The Journal of Supercomputing (2019) 75:1610–1624
https://doi.org/10.1007/s11227-018-2718-6

A heuristic technique to improve energy efficiency with dynamic load balancing

Alberto Cabrera[1] · Alejandro Acosta[1] · Francisco Almeida[1] · Vicente Blanco[1]

Published online: 19 December 2018

Abstract

Heterogeneous computers require a well-distributed workload to operate efficiently. When possible, this load balancing procedure should redistribute the workload with minimal knowledge of the system architecture, to reduce overhead. We propose a generic dynamic load balancing technique for iterative problems, independent from the resource to optimize. Proof of this generalization is given through formalization of the designed technique. A heuristic algorithm is defined based upon this formalization, with a structure that facilitates different objective functions. As a result, swapping the objective function can be done with relatively low effort. This heuristic is implemented to minimize energy consumption in an application. We use this application to solve three different dynamic programming problems with multiple GPUs. The implementation is described and then compared against two different workloads, the homogeneous distribution and another dynamic load balancing technique. Our experimentation shows good results in minimizing the overall energy consumption with low overhead.

Keywords Dynamic load balancing · Iterative algorithms · Parallel computing · Energy efficiency

✉ Alberto Cabrera
Alberto.Cabrera@ull.edu.es

Alejandro Acosta
aacostad@ull.edu.es

Francisco Almeida
falmeida@ull.es

Vicente Blanco
Vicente.Blanco@ull.edu.es

[1] HPC Group, Escuela Superior de Ingeniería y Tecnología, Universidad de La Laguna, San Cristóbal de La Laguna, Spain

1 Introduction

The path toward exascale has brought a substantial attention to energy consumption and efficiency. The tendency to build environments only with traditional multicore systems has been reduced drastically in favor of heterogeneous platforms that contain coprocessors such as multiGPU systems and ARM architectures. The Top500 [18] list reflects this trend in the top computers, with the appearing of the mentioned technologies in the most powerful machines, but also raises awareness with the high power drawn for these powerful computers. The new node architectures introduce load balancing issues inside the computational nodes themselves, increasing the difficulty to achieve the optimal use of the computational resources. Homogeneous load distributions are no longer desirable, and applications need to be tuned to reach peak performance when the target parallel computer changes [8]. Now, not only time is of interest but energy has to be considered as well to minimize the costs of sustained computing, and while the same optimization principles can be applied to energy, they have to be adapted accordingly to different constraints. Additionally, the numerous measurement devices and software available [3,5,10] increase the heterogeneity of the scientific community applications.

We propose a generic heuristic algorithm to balance the workload between different parallel processes dynamically in iterative algorithms. This generalization is based on an arbitrary objective function that can be changed accordingly to the needs. Some examples of the possibilities for objective functions are performance, energy efficiency, communication time or memory accesses. We have formalized our algorithm to support our claims and understand the requirements for our objective functions to work. Then, we present a pseudo-code of this formalization using principles from skeleton programming to facilitate the different objective functions.

To back our proposal, we also present an specific implementation of the algorithm to minimize energy consumption in a heterogeneous multiGPU environment. This implementation was used to solve iterative problems that utilize dynamic programming techniques. Starting from an even distribution, we apply the load balance heuristic to minimize the total energy consumption of the execution. An analysis of the results obtained allows to illustrate the advantages and the drawbacks of our proposal.

Our computational experimentation reveals that our technique achieves less energy consumption than the homogeneous workload balance. Moreover, compared to a load balancing technique that has been optimized for performance, we obtain less energy consumption for most of the presented cases. Since the technique has been developed independently from the objective resource, it is possible to apply the same approach to different resources by modifying the objective function.

This work is structured as follows: related work in the field of energy efficiency and load balance is described in Sect. 2. Section 3 introduces the formalization of the problem for iterative algorithms. Section 4 describes the heuristic algorithm derived from the presented concepts in the previous one. Section 5 presents our computational experience for an implemented application and finally, Sect. 6 presents our conclusions.

2 Related work

In order to improve time performance in heterogeneous systems, multiple techniques can be found in the literature over the last decades. Load balancing techniques have been discussed for improving performance for complete parallel systems, for high throughput networks and parallel storage systems. General models for dynamic load balancing are presented and compared for multiple schemes such as spectral graph partitioning, the generalized dimension exchange or the diffusion method, illustrating their main issues, as well as scalability studies to provide insight into load balancing in different architectures [12,15,26,27]. Multiple libraries that implement dense linear algebra packages make use of well-formulated computational models based on Directed Acyclic Graphs in conjunction with dynamic task scheduling. This is the case for Plasma [13] and FLAME [25] for shared memory codes. Moreover, an effort to port this knowledge to hybrid architectures composed by multicores and GPUs is present in libraries such as MAGMA [2] and FlameGPU [22]. Another approach is the development of applications using highly tuned programming skeletons, which covers the general aspects of a given implementation, to then develop the specifics of the algorithm. This approach, made to obtain maximum time efficiency with an optimal load balance, could be adapted to obtain energy-efficient implementations of different algorithms. Examples of this concept are SkelCL [23], Marrow [16] and DPSKEL [1,20].

Moreover, with the actual trend of high-performance computing, extensive effort is being made to obtain energy-aware techniques for existing algorithms in modern architectures, including the application of scheduling algorithms and heuristic methods focused on improving the efficiency of these high-power systems. HEROS [11] introduces a load balancing algorithm for energy-efficient resource allocation in heterogeneous systems, PVA [24], peer VMs aggregation, proposes to enable dynamic discovery of communication patterns and reschedule VMs based on the acquired knowledge with virtual machine migrations. Energy-aware allocation heuristics have also been proposed for a complete data center [4]. ALEPH [21] addresses the bi-objective optimization problem for performance and energy (BOPPE) for