produce parallel programs able to use the full amount of computing resources available in a microprocessor. That energy saving strategy focuses on efficient use of multicore processors that are found in various computing devices, like mobile devices. Actually, as a growing trend, multicore processors are found as part of various specific purpose computers since 2003, from High Performance Computing servers to mobile devices. However, it is not clear how multiprogramming affects energy efficiency. This paper presents an analysis of different types of multicore-based architectures used in computing, and then a valid model is presented. Based on Amdahl's Law, a model that considers different scenarios of energy use in multicore architectures it is proposed. Some interesting results were found from experiments with the developed algorithm, that it was execute of a parallel and sequential way. A lower limit of energy consumption was found in a type of multicore architecture and this behavior was observed experimentally.**

***Keywords*—Energy Consumption; Multicore Processors; Parallel Programs; Amdahl's law**

I. Introduction

With the emerging of High Performance Computing, it was possible to carry out sophisticated calculations that before used to take a far longer time to be accomplished. This kind of computing has permitted to generate processes with much more elaborated and complex operations. Usually, HPC servers need high quantities of energy per unit time to work, for example, to implement a machine which execute a billion operations per second, or, in others words, a machine that would reach an exaflop, would require a big energy budget, comparable with the energy requirements of a town. Hence, energy consumption in data centers and supercomputers, is a topic which before did not have any importance, but is currently becoming more important every year.

Since 2004, multicore processors have been the object of interest of researchers that aim to exploit the potential of a die with more than one processing unit [1]. Clearly this is not only a case of study for servers and mainframes, because it is common to have multicore processors in mobile devices. This is an indicator that parallel programs are being used in various computing devices, from servers to smartphones.

A feature that makes a difference between mobile devices and other computer appliances, is that they are totally energy-bounded by a battery [2]. Across the years, battery technologies that feed devices have permitted to increase the capacity of these batteries without changing the size or weight of these components. However, mobile software developers often design and write applications which consider that the energy supplied is ideal, the battery is completely charged, and that usually it is not important to be aware of the energy impact caused by their applications. Neither are they worried enough by the fact that more hardware components being used and additional wireless requests will have a significant impact on the consumption of battery power.

A feasible solution for energy limitations is the introduction of parallel programming techniques that use efficiently all the computing resources available into modern smartphones. Actually, all industry is in fact using processors produced by companies such as Samsung, Qualcomm, Intel, etc. All these companies have been changing their hardware paradigms to offer more computing power into a single die; that includes the addition of independent processor chips joined by different mechanisms like shared cache, bridges, etc. In the market, the minimum number of cores present into a mobile multiprocessor is two. The reasons for this change are the constraints found in the traditional single-core paradigm, such as incremental heat and power dissipation [1].

The chipset landscape has a variety of multiprocessors divided by diverse features. Commonly, the designs are separated by their hardware architecture into symmetric and asymmetric [3], [4], but some researchers have also found another classification regarding their energy use features, e.g. a machine where processors can be turned off individually, or another machine where that is not possible [5].

Hardware classification separates processors by the computing capacity of their different cores. A sign of this class of processors is the presence of digital signal processors, GPU's and CPU's working together inside a chip. When cores have different capabilities, our multiprocessor is named heterogeneous or asymmetric. Otherwise, when all cores are similar, the multiprocessor is classified as homogeneous or symmetric.

We can distinguish three types of multicore processors if we consider its energy behavior. The first is a multicore where all cores are turned on, independently of how many cores a task needs. When the process is finished, all the cores will be turned off. This behavior implies that the same amount of energy is required to operate one or more cores in the processor. The second case is observed when the energy spent is proportional to the number of active cores; hence if a process uses one core the processor consumes a unit n of energy, if process uses two cores, it consumes $2n$ units of energy, and so on. In the third case the processor has two operational voltages, one per each core, and another that feeds the entire processor. When all cores are turned off, each one of them operates in a energy state called *idle*, in addition a base voltage that feeds the other components of the processor is also present. Those different architectures are widespread in different appliances. In the traditional computing domain, symmetric architectures clearly dominate most of the market. But in mobile appliances, thanks to their low-power advantages that they offer [6], asymmetric chips are widely installed on them. In fact, asymmetric processors are even being used in server and mainframe environments due to these energy efficiency features [7].

Several authors have proposed models of energy consumption for multicore architectures. However, the experimental results have not matched the models, ie, the analysis of the models show energy savings [9], [11], [12], while in some experiments shown that programs on multicore architectures spend more energy when working with multiple threads [8], [15]. We believe that this contradiction is due to the models do not consider the types of processors, concerning the use of energy. This paper presents an analysis of different types of multicore-based architectures used in computing, and then a valid model is presented. Based on Amdahl's Law, a model that considers different scenarios of energy use in multicore architectures it is proposed.

