ANN. MULT. GPU PROG.

Assessing Energy Consumption and Runtime Efficiency of Master-Worker Parallel Evolutionary Algorithms in CPU-GPU Systems

Juan José Escobar[1*], Julio Ortega[1], Antonio Díaz[1], Jesús González[1], Miguel Damas[1]

*Correspondence: jjescobar@ugr.es

[1]Dept. of Computer Architecture and Technology, CITIC, University of Granada (Spain)

Full list of author information is available at the end of the article

Abstract

Thanks to parallel processing, it is possible not only to reduce code runtime but also energy consumption once the workload has been adequately distributed among the available cores. The current availability of heterogeneous architectures including GPU and CPU cores with different power-performance characteristics and mechanisms for dynamic voltage and frequency scaling does, in fact, pose a new challenge for developing efficient parallel codes that take into account both the achieved speedup and the energy consumed. This paper analyses the energy consumption and runtime behavior of a parallel master-worker evolutionary algorithm according to the workload distribution between GPU and CPU cores and their operation frequencies. It also proposes a model that has been fitted using multiple linear regression and which enables a workload distribution that considers both runtime and energy consumption by means of a cost function that suitably weights both objectives. Since many useful bioinformatics and data mining applications are tackled by programs with a similar profile to that of the parallel master-worker procedure considered here, the proposed energy-aware approach could be applied in many different situations.

Keywords: Energy-aware workload distribution; heterogeneous parallel architectures; master-worker parallel evolutionary algorithms

1. Introduction

The availability of heterogeneous architectures enables codes to take advantage of different levels of parallelism to improve efficiency in terms of speedup. More specifically, in current heterogeneous architectures, graphic processing units (GPU) usually provide massive data-level parallelism (DLP) together with thread-level parallelism (TLP), which can also be used with multicore microprocessors that include superscalar cores (or CPU cores) implementing instruction-level parallelism (ILP). Nevertheless, in addition to enabling performance efficient parallel codes to be executed, heterogeneous architectures including GPU and CPU cores could also constitute an efficient approach for saving energy, once various challenges arising from CPU-GPU heterogeneous computing have been overcome. In fact, papers such as [1] consider the efficient cooperation of CPU and GPU to be an important requirement for achieving exascale performance. More specifically, in the future realm of exascale platforms, power and energy efficiency are central issues as indicated in [2] which cites a 2010 report by the US Department of Energy (DOE) which estimates that the annual power cost of operating an exascale system implemented with current technology to be about 2.5 billion dollars per year.

The current trend for tackling energy efficiency problems while preserving the improvement in performance is represented by nodes which include various superscalar multicore microprocessors (CPU cores) and GPUs (although other accelerators such as FPGAs, vector units, etc. are also possible). Some of the issues listed in [1] to be considered when developing performance-energy efficient codes for heterogeneous CPU-GPU systems are the size of CPU and GPU memories, CPU-GPU memory-bandwidth limitations, load balancing between the GPU and CPU cores, the overlapping of data transfer with CPU and GPU computation, and the characteristics relating to the amount of parallelism and branch divergence of the application in question.

Many of these issues have been previously analyzed for different applications. In this paper, we provide an insight into the development of a high-performing, energy-efficient workload distribution of parallel master-worker evolutionary algorithms in platforms where GPU and CPU cores are considered equal for sharing the application workload. In the type of evolutionary procedure considered here, the majority of the workload corresponds to the fitness evaluation of the solutions that constitute the population evolved through generations (iterations) of the algorithm. The way that solutions are allocated to GPU and CPU cores will therefore determine the efficiency in performance and energy behavior of the parallel algorithm once a suitable workload distribution among the available cores has been attained.

Following on from this introduction, Section 2 describes a cost function that takes into account the two goals of runtime and energy consumption to direct the search for efficient workload distributions; Section 3 explores our target application, i.e. parallel master-worker evolutionary algorithms, details our experimentation, and analyzes the results obtained; Section 4 describes the main contributions in this area and references the most relevant papers; and Section 5, outlines our conclusions.

2. A cost function for power and performance aware scheduling

The problem to hand is to find an efficient workload distribution among the available processors which is efficient both in terms of the speedup achieved and energy consumption. In order to reach this goal, the required scheduling procedure should be able to locate tasks on the available processors according to predictions about their computational cost and the corresponding energy consumption. The scheduling procedure therefore requires certain information about the performance and energy consumption characteristics of the processors and the other elements in the system where the tasks are executed. In Equation 1, the functions t*$(C_i, f_{j,k}(i,j=1,..p),(k=1,..,FL))$ and E*$(C_i, f_{j,k}(i,j=1,..p),(k=1,..,FL))$, respectively, correspond to models for the running time and energy consumption in terms of the workloads (in cycles) of the different p tasks, C_i $(i,j=1,..p)$, and the operating frequencies $f_{j,k}$ $(j=1,..p),(k=1,..,FL)$ of the p processors where the corresponding tasks have been allocated by the scheduling procedure.

$$\Delta t = \frac{t^*(C_i, f_{j,k}(i,j=1,..p),(k=1,..,FL)) - t_{MIN}}{t_{MAX} - t_{MIN}} \qquad (1)$$
$$\Delta E = \frac{E^*(C_i, f_{j,k}(i,j=1,..p),(k=1,..,FL)) - E_{MIN}}{E_{MAX} - E_{MIN}}$$

In order to select a processor and the corresponding frequency for a given task, the scheduling algorithm could use a cost function which takes into account both the energy and runtime objectives through Δt and ΔE in Equation 1. In our case, we propose the cost function $\Delta = a\Delta t + b\Delta E$ with $0 \leq a \leq 1$, $0 \leq b \leq 1$, and $a+b=1$. This cost function promotes allocations with low values of Δt and ΔE, and lower values for one factor whenever the other factor grows. Depending on the relative values of a and b, it is possible to place more importance on either lower runtime or lower energy consumption. It is therefore necessary to define the time and energy models, t^* and E^*, and the values of t_{MAX}, t_{MIN}, E_{MAX}, and E_{MIN} used in Equation 1. It is not easy to determine models that provide accurate enough time and energy predictions to enable efficient task scheduling procedures to be designed. In his article [O'Brien2017], the author presents a detailed survey of the different energy and power predictive models. Such models can be classified according to their level of abstraction, their accuracy, whether they predict power and/or energy, and their portability. The level of abstraction specifies whether all the components in a node are modelled independently or the corresponding dependences among them are modelled by considering either linear or non-linear dependences. An energy consumption prediction model could forecast instantaneous or average power and could take into account the dynamic power component, or both dynamic and static components. The energy model, meanwhile, can either be predicted from time and power models or from a specific energy model.