This paper is organized as follows: Section II contains a brief review of some papers that have studied energy and power consumption in multiprocessors. Some of these papers deal with an adequate description of how power is used depending of many factors. Section III presents a description model based on the power using the muticore processors. The model has three stages, the last stage is the general, though the first two we consider necessary to understand the behavior of the model. Section IV presents some experimental results. These experiments are running the benchmark Linkpack in multicore processors. Finally, in Section V the conclusions of this work are presented.

II. Related work

Research over energy enhancements in multicore architectures, particularly about Amdahl's Law extensions, is focused on finding energetically sustainable architectures , using options like a set of rules that conforms a framework [5], or using techniques of CPU management, such as core offlining [8] and finding voltage and frequency optimal operating values of a multicore system [9].

Amdahl's Law has been used by some authors to model performance in computing scenarios [10]. Using their ideas to decide if multithreading programming gives performance and energy benefits, some of them get optimistic results [9], [11], [12], and others find pesimistic results when an enhanced energy consumption by adding threads is expected [8]. A discussion is centered in the fact of how many cores are needed to work properly, having in mind that more cores working in parallel represent execution speed, but require more energy to function. We pretend to use our model to describe energy consumption only in symmetric computers for the moment. This will be the base for our next study that we shall glimpse as future work.

Cho and Melhem [5] studied the mutual effects of parallelization, program performance, and energy consumption. Their proposed model was tested on a machine that could apply core offlining. With this work, they predict that more cores combined with a high percentage of parallel code in a process, helps to reduce energy use.

Basmadjian and de Meer [6] worked in the design of a software-based model of power consumption for multicores, and they suggested that it is important to bear in mind that the presence of more that one processing unit, has a direct relation in power behavior, because working with multiple cores implies computing sharing. They also mentioned that a hardware-level measure is not trivial, when the presence of a high quantity of circuits inside every multiprocessor is considered.

Hill and Marty [11] studied Amdahl's Law impact over various processors chip distributions, using homogeneous and heterogeneous dies. They suggested the idea of designing a chip that gives priority to global chip performance over individual core efficience. Even if a chip is composed of several cores, some of these could work together in order to offer a higher sequential performance; they concluded that an asymmetric multicore has better performance results against a symmetric multicore.

Sun and Chen [13] showed that multicore architectures are not limited properly by Amdahl's predictions, but a real drawback would be the disparity of technology improvements between CPU speed and memory latency. Thus, they conclude that there are not significant limitations in scalability in number of cores, but more research is needed to overcome memory limitations.

About using benchmark, Oi and Niboshi [14] made a study using two different CPUs and platforms. They did, in particular, a power measure for individual instruction and complete workload level. They studied the power behavior of a server, varying the operating clock frequency. The other interesting area that they tackled was performance, reflected in parameters like transaction response time and so on.

Isidro, Meneses and Hernández [15] showed previously to this paper that an energy-study design depends of many more factors such as hardware components that participate in application execution. They also proved that is possible to model and predict energy usage for an application, knowing the quantity of parallel and sequential code portions that a program has. This paper is process-oriented, as opposed to other studies which give more emphasis to hardware-oriented strategies. Finally, in this paper, we explain that there exist two different multicore architectures and three different energy

usage models in multicore. Based on this affirmation, from Section 3 to Section 6, mathematical models of each one are explained.

III. Energy consumption model

In this paper the model of energy consumption for multicore processes is proposed. This model is an extension of Amdahl's law considering the types of processors for its energy behavior. It is well known that the relationship between power (Watts) and energy (Joules) is $Joules = Watts \times Seconds$. So, the idea is to have the power model and combine it with Amdahl's time model.

From [10][16], the speedup of a program to solve a problem of size N over p processing units $\psi(N,p)$ is divide the time to solve the problem of size n in a processor $T(N,1)$, between the time the same problem in p processors $T(N,p)$.

$$\psi(N,p) = \frac{T(N,1)}{T(N,p)}.$$

Such that, the time to solve a problem of size n by a processor, $T(N,1)$ or simply $T(N)$, can be split into the inherently sequential part $\sigma(N)$ and potentially parallel part $\rho(N)$.

$$T(N,1) = \sigma(N) + \rho(N). \tag{1}$$

And, the time to solve a problem of size N over p processing units is represented by

$$T(N,p) = \sigma(N) + \frac{\rho(N)}{p} + \kappa(N,p), \tag{2}$$

where $\kappa(N,p)$ is the overhead obtained by dividing the potentially parallel part in p processing units.