For example, one possible energy consumption model E^* could be derived from the power consumption equations corresponding to CMOS circuits that include the terms associated to capacitive, short-circuit and leakage power. It is common to assume that the capacitive term is the most significant and so power consumption in a processor can be approximated as:

$$Pow = \beta \times f \times V^2 \tag{2}$$

where parameter β represents the product of the number of transistors switching in the processor per clock cycle and the total capacitance load, f is the clock frequency of the processor, and V is the supply voltage. The energy E_i consumed by a given task i that requires C_i clock cycles in a processor with a supply voltage V_i can therefore be estimated from (2) by

$$E_i = \beta \times f \times {V_i}^2 \times {C_i}/_{f} = \beta \times {V_i}^2 \times C_i \tag{3}$$

Whenever a processor is idle, there is also a so-called indirect energy consumption that for a given processor k can be estimated by

$${E_k}^{idle} = \beta \times f \times {V_{idle}}^2 \times t_k \tag{4}$$

where V_{idle} is the supply voltage of the processor in its idle state, and t_k is the amount of time in which processor k has been in this state. The tasks are located on the processors included in a heterogeneous platform with p processors, P_j ($j=1,..,p$). Each processor P_j can operate at different voltage supply levels, $V_{j,l}$ ($l=1,..,\omega(j)$), corresponding to different clock frequencies $f_{j,l}$, ($l=1,.., \omega(j)$).

The parameters t_{MAX} and t_{MIN} in Equation 1 can be estimated from the maxima and minima values for the clock cycles C_i required to complete the estimated workloads of the different tasks $(i=1,\ldots n)$ and the frequencies of the available processors, $f_{j,l}$, $(j=1,..,p, l=1,.., \omega(j))$ as follows:

$$t_{MAX}=max(C_i\,(i=1,\ldots n))/min(f_{j,l},\,(j=1,..,p,\,l=1,..,\,\omega(j))) \tag{5}$$
$$t_{MIN}=min(C_i\,(i=1,\ldots n))/max(f_{j,l},\,(j=1,..,p,\,l=1,..,\,\omega(j)))$$

The parameter t_{MAX} is the time required by the task with the heaviest workload when it is executed in a processor running at the lowest frequency. In the same way, t_{MIN} is the lowest running time for the lightest task. It is also possible to estimate the energy consumption parameters, E_{MAX} and E_{MIN} as follows:

$$E_{MAX}= \beta \times n \times max(C_i\ (i=1,\dots n)) \times [max(V_{j,l},\ (j=1,..,p,\ l=1,..,\ \omega(j)))]^2 \qquad (6)$$
$$E_{MIN}= \beta \times n \times min(C_i\ (i=1,\dots n)) \times [min(V_{j,l},\ (j=1,..,p,\ l=1,..,\ \omega(j)))]^2$$

It is possible to take advantage of previous power and energy models through dynamic voltage and frequency scaling (DVFS) techniques and many papers proposing energy-aware scheduling are based on the availability of DVFS [3]. Nevertheless, processors usually implement energy management policies defined by the vendors that are not available at the user level, and alternative approaches to fit models to the experimental measures using a suitable regression method [4]. This is the approach followed in this paper.

3. Results on master-worker parallel applications

The experimental work described in this section has been conducted in a cluster node including two Intel Xeon E5-2620 v4 HT processors with eight cores per socket (therefore running up to 32 threads per node). The node also includes an NVIDIA Tesla K40m at 755 MHz with 12 GB of global memory, 288 GBytes/s of maximum memory bandwidth, and 2880 CUDA cores distributed into 15 SMXs (Stream Multiprocessors). In the following subsections, we shall describe the application executed in our experiments and the issues relating to the energy consumption measures, and analyze the experimental results.

3.1. Characteristics of the target application

The results of this paper can be applied to parallel programs whose dependence graphs include tasks T1, T2, ….., TN that can be executed in parallel after task T0. Once the tasks have been executed and then synchronized, task T0 is executed again to generate another set of parallel tasks T1, T2,…, TN, and so on. The runtime of task T0 is also negligible in terms of the runtime of each parallel task T1,..,TN. Many useful parallel applications follow this dependence graph. In this paper, we have considered a multi-objective evolutionary algorithm that has been parallelized according to a master-worker paradigm. In each generation, the fitness of individuals in the population must be evaluated in terms of a certain performance procedure that might require costly computation. More specifically, the evolutionary multi-objective procedure analyzed here is applied to solve a feature selection problem in a BCI application [4]. The population individuals correspond to different sets of features that define the components of the patterns to be classified. These sets of features must be evaluated by the accuracy of the classifier once it has been adjusted using the training patterns characterized by the selected features. The iterations required to train the classifier usually require a high amount of computing time. For example, in the runtimes for the steps of our multi-objective feature selection procedure analyzed with `gprof` [5], the fitness evaluation needs between 99.93% (with 120 individuals in the population) and 98.60% of the runtime (with 15,000 individuals). As fitness evaluation is completely independent for each individual in the population, a master-worker approach represents a very suitable alternative to efficiently parallelize this type of problem. Considering these characteristics, an energy-aware procedure to allocate the fitness evaluation tasks also affords an energy-efficient master-worker evolutionary algorithm because even when there is a very large number of individuals in the population, the percentage of runtime devoted to evaluating population fitness would be higher than 95%. Although it appears to be a very specific situation, many real applications tackled by evolutionary metaheuristics and executed in parallel on platforms including GPU and CPU cores match this profile.