Consider a multiprocessor with n cores. Then CPU power is represented by W_{CPU}, and measured in Watts. Independently of the power usage architecture of a processor, it is possible to represent the entire power consumption by the addition of three parameters: base power, power of active cores, and idle power for all cores. Thus, the power of a processor with N cores with active cores p is:

$$W_{CPU}(N,p) = W_{base} + pW_{active} + nW_{idle}, \tag{3}$$

where $p < n$. Depending on the distribution of the process resources into a multiprocessor, these parameters could be significant or not. For example, (4) represents power required by a sequential program in a processor with n cores.

$$W_{CPU}(N,1) = W_{base} + W_{active} + nW_{idle}. \tag{4}$$

Now, having time and power expressions, it is possible to model energy consumption in Joules(J). Depending if the program is sequential (one thread), or parallel with p threads, two cases are presented.

The energy required to solve a problem of size N on a single core, $J(N,1)$, can be split into the inherently sequential part $J_\sigma(N,1)$ and potentially parallel part $J_\rho(N,1)$:

$$J(N,1) = J_\sigma(N,1) + J_\rho(N,1). \tag{5}$$

The energy required for the sequential and parallel portions are modeled with (6) and (7) respectively:

$$J_\sigma(N,1) = J_\sigma(N) = \sigma(N)\left(W_{base} + W_{active} + nW_{idle}\right), \tag{6}$$
$$J_\rho(N,1) = J_\rho(N) = \rho(N)\left(W_{base} + W_{active} + nW_{idle}\right), \tag{7}$$

Now, the energy required to solve a problem of size N on p cores in a processor with n cores, where $p < n$, is represented in (8):

$$J(N,p) = J_\sigma(N,1) + J_\rho(N,p) + J_\kappa(N,p). \tag{8}$$

The sequential part is the same that (6). The energy required to execute the parallel portion over p cores is:

$$J_\rho(N,p) = \frac{\rho(N)}{p}\left(W_{base} + pW_{active} + nW_{idle}\right). \tag{9}$$

The next step for the model is to represent the energy speedup, $\psi_J(N,p)$, that a parallel program will reach against their sequential version.

$$\psi_J(N,p) = \frac{J(N,1)}{J(N,p)} = \frac{J_\sigma(N) + J_\rho(N)}{J_\sigma(N) + J_\rho(N,p) + J_\kappa(N,p)} \tag{10}$$

We can consider three general scenarios for the power of a multicore processor. Equation (3) is a general schema of power consumption was presented, showing the three parts that generally compose the entire CPU power consumption. Now, specific formulae for each architecture will appear. The first scenario consists in a machine model where all cores are turned on, independently of the task size and number of threads used. In this case, CPU power uses only a base power value, and *idle* and *active* states are discarded, as show in (11). The second scenario works with just the necessary cores that a process needs, the rest of cores remain turned off. Hence, CPU power usage is represented by p times the *active* power of a unitary core, as show in (12). The last architecture is found when a processor has baseline power and also an individual operation power per core. In that case, the CPU power is the sum of base power, *idle* power and *active* power, as show in (13).

$$W_{CPU} = W_{base}, \tag{11}$$
$$W_{CPU} = pW_{active}, \tag{12}$$
$$W_{CPU} = W_{base} + pW_{active} + nW_{idle}. \tag{13}$$

A. Model applied to constant power usage multicore

The first scenario in our classification that we have observed is any multicore architecture where energy consumption is directly proportional to the number of cores that are turned on (from one to *n*), i.e. when the power using the multicore processor is constant. Its means that the W_{CPU} is described in (11)

$$\begin{aligned}
\frac{J(N,1)}{J(N,p)} &= \frac{J_\sigma(N,1) + J_\rho(N,1)}{J_\sigma(N,1) + J_\rho(N,p) + J_\kappa(N,p)} \\
&\leq \frac{J_\sigma(N,1) + J_\rho(N,1)}{J_\sigma(N,1) + J_\rho(N,p)} \\
&\leq \frac{\sigma(N)W_c + \rho(N)W_c}{\sigma(N)W_c + \frac{\rho(N)}{p}W_c} \\
&\leq \frac{W_c(\sigma(N) + \rho(N))}{W_c(\sigma(N) + \frac{\rho(N)}{p})} \\
&\leq \frac{\sigma(N) + \rho(N)}{\sigma(N) + \frac{\rho(N)}{p}}
\end{aligned} \tag{14}$$

If we consider f as the percentage of inherently sequential execution time [10][16]:

$$f = \frac{\sigma(N)}{\sigma(N) + \rho(N)}. \tag{15}$$

And if we substitute (15) in (14), then we obtain

$$\frac{J(N,1)}{J(N,p)} \leq \frac{1}{f + \frac{1-f}{p}}. \tag{16}$$

We can see that the resulting inequality (16) is Amdahl's Law. Where f represents the percentage of inherently sequential execution time [10][16].