The platform considered here includes two different kinds of processors among which the individuals in the population evolved by the evolutionary algorithm are distributed to compute their corresponding fitness. Thus, the workload scheduling is reduced to determine the rate x of individuals allocated to the GPU cores and the rate 1-x of those allocated to the CPU cores. The following equation shows a model for the runtime

$$t = gen(Nt_{master} + \max(\left\lceil \frac{xN}{P_{GPU}} \right\rceil t_{GPU}, \left\lceil \frac{(1-x)N}{P_{CPU}} \right\rceil t_{CPU})) \quad (7)$$

where *gen* is the number of generations, *N* is the number of individuals in the population, and P_{GPU} and P_{CPU} are the GPU and CPU cores in the platform, respectively. The *xN* individuals allocated to the GPU cores are evenly distributed among the GPU cores, and the *(1-x)N* individuals are equally distributed among the CPU cores. The parameter t_{master} corresponds to the time required by one of the cores to process the master task *T0* for a given iteration, while parameters t_{GPU} and t_{CPU} are, respectively, the time required by the GPU cores and the CPU cores to evaluate one individual. The parameters t_{master}, t_{GPU} and t_{CPU} can also be expressed as a function of the corresponding workload and the frequency of the corresponding processor as $t_{master}=W_{master}/F_{CPU}$, $t_{CPU}=W_{CPU}/F_{CPU}$, and $t_{GPU}=W_{GPU}/F_{GPU}$, where F_{CPU} and F_{GPU} are the frequencies of the CPU and GPU cores, respectively, and W_{master}, W_{CPU}, and W_{GPU} are, respectively, estimations of the cycles of the workloads of *T0*, the evaluation of an individual in the CPU, and the evaluation of an individual in the GPU. In this way, the execution time can be modeled in the following way:

$$t = gen(N \frac{W_{master}}{F_{CPU}} + \max(\left\lceil \frac{xN}{P_{GPU}} \right\rceil \frac{W_{GPU}}{F_{GPU}}, \left\lceil \frac{(1-x)N}{P_{CPU}} \right\rceil \frac{W_{CPU}}{F_{CPU}})) \quad (8)$$

Various considerations should be made in terms of the model in Equation 10. We assume that the times t_{GPU} and t_{CPU} required by the GPU cores and CPU cores, respectively, to evaluate one individual are the same for all the individuals in the GPU cores (t_{GPU}) or in the CPU cores (t_{CPU}). This situation can also be considered very unusual although it is possible to find useful applications that behave in the same way. The electroencephalogram (EEG) feature selection for BCI considered here can be suitably modelled accordingly as each individual has been assessed over a fixed number of iterations of the k-means algorithm to qualify the usefulness of the set of selected features that each individual in the population codifies. The data parallelism provided by the GPU therefore provides similar acceleration to the k-means algorithm for all the individuals in the population as demonstrated in some of our previous papers [6].

It is possible to fit two linear regressions by considering the values of the load distribution *x* that verifies

$$\left\lceil \frac{xN}{P_{GPU}} \right\rceil \frac{W_{GPU}}{F_{GPU}} < \left\lceil \frac{(1-x)N}{P_{CPU}} \right\rceil \frac{W_{CPU}}{F_{CPU}} \text{ and } \left\lceil \frac{xN}{P_{GPU}} \right\rceil \frac{W_{GPU}}{F_{GPU}} > \left\lceil \frac{(1-x)N}{P_{CPU}} \right\rceil \frac{W_{CPU}}{F_{CPU}}$$

We shall now describe an approximate energy consumption model. First, given a GPU including P_{GPU} cores running at frequency F_{GPU}, the energy consumed by the evaluation of *xN* individuals distributed among the P_{GPU} cores can be expressed as the product of the instantaneous power and the runtimes of the cores and their energy consumption while they are idle. In this way, the energy consumed by the GPU in each generation (*gen*) can be given as:

$$\begin{aligned} E_{GPU} &= Pow_{GPU} \left\lfloor \frac{xN}{P_{GPU}} \right\rfloor t_{GPU} + \frac{Pow_{GPU}}{P_{GPU}} \left(xN - \left\lfloor \frac{xN}{P_{GPU}} \right\rfloor P_{GPU} \right) t_{GPU} + E_{idleGPU} \\ &= Pow_{GPU} \frac{xN}{P_{GPU}} t_{GPU} + E_{idleGPU} \end{aligned}$$

where *Pow_{GPU}*

where Pow_{GPU} is the instantaneous power consumed when all the P_{GPU} processors of the GPU are evaluating individuals and $E_{idleGPU}$ is the energy consumed by the idle cores. Similarly, for each generation, the energy consumed by the P_{CPU} cores of the CPU at frequency F_{GPU} can be modeled as:

$$E_{CPU} = Pow_{CPU}\left\lfloor\frac{(1-x)N}{P_{CPU}}\right\rfloor t_{CPU} + \frac{Pow_{CPU}}{P_{CPU}}\left((1-x)N - \left\lfloor\frac{(1-x)N}{P_{CPU}}\right\rfloor P_{CPU}\right) t_{CPU} + E_{idleCPU}$$
$$= Pow_{CPU}\frac{(1-x)N}{P_{CPU}} t_{CPU} + E_{idleCPU}$$

where Pow_{CPU} is the instantaneous power consumed when all the P_{CPU} CPU cores are evaluating individuals, and $E_{idleCPU}$ is the energy consumed by the idle cores. Taking into account the previous expressions, the energy consumed across the generations executed by the algorithm can be given as

$$E = gen\left(Pow_{GPU}\frac{xN}{P_{GPU}}t_{GPU} + Pow_{CPU}\frac{(1-x)N}{P_{CPU}}t_{CPU} + E_{idleGPU} + E_{idleGPU} + \epsilon_{energy}\right)$$
$$= gen\left(Pow_{GPU}\frac{xN}{P_{GPU}}\left(\frac{W_{GPU}}{F_{GPU}}\right) + Pow_{CPU}\frac{(1-x)N}{P_{CPU}}\left(\frac{W_{CPU}}{F_{CPU}}\right) + E_{idleGPU} + \epsilon_{energy}\right) \quad (9)$$

The first and second terms in (11) correspond to the energy consumed by the GPU cores and the CPU cores, respectively, when they execute their respective load rates *x* and *1-x*, and the third and fourth terms to the energy consumed by the GPU and CPU idle cores. The fifth term in (11), ϵ_{energy}, corresponds to the energy consumed by task *T0* and the other platform elements (memory, buses, I/O, *uncore* elements in the microprocessors, etc.). Models for $E_{idleGPU}$ and $E_{idleCPU}$ can also be obtained in terms of platform parameters and workload distribution as follows and the energy consumed by the idle GPU cores is given by

$$E_{idleGPU} = Pow_{idleGPU}\left(1 - \frac{xN}{P_{GPU}} + \left\lfloor\frac{xN}{P_{GPU}}\right\rfloor\right)\left(\frac{W_{GPU}}{F_{GPU}}\right) \quad (10)$$

and the energy consumed by the idle CPU cores by

$$E_{idleCPU} = Pow_{idleCPU}\left(\left(1 - \frac{(1-x)N}{P_{CPU}} + \left\lfloor\frac{(1-x)N}{P_{CPU}}\right\rfloor\right)\left(\frac{W_{CPU}}{F_{CPU}}\right)\right) \quad (11)$$

Model parameters for time and energy consumption in (8) to (11) could be obtained from regressions on the experiments performed with different distribution rates *x* and *1-x*, and number of individuals *N*, and generations *gen*, given the characteristics of the GPU-CPU platform in terms of the number of processors, P_{CPU} and P_{GPU}, and their operating frequencies F_{GPU} and F_{CPU}. By fitting (8) with the experimental time measurements, it would be possible to determine the parameters W_{master}, W_{GPU}, and W_{CPU}. Once these values have been substituted in (9)-(11), the experimental energy consumption values can be used to determine Pow_{GPU}, Pow_{CPU}, $Pow_{idleGPU}$ and $Pow_{idleCPU}$, and the shape of ϵ_{energy} after fitting the model for energy consumption (11). It is then possible to build the cost function for population conditions, generations, frequencies, etc. that have not been previously executed by determining the relative deviations given in Equation 1, and to estimate the best value for *x* for a given population (number *N* of individuals) and operating frequencies to execute the application on a given platform (number of GPU and CPU cores and the remaining parameters). This approach for static workload scheduling is experimentally analyzed in Section 3.3.

More specifically, we consider a linear regression model that according to Equations 9-11 presents the following terms:

$$E = A_0 + A_1 \times \left(\frac{xN}{P_{GPU}}\right) + A_2 \times \left\lfloor\frac{xN}{P_{GPU}}\right\rfloor + A_3 \times \left\lfloor\frac{(1-x)N}{P_{CPU}}\right\rfloor + \epsilon_{energy} \qquad (12)$$

where the coefficients A0 – A3 can be related with the parameters of the model in (9)-(11) as:

$$A_0 = gen \times \left(Pow_{idleGPU} \times \frac{W_{GPU}}{F_{GPU}} + Pow_{idleCPU} \times \frac{W_{CPU}}{F_{CPU}} \times \left(1 - \frac{N}{P_{CPU}}\right) + Pow_{CPU} \times \frac{N}{P_{CPU}} \times \frac{W_{CPU}}{F_{CPU}}\right)$$

$$A_1 = gen \times \left(\left((Pow_{GPU} - Pow_{idleGPU}) \times \frac{W_{GPU}}{F_{GPU}}\right) - \left((Pow_{CPU} - Pow_{idleCPU}) \times \frac{W_{CPU}}{F_{CPU}} \times \frac{P_{GPU}}{P_{CPU}}\right)\right)$$

$$A_2 = gen \times Pow_{idleGPU} \times \frac{W_{GPU}}{F_{GPU}}$$

$$A_3 = gen \times Pow_{idleCPU} \times \frac{W_{CPU}}{F_{CPU}}$$

3.2. Energy measures

Instantaneous power and energy consumption can be evaluated by the performance monitoring counters provided by the corresponding processor. For example, [7] describes the extension of the Performance API (PAPI) library to measure power and energy based on these counters. In this paper, however, the node's instantaneous power and energy consumption have been measured with a wattmeter that we have developed based on an *Arduino Mega* card and this is shown in Figure 1. It provides four real-time measurements per second for each of the four nodes of our platform corresponding to the instantaneous power (in Watts) and the cumulated consumed energy (in Wxh) of the entire node. The measurements are obtained thanks to sensors that provide the amount of electric current in the wire connecting the node to the electricity grid. By measuring energy consumption at these points, not only is it possible to obtain the energy consumption of the active components but also the losses arising from power supply conversions. In this way, the conclusions attained from these consumption measurements for the entire node are relevant as they clearly demonstrate whether a proposed strategy devised to improve energy efficiency is effective and useful.

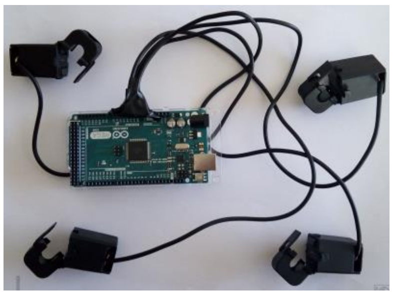
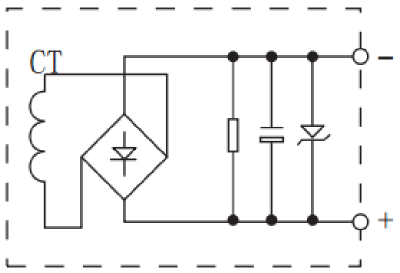

Figure 1. Arduino-based wattmeter used to measure power and energy (left) and circuit scheme of the current sensor (right)