For this case, the model of constant energy, and based on (3), it is only necessary a base level power for each core, since an *idle* power is non existent here. Equation (13) describes adequately this multicore behavior. Figure 1 represents the speedup curves depending of the quantity of sequential code f that a process contains. The graph illustrates that speedup increases as a combination of high potential parallel code and a large amount of cores. The acceleration shown in Figure 1 for $f < 0.2$ means that you can have energy savings when more cores are used in a multicore processor to solve a parallel process.

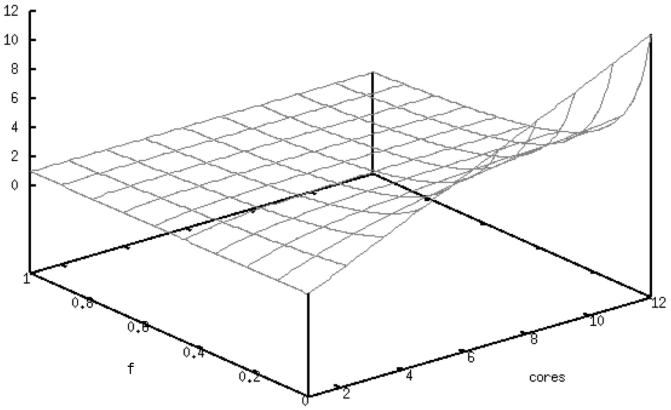

Fig. 1: Energy speedup for a constant energy usage, depending of the number of cores and the sequential portion f of the process

B. Model applied to offlining multicore

In the second scenario, it is not considered the base power or idle power, ie, power is proportional to the number of active cores. This scenario is an offlining energy architecture. If an offlining energy architecture is present in the energy architecture, then only the active power for each used core exists. The total power is modeled with (12), and then, the speedup is given by:

$$\begin{aligned}
\frac{J(N,1)}{J(N,p)} &= \frac{J_\sigma(N,1) + J_\rho(N,1)}{J_\sigma(N,1) + J_\rho(N,p) + J_\kappa(N,p)} \\
&\leq \frac{J_\sigma(N,1) + J_\rho(N,1)}{J_\sigma(N,1) + J_\rho(N,p)} \\
&\leq \frac{\sigma(N)W_{cpu}(1) + \rho(N)W_{cpu}(1)}{\sigma(N)W_{cpu}(1) + \frac{\rho(N)}{p}W_{cpu}(p)} \\
&\leq \frac{\sigma(N)W_{active} + \rho(N)W_{active}}{\sigma(N)W_{active} + \frac{\rho(N)}{p}pW_{active}} \\
&\leq 1.
\end{aligned}$$

Hence, the behavior of this scenario tells us that the lower bound for the energy used in a process with this kind of multicore, corresponds always with the energy used by a sequential execution, expressed in (18):

$$\psi_J(N,p) = \frac{J(N,1)}{J(N,p)} \leq 1, \tag{17}$$

$$J(N,1) \leq J(N,p). \tag{18}$$

Therefore, in this scenario no energy savings when parallel programs running on multicore processors. However, having a lower limit can be referenced to find the best energy performance of a parallel program.

C. General case: Model applied to multicores with base and individual core power

The third energy scenario corresponds to all multicores when there exists a base power that feeds the entire chip, and adds more power depending of the number of cores that a process requests. Additional to these power values, each core has an *idle* power that must be taken into account. Consider that all cores are symmetric for this model.

$$\begin{aligned}
\frac{J(N,1)}{J(N,p)} &= \frac{J_\sigma(N,1) + J_\rho(N,1)}{J_\sigma(N,1) + J_\rho(N,p) + J_\kappa(N,p)} \\
&\leq \frac{J_\sigma(N,1) + J_\rho(N,1)}{J_\sigma(N,1) + J_\rho(N,p)} \\
&\leq \frac{\sigma(N)W_{cpu}(1) + \rho(N)W_{cpu}(1)}{\sigma(N)W_{cpu}(1) + \frac{\rho(N)}{p}W_{cpu}(p)},
\end{aligned}$$

if $W_{CPU}(p) = W_{base} + pW_{active} + nW_{idle}$,

$$\Psi_J = \frac{J(N,1)}{J(N,p)} \leq \frac{1}{f + (1-f)(\frac{W_{cpu}(p)}{pW_{cpu}(1)})} \tag{19}$$

The energy speedup of this case depends on the power used in the execution of sequential process and power used in the execution of parallel processing, as show in (19).