The current sensor is the YHDC-SCTD010T-5A and its circuit scheme is also shown in Figure 1. It can provide up to 5A, with a proportional output of between 0 and 5V and an accuracy of $\underline{+}2\%$. The *Arduino* board includes a 10-bit A/D converter and uses its internal voltage reference of 2.56V (only available in *Arduino Mega* boards) in order to take full advantage of its dynamic range. Although this approach limits the maximum value of the input to 2.56V, it is possible to measure up to 588W in the experiments. As the nodes consume less power, this bound does not represent a real limitation. The *Arduino Mega* board is connected by USB to one of the cluster nodes in order to transmit the data to the system through the corresponding serial port and obtain power. In the case of a Linux computer, the interface /dev/ttyACM0 is created to send and receive data. In our first version of the wattmeter, the *Arduino* board takes four measurements per second per sensor and transmits the data for instantaneous power (in W) and consumed energy (in Wxh) for each sensor through the port. The energy consumption values can be set to zero at any time so that energy consumption for a given period of time can be easily estimated. The program executed by the *Arduino Mega* board is written in *Python*.

In terms of the operating mode control, the standard Advanced Configuration and Power Interface (ACPI) [8] includes mechanisms to manage and save energy, adequately controls BIOS operation, and provides information about the configuration and control of the processor states in terms of energy consumption (C0, C1, C2, C3,.., Cn) and performance (P0, P1,…, Pn). In the same way, the Linux kernel implements the infrastructure *cpufreq* [9] that allows the operating system (either automatically through the events generated by the ACPI or through user program calls) to change the operating frequency of the processor for energy saving. The so-called *governors* [10] are included in the *cpufreq* to implement specific policies to control the processor clock. The interface to use these services at the user level can be found in cpufreq.h [11].

3.3. Results analysis

We have implemented an OpenCL (version 1.2) code (compiled with GCC 4.8.5) for the target multi-objective feature selection problem corresponding to a BCI task [4] applied to a dataset containing 178 patterns extracted from the data recorded in the BCI Laboratory of the University of Essex. Each pattern is an EEG described by 3600 features corresponding to 12 features for each of the 20 temporal segments and 15 electrodes [4].

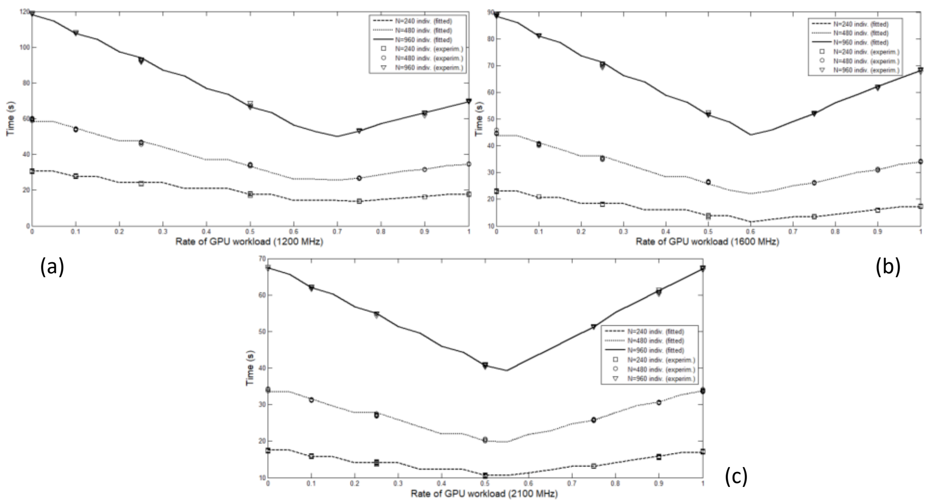

Figure 2. Curves fitted with model (8) for experimental running times with N=240, 480, 960 and FCPU=1200 (a), 1600 (b), and 2100 MHz (c)

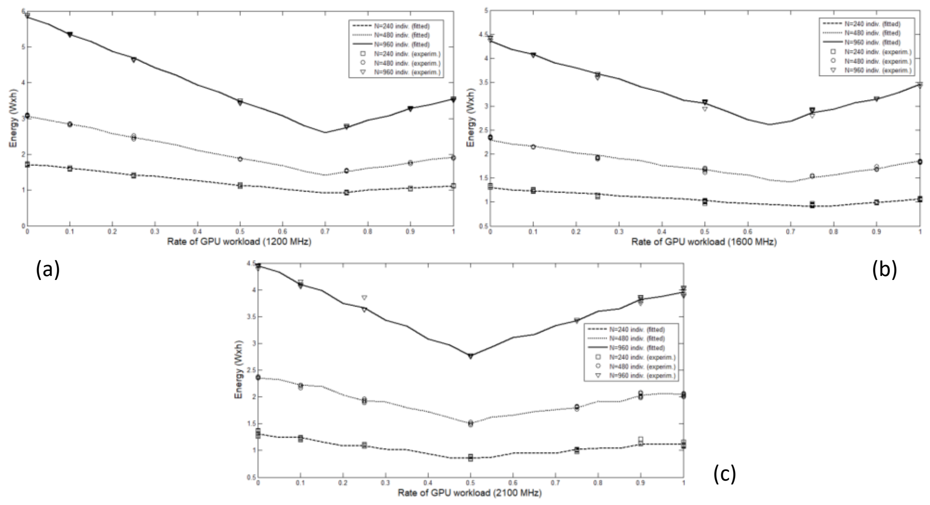

Figure 3. Curves fitted with model (9) for experimental consumed energies with N=240, 480, 960 and FCPU=1200 (a), 1600 (b) and 2100 MHz (c)

In our experiments, the CPU cores have executed the threads allocated to them at 1200 MHz, 1600 MHz and 2100 MHz. According to the experimental results observed, the ϵ_{energy} term in Equation (9) has been modelled as

$$\epsilon_{energy} = \omega(x - x_c)\left\lfloor \frac{x}{x_c} \right\rfloor$$

where ω is a proportionality constant, and x_c can be obtained from the crossing point of two linear regressions: the first uses experimental results of *x* values close to 0, corresponding to much higher workloads in the CPU cores than in the GPU ones, whereas the second linear regression is applied to *x* values close to 1, and also values where the GPU is much more loaded than the CPU. The points on the graphs in Figures 2 and 3 provide the time and energy measurements and the curves have been fitted using our models and linear regression (Equations 7 to 12). In every case, the regression is statistically significant and the R^2-statistics is higher than 0.945. As Figures 2 and 3 show, the accuracy of the fitted curves is acceptable. In particular, the minima of the curves correspond to those experimentally observed.

The parameters of our models described in Equations 8 and 9 can also be obtained from the experimental results obtained in the case of *N*=240, *gen*=50, and the three considered values of F_{CPU}. These values allow our models to be specified for our platform and application by fitting the curves of the models for the corresponding N values (in our experiments 480 and 960 individuals). Table 1 provides the mean relative errors for the time and error predictions made using the parameters of Equations 8 and 9 obtained from N=240.

	Energy prediction mean relative error	*Time prediction mean relative error*
	240 indiv. 1.2 GHz	
480 indiv. 1.2 GHz	0.166±0.035	0.031±0.026
960 indiv. 1.2 GHz	0.223±0.042	0.015±0.011
	240 indiv. 1.6 GHz	
480 indiv. 1.6 GHz	0.171±0.035	0.023±0.019
960 indiv. 1.6 GHz	0.246±0.049	0.014±0.11
	240 indiv. 2.1 GHz	
960 indiv. 1.6 GHz	0.108±0.012	0.020±0.015
960 indiv. 2.1 GHz	0.179±0.013	0.016±0.013

Table1. Mean relative error of energy and time prediction from the parameters obtained with the models for N=240 individuals

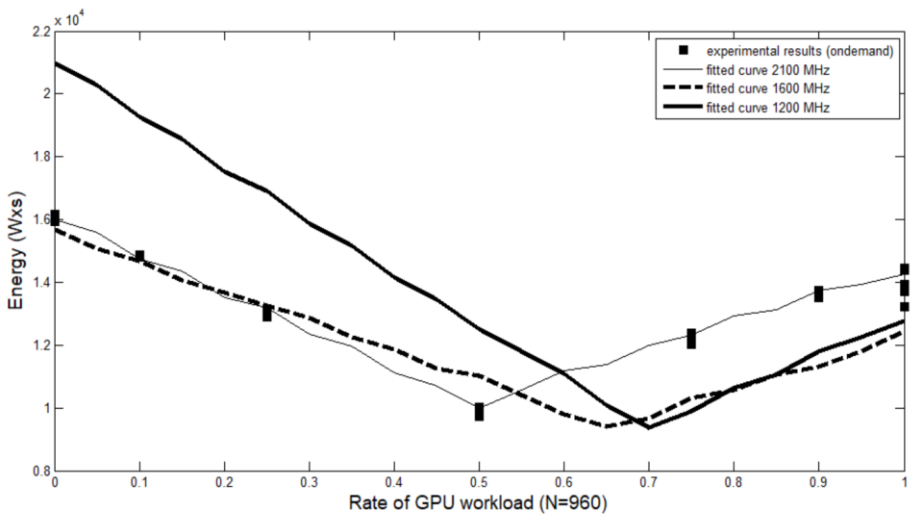

Figure 4. Curves fitted for experimental consumed energies with Model 9 and N=960. The black squares represent the experimental results

The experimental results obtained for the *ondemand* option of CPUfreq *governors* [10] are quite similar to those obtained when F_{CPU}=2100 MHz in both runtime and energy consumed. Figure 4 shows the corresponding fitted curves of energy consumed and the experimental results obtained with the *ondemand* option for N=960. As can be seen, the values obtained by using *ondemand* are best when $x \leq 0.5$, corresponding to situations where the CPU workload is larger than the GPU workload. This is correct as the CPUfreq *governors* only control the states of the CPU cores.

Figure 5 shows some curves corresponding to the temporal evolution of the instantaneous power. The curves in Figure 5a illustrate the evolution for different operation frequencies in the CPU cores with equal distribution of individuals among GPU and CPU cores (x=0.5) while Figure 5b gives the different distribution curves of individuals at the same operating frequency in the CPU cores (F_{CPU}=2100 MHz). Figure 5a clearly shows that the instantaneous power values are higher for the larger operating frequencies. Instantaneous power values for 1200 and 1600 MHz are in fact closer to each other than they are for 1600 and 2100 MHz. Figure 5b shows that the instantaneous power values also change with the rate of individuals allocated to the CPU cores. The curve with the lowest instantaneous power values corresponds to the situation where all the individuals are allocated to the GPU. Regarding the other two curves, the one with larger power values corresponds to *x=0.5*. Although only half of the population is allocated to the CPU cores, this behavior could be explained by the power consumed by the elements of the node required to communicate the CPU core where the master thread is running and the GPU.