Observe that depending on the values of power, the speedup graphics present variations. We can predict in an example with low values of base, idle and active power, a behavior like the one shown in Figures 2 and 3:

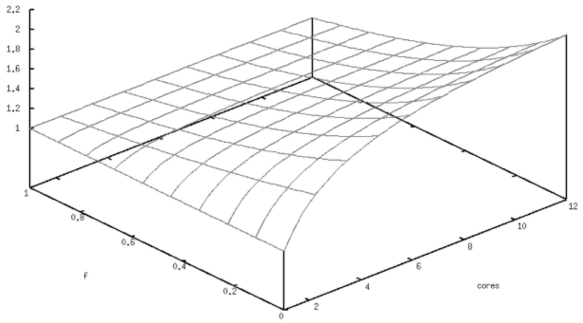

Fig. 2: Energy speedup for the third scenario, with $W_{active} = 10w$, $W_{base} = 1w$, $W_{idle} = 1w$ and n from 0 to 12

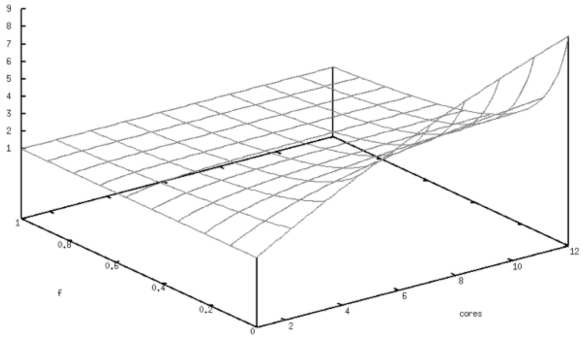

Fig. 3: Energy speedup for the third scenario, with $W_{active} = 2w$, $W_{base} = 40w$, $W_{idle} = 2w$ and n from 0 to 12

IV. Experimental results

To test the model, the benchmark Linpack was used to stress the multicore processor [17], [18]. Linpack is a measure of a computer's floating point rate of execution, that is determined by running a computer program that solves a dense system of linear equations. Despite the existence of other benchmarks that are focused on measuring diverse aspects of interest in software such as peak-use, complexity, cost per performance, among other aspects, Linpack was implemented because the issue of interest in this research is to stress the entire processor, including the floating-point unit.

In order to perform measurements, two devices were used to make the respective tests, these devices are listed below:

- Tablet Samsung Galaxy Tab 2 10.1, version GT-P5110, with Android OS 4.0 and 1.0 GHz OMAP4430 dual-core 45 nm ARM Cortex-A9.

This device correspond to mobile environment. However the device has multiple cores and it is possible to make an analysis based on execution time of an algorithm that could employ a variable quantity of resources, running in a parallel or a sequential mode. Also Linpack benchmark has been used in a server environment, but with few changes, it was programmed by us for a mobile scenario also.

Linpack in mobile version is able to execute vector-vector, vector-matrix and matrix-matrix products, and was developed under various multicore platforms. We made several implementations using POSIX threads and Android Java threads to generate parallel code. In this work, Linpack benchmark implementations were used over Android.

A. Working with Android

The tests were conducted in low-level Android libraries written in C and Java libraries used. The purpose of these comparisons is to detect the same energy behavior of the model regardless of whether the program is working layer, that is, if you work with programs that run directly on multicore or go through a virtual machine.

Inside every Android device there is a special implementation of VM named *Dalvik VM*, whose goal is to run Java applications on mobiles. The *Dalvik VM* is well suited for a portable scenario, offering a reduced energy consumption and a better performance, compared with Java Virtual Machine [19]. Other features of DVM that mobile devices are capable to use are low-memory requirements and delegation of tasks for the operating system, like memory and threads management.

However, we programmed Linpack, writing all the logic in Java Native Code or *pthreads* as a JNI module[1]. Remember that there are two C/C++ libraries, one of them is bionic, present in Android based systems, while the other set of functions is `glibc`, present in Linux based systems. The most profound characteristics that have permitted the use of bionic instead `glibc` in Android are the limited storage and a lower CPU speed. Actually, bionic is a lightweight library that comes with some bounds, compared with the GNU version.