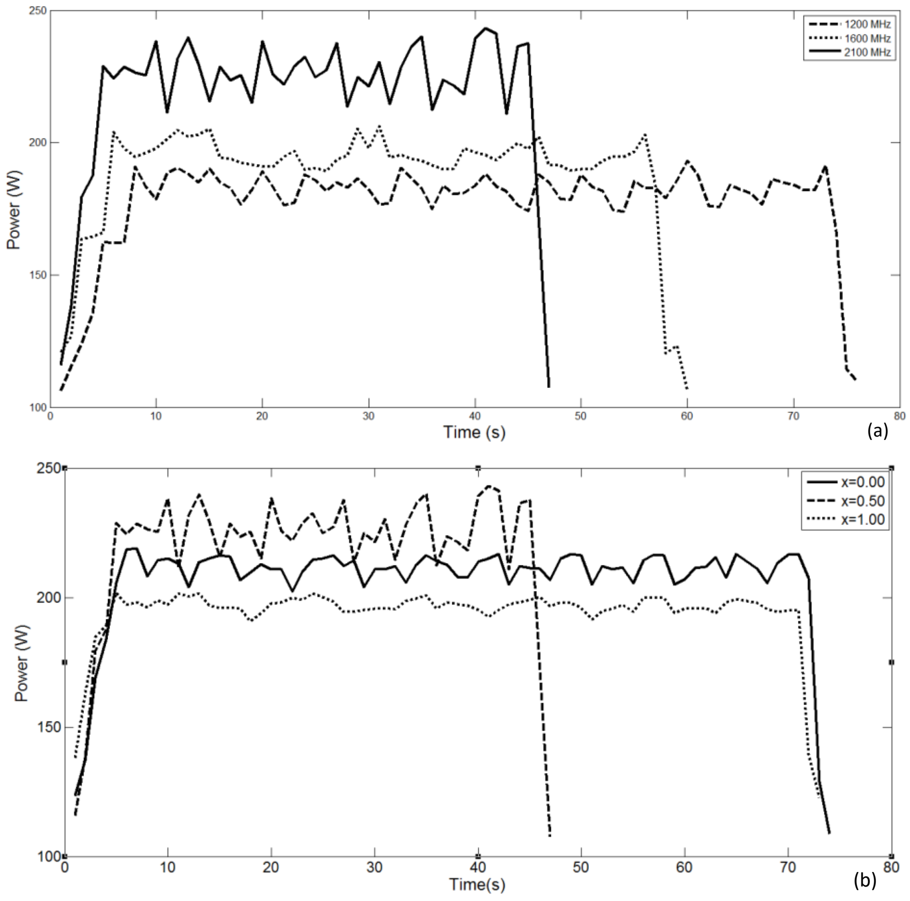

Figure 5. Temporal evolution of the instantaneous power: (a) for *x*=0.5 and different values of frequency in the CPU cores; and (b) for a frequency of 2100 MHz in the CPU cores and different values of *x* (rate of workload allocated to the GPU) and N=960

Figure 6 shows the shape of the cost function *Δ=aΔt+bΔE* for N=240 individuals and 1600 MHz, and different values for parameters *a* and *b*. Depending on these values, the minimum of the cost function corresponds to a minimum in the energy consumption (x=0.75), in the running time (x=0.60), or represents a trade-off between time and energy.

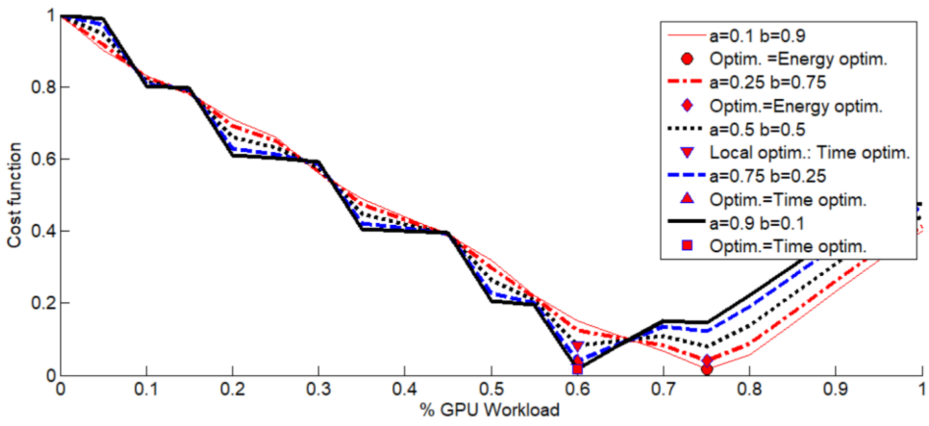

Figure 6. Cost function for different values of parameters *a* and *b*, for *N*=240 and F_{CPU}=1600 MHz.

4. Related work

A large number of energy-aware scheduling procedures have already been proposed. Although the majority of these require programmer-exposed DVFS strategies for runtime power management, it is not possible to take into account energy consumption and time optimization principles in platforms that do not allow the user to access and control the DVFS alternatives online (or this alternative is so costly and should be avoided). A black-box scheduling approach is therefore proposed in [12] based on an offline power model and an online workload modeling. Other approaches build power and energy consumption models either by running micro-benchmarks [13] or by evaluating the energy consumption of the platform components [14]. In their paper [15], the authors define energy-aware strategies in codes for sparse linear systems after analyzing and modelling the different power-saving modes of CPU cores. Along the same lines, [16] proposes energy consumption characterization by applying multiple linear regression models.

As energy consumption and runtime are competing objectives, a multi-objective (more specifically a bi-objective) approach is required to tackle the development of an energy-aware scheduling problem. By way of future work, the paper [17] proposes the use of multi-objective evolutionary algorithms to learn about the trade-offs evaluated by the two-level schedulers described in the paper. Since a scheduling algorithm built on a Pareto-based multi-objective evolutionary algorithm would require a long computing time together with a strategy to select from the different alternatives in the Pareto front, a better alternative is to use a cost function that weights the energy and time objectives as proposed in [18] using a weighted energy-delay product corresponding to the desired tradeoff among those defining the alternatives represented by the Pareto front. In this paper, we also follow this proposal and we propose a cost function that comprises the two goals of energy consumption and runtime although not through an energy-delay product.

The energy efficiency of GPU has attracted interest in recent years and has been previously analyzed in various papers [19-21]. In terms of the energy consumption efficiency of hybrid CPU-GPU platforms, some relevant results can be found in various publications [22, 23]. For example, [22] provides analytical models to provide insight into performance gains and energy consumption in different CPU-GPU platforms and concludes that greater parallelism allows opportunities for energy saving and encourages the development of energy saving parallel applications. The proposal in this paper is a workload balancing procedure based on the multi-objective cost function and built by regression, following the approach described in [16].