Another issue to consider is that many operating systems are ruled by policies named CPU frequency schedulers, that scale the chip frequency up or down in order to save power. Operating frequency can be: Scaled automatically depending on the systems loads, chosen by the user, or managed in response to ACPI events. This point is crucial in mobile devices, because a daemon that puts our device in an idle state our device if unused is convenient. Some devices can even support more than one governor if it is permitted in a list defined by the manufacturer. The common feature in mobile governors is that usually they seek the least possible CPU usage and, as a consequence, they try to sleep the processor cores as much as possible.

Galaxy Tab 2 has implemented *Interactive* governor, other available governors for most of the Android devices are: *OnDemand, Performance, Userspace, Powersave, Conservative*, among others. The differences among them reside in the time lapse that one keeps working the multicore, and the performance requirements of the user or application executed. In this paper, we did not consider the effects of each one of these schedulers, and the experiments worked with the original governor installed by default.

The results of the experiments are shown next, where Linpack matrix products with one and two threads were executed. This operation has an $O(n^3)$ complexity, enough to obtain processor usage values near to 100% of processor use.

Both experiments were executed in the same platform. In order to compare energy behavior between Android JNI and pure Java, we prepared two applications that perform the same

[1] In this paper, we use JNI module as the code section that runs with *pthreads*

function, but were developed differently. The first application is an Android version that makes the heavier computing in a native language, and the second version does the entire job over Java.

Table I illustrates time and power required to execute the entire program, and the quantity of resources used for sequential and parallel sections of the program when it runs over one thread. Table II shows the same information for a program that runs in two parallel threads.

Problem size	Sequential section time	Parallel section time	Total time	Sequential section energy	Parallel section energy	Total energy
500x500	0.0173	10.649	10.66	0.067	6.207	6.275
1000x1000	0.0474	110.223	110.2	0.084	61.631	61.71
1500x1500	0.0848	427.094	427.1	0.078	236.12	236.2
2000x2000	0.1585	1065.523	1065.6	0.058	593.34	593.4

TABLE I: Time (in seconds) and energy (in Joules) values obtained with single precision matrix-matrix product, using JNI and Android with one thread.

Problem size	Sequential section time	Parallel section time	Total time	Sequential section energy	Parallel section energy	Total energy
500x500	0.0173	5.295	5.312	0.067	6.207	6.275
1000x1000	0.0484	55.737	55.78	0.085	61.854	61.94
1500x1500	0.0849	228.781	228.8	0.088	253.133	253.2
2000x2000	0.1298	611.147	611.27	0.144	670.552	670.6

TABLE II: Time (in seconds) and energy (in Joules) values obtained with single precision matrix-matrix product, using JNI and Android with two threads.

Table I and Table II show better performance when the program has been built in parallel. On the other hand, energy consumption has not the same effect, because the energy required grows if the program is multithreading.

The second experiment executes the same matrix-matrix algorithm but discarding the use of a native language. Hence, we only use Java to generate all the workload.

Problem size	Sequential section time	Parallel section time	Total time	Sequential section energy	Parallel section energy	Total energy
500x500	0.032	15.968	16.0	0.015	7.386	7.401
1000x1000	0.058	193.782	193.8	0.024	83.175	83.2
1500x1500	0.234	751.166	751.4	0.126	315.274	315.4
2000x2000	0.406	2028.374	2028.8	0.191	851.409	851.6

TABLE III: Time (in seconds) and energy (in Joules) values obtained with single precision matrix-matrix product, using pure Java with one thread.

Table III shows the execution time and energy used for matrix multiplication operations of different sizes, but using only one thread for all the work. Table IV shows also the time needed and energy consumed for the same program, but executed with two threads instead.

From Table I and III, we can see that the time and energy used to execute the task is lower for the JNI version of the program. Also, the potentially parallel section of the multiplication matrix algorithm is by far, the heaviest section of the program.

Problem size	Sequential section time	Parallel section time	Total time	Sequential section energy	Parallel section energy	Total energy
500x500	0.036	7.9824	8.018	0.016	7.501	7.517
1000x1000	0.056	98.04	98.09	0.022	86.048	86.07
1500x1500	0.201	400.959	401.1	0.137	337.563	337.7
2000x2000	0.407	1173.365	1173.7	0.203	962.097	962.3

TABLE IV: Time (in seconds) and energy (in Joules) values obtained with single precision matrix-matrix product, using pure Java with two threads.

From Tables II and IV, one may verify that parallelism gives benefits in both versions of the program, independently of the mechanism used to generate the code. Also, energy consumption for both programs is still lower when we use one thread to execute the program. Finally, while the performance gets speedups from 1.7x for 2000x2000 matrix product, to 2.0x for 500x500 matrix, we observe an opposite effect in energy consumption.

In Figure 4, observe that the least execution time is obtained with parallel applications, regardless if we use Java or JNI, and making a comparison among 4 program versions, a JNI implementation with 2 threads results the fastest of all.

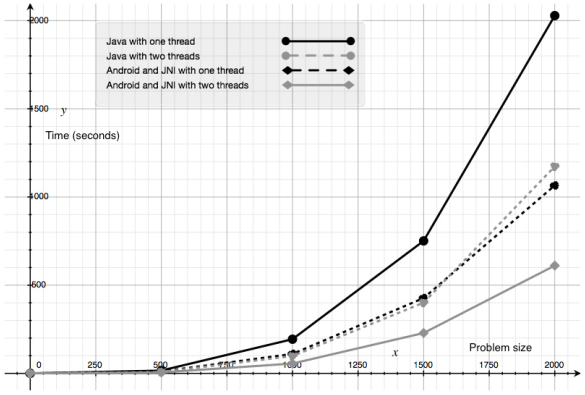

Fig. 4: Time values obtained with Linpack, using Java and JNI.

Figure 5 shows that both sequential programs (Java and JNI) produce energy savings, compared with parallel versions of the same program; this is in agreement with (18) about offlining multicores. Finally, both Figures highlight the gap between parallel and sequential results, which increases depending on the size of the problem.

The tool used to measure energy consumption and execution time was an application called PowerTutor [20], which estimates power usage by reading parameters available in Linux-based systems, like Android. For example, knowing the energy used by the processor is possible thanks to the parameters `/proc/stat` and `/proc/cpuinfo` that reveal the operating frequency of the chip and determine the power used during a task execution. Furthermore, one advantage of using this method to estimate power and energy usage resides in the difficulty to obtain power readings using a power meter in a very reduced environment. The other fact that supports this

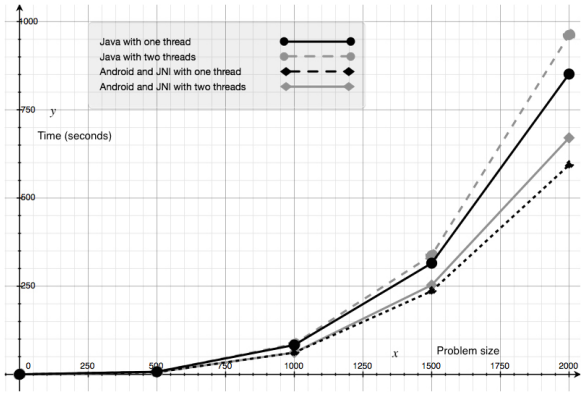

Fig. 5: Energy values obtained with Linpack, using Java and JNI.

method for our experiments is the low error rate of PowerTutor, ca. 6.27% [21].

V. Conclusion

One of the main contributions of this model, is the possibility of offering an explanation for the apparently contradictory results reported by several authors, because while some of them show energy savings using parallel programming in multicore processor [9], [11], [12], there is work which shows that energy consumption is punished by parallel techniques [8], [15].

Energy consumption contradictions exist because those models do not consider the existence of different power usage scenarios in multicore processors. It is very likely that the reason why authors finds energy savings with parallel applications is a processor that works with the first energy usage scenario exposed, where the multicore has a constant energy usage, independently of the number of turned on cores.

The model works for parallel programs running on multicore architectures. The model is tuned so that the number of processes that are used are less than or equal to the number of processor cores. However, with a number of threads higher than the number of cores n, it behaves as if working with n cores.

The model considers three main scenarios based on the power using the muticore processors. In the first scenario, where the processor uses the same power regardless of the number of active cores, Amdahl's law is obtained. Acceleration performance is proportional to the acceleration in energy consumption, which results in energy savings. In fact it is proposed in [6] that there is an energy benefit when computing resources are shared. In our model that behavior is reflected in the first scenario, where a constant power feeds the entire chip. This is to say that when a multicore processor has energy savings with a multithreading program, it is because all computing resources are shared in the processor.

In case where offlining scenario is present, we can observe the existence of a lower bound of energy consumption explained in this work, present when the sequential version of a program runs over the processor. When the number of threads of a parallel program increases, a reduction in execution time is obtained, while an increase in energy consumption will take place at the same time. Such behavior is seen in Figures 4 and 5, where we show that the lower bound in energy consumption corresponds to a totally sequential execution program. In this scenario no energy savings when parallel programs running on multicore processors. However, having a lower limit can be referenced to find the best energy performance of a parallel program.