5. Conclusions

The problem of scheduling on heterogeneous architectures to minimize both runtime and energy consumption has recently attracted interest as energy consumption has become one of the major concerns in high-performance computing and data-center facility management. In addition to the availability of heterogeneous architectures with different energy consumption characteristics, techniques such as dynamic voltage and frequency scaling (DVFS) make it possible to devise task scheduling procedures which are concerned with minimizing both runtime and energy consumption. Nevertheless, DVFS control is not available at the user level and it may even be hard to find energy and power prediction models that are accurate enough to be used by an efficient scheduling procedure. One alternative approach is to fit black-box models to the experimental results obtained by the target application in certain conditions and to use these to predict application behavior in other experimental conditions.

This paper analyzes energy consumption and runtime behaviors in GPU-CPU platforms of parallel master-worker evolutionary algorithms applied to a feature selection problem for EEG classification in BCI tasks. We have defined models for runtime and energy consumption that have been fitted to the experimental results by multiple linear regression with values of the R^2-statistics which are greater than 0.945, and statistical significance in every case. These models provide prediction errors which are lower than 24.6% for energy consumption and 3.1% for runtime in the experimental alternatives we have considered. Since many bioinformatics and data mining applications involve classification, clustering, feature selection, and optimization that due to their complexity require metaheuristics such as evolutionary algorithms, the conclusions of the approach described in this paper may be relevant to many different situations. Much work remains to be done, however, in terms of improving energy consumption models for other platform applications and elements other than the GPU and CPU cores.

Acknowledgments

This work has been funded by Project TIN2015-67020-P (Spanish “Ministerio de Economía y Competitividad” and ERDF funds).

References

[1] Mittal, S.; Vetter, J.S.:”A survey of CPU-GPU heterogeneous computing techniques”. ACM Comput. Surv. 47, 4, Article 69, 35 pages. July, 2015. DOI: http://dx.doi.org/10.1145/2788396.

[2] O’Brien, K.; Pietri, I.; Reddy, R; Lastovetsky, A.; Sakellariou, R.:”A survey of power and energy models in HPC systems and applications”. ACM Comput. Surv. 50, 3, Article 37, 38 pages. July, 2017. DOI: http://dx.doi.org/10.1145/3078811.

[3] Lee, Y.C.; Zomaya, A.Y.:”Energy conious scheduling for distributed computing systems under different operationg conditions”. IEEE Trans. On Parallel and Distributed Systems, Vol.22, No.8, pp.1374-1381. August, 2011.

[4] Ortega, J.; Asensio-Cubero, J.; Gan, J. Q.; Ortiz, A.: “Classification of motor imagery tasks for BCI with multiresolution analysis and multiobjective feature selection”. BioMedical Engineering OnLine, 2016.

[5] GNU gprof manual: http://sourceware.org/binutils/docs/gprof/index.html

[6] Escobar, J.J.; Ortega, J.; González, J.; Damas, M.; Díaz, A.F.: “Parallel high-dimensional multi-objective feature selection for EEG classification with dynamic workload balancing on CPU-GPU”. Cluster Computing. 2017;20(3):1881–1897.

[7] Weaver, V.N.; Johnson, M.; Kasichayanula, K.; Ralph, J.; Luszczek, P.; Terpstra, D.; Moore, S.:”Measuring energy and power with PAPI”. 41st Intl. Conference on Parallel Processing Workshops (ICPPW), pp. 262-268, 2012

[8] Advanced configuration and power interface specification (ACPI): http://www.acpi.info/

[9] CPU frequency scaling: https://wiki.archlinux.org/index.php/CPU_frequency_scaling

[10] CPUFreq Governors: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[11] `cpufreq.h`: https://code.woboq.org/linux/linux/include/linux/cpufreq.h.html

[12] Barik, R..; Farooqui, N.; Lewis, B.T.; Hu, C.; Shpeisman T.: “A black-box approach to energy-aware scheduling on integrated CPU-GPU systems”. In: CGO’2016:70–81ACM; 2016; Barcelona, Spain.

[13] Hong, S.; Kim, H.:"An Integrated GPU Power and Performance Model". SIGARCH Computer Architecture News. 2010;38(3):280–289.

[14] Ge, R.; Feng, X.; Burtscher, M.; Zong, Z.: "PEACH: A Model for Performance and Energy Aware Cooperative Hybrid Computing". In: CF'2014:24:1– 24:2ACM; 2014; Cagliari, Italy.

[15] Aliaga, J.I.; Barreda, M.; Dolz, M.F.; Martín, A.F.; Mayo, R.; Quintana-Ortí, E.S.:"Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems". Cluster Computing, 17, pp. 1335-1348, 2014.

[16] De Sensi, D.:"Predicting performance and power consumption of parallel applications". In 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2016. DOI: 10.1109/PDP.2016.41.

[17] Dorronsoro, B.; Nesmachnow, S.; Taheri, J.; Zomaya, A.Y.; Talbi, E-G; Bouvry, P.:"A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems". Sustainable Computing: Informatics and Systems, 4, pp.252-261, 2014.

[18] Ge, R.; Feng, X.; Cameron, K.W.:"Improvement of Power-Performance Efficiency for High-End Computing". In: IPDPS'2005:233–240IEEE Computer Society; 2005; Denver, Colorado, USA.

[19] Wang, Y.; Ranganathan, N.:"An instruction-level energy estimation and optimization methodology for GPU". 2011 11th Intl. Conf. on Computer and Information Technology, pp.621-628, 2011.

[20] Cebrián, J.M.; Guerrero, G.D.; García, J.M.:"Energy efficiency analysis of GPUs". 2012 IEEE 26th Intl. Parallel and Distributed Processing Symp. Workshops & PhD Forum, pp. 1014-1022, 2012.

[21] Mittal, S.; Vetter, J.S.:"A survey of methods for analyzing and improving GPU energy efficiency". ACM Comput. Surv. 47, 2, Article 19, 23 pages. July, 2014. DOI: http://dx.doi.org/10.1145/2636342.

[22] Marowka, A.. "Energy Consumption Modeling for Hybrid Computing". In: Euro-Par'2012:54–64Springer; 2012; Rhodes Island, Greece.

[23] Allen, T.; Ge, R..: "Characterizing Power and Performance of GPU Memory Access". In: E2SC'2016:46–53IEEE Press; 2016; Salt Lake City, Utah, USA.