In particular, in our experiment in android, the behavior of the lower bound of energy of the sequential version remains. With these experiments, we demonstrate that there is a notable improvement using native methods over Java, in performance and energy consumption, independently of the number of threads occupied. So, we claim that reducing virtual machine usage, gives a better performance in a high demanding computing application within the mobile scenario. We can infer that the behavior of power that has this processor is similar to the offlining scenario.

The third scenario is the most general and energy savings depend on the amount of resources that the multicore processor (19). The

$$\frac{W_{cpu}(p)}{pW_{cpu}(1)}$$

ratio that appears in (19), and CPU power (3) are the keys to our model to explain the behavior of parallel execution in different architectures of multicore processors. For example when $W_{cpu}(p) = W_{cpu}(1)$, general model (19) exhibits the behavior of the first scenario. And when $W_{base} + nW_{idle}$ is near to zero, general model (19) describes the behavior of the second scenario.

As future work, we will develop measurements in other computing environments, such as servers, laptops and desktop computers to verify the pattern of energy consumption.

Acknowledgment

R.I.R. thanks Section of Research and Graduate Studies (SEPI) of ESCOM-IPN, and Julieta Medina Garcia for scholarship support (SIP project 20131498). A.M.V. and E.H.R. thanks Cinvestav-IPN, and Section of Research and Graduate Studies (SEPI) of ESCOM-IPN, by the resources provided and the facilities for this work.

References

[1] S. H. Fuller and L. E. Miller, "Computing performance: Game over or next level?" *The National Academies Press*, pp. 31–38, 2011.

[2] P. Zheng and L. M. Ni, *Smart Phone and Next Generation Mobile Computing*. Elsevier Science and Tech, 2006.

[3] L. Carro and M. B. Rutzig, "Multi-core systems on chip," *Handbook of Signal Processing Systems*, pp. 485–514, 2010.

[4] D. H. Woo and H.-H. S. Lee, "Extending amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, no. 12, pp. 24–31, 2008.

[5] S. Cho and R. G. Melhem, "On the interplay of parallelization, program performance, and energy consumption," in *IEEE Transactions On Parallel and Distributed Systems*, vol. 21. IEEE, 2010, pp. 342 – 353.

[6] R. Basmadjian and H. de Meer, "Evaluating and modeling power consumption of multi-core processors," in *Future Energy Systems: Where Energy, Computing and Communications Meet (e-energy)*, 2012, pp. 1–10.

[7] Z. Ou, B. Pang, Y. Deng, and J. Nurminen, "Energy- and cost-efficiency analysis of arm-based clusters," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 115–123.

[8] A. Carroll and H. Heiser, "Mobile multicores: Use them or waste them," *5th Workshop on Power-Aware Computing and Systems HotPower '13*, 2013.

[9] S. M. Londoño and J. P. de Gyvez, "Extending amdahl's law for energy-efficience," in *Energy Aware Computing (ICEAC), 2010 International Conference*. IEEE, 2010, pp. 1–4.

[10] G. M. Amdahl, "Validity of the single-processor approach to achieve large-scale computing capabilities," in *AFIPS Conference*, vol. 30, 1967, pp. 483–485.

[11] M. D. Hill and M. M.R, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[12] R. Swapnoneel, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *4th Conference on Innovations in Theoretical Computer Science ITCS '13*, 2013, pp. 283–304.

[13] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *Journal of Parallel Distributed Computing*, vol. 70, no. 2, pp. 183–188, 2010.

[14] H. Oi and S. Niboshi, "Power-efficiency study using specjenterprise2010," in *Systems Conference (SysCon), 2013 IEEE International*, 2013, pp. 812–817.

[15] R. I. Ramirez, E. H. Rubio, and A. M. Viveros, "Energy consumption in mobile computing," in *Electronics, Communications and Computing (CONIELECOMP), 2013 International Conference*, 2013, pp. 132–137.

[16] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

[17] J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark; past, present and future," in *Concurrency and Computation: Practice and Experience, 15*, 2003, pp. 803–820.

[18] J. Dongarra, "Performance of various computers using standard linear equation software," in *SIGARCH Computer Architecture News*. ACM, 1992, pp. 22–44.

[19] K. Paul and A. Kumar, "Android on mobile devices: An energy perspective," in *In Computer and Information Technology (CIT), 2010*, 2010, pp. 2421–2426.

[20] "Power tutor: A power monitor for android-based mobile platforms."

[21] L. Z. Lide Zhang, B. S. Tiwana, R. P. Dick, and Z. Qian, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS*. IEEE, 2010, pp. 105–114.

[22] iOS Developer Series, *Threading Programming Guide*, Apple Inc., 2014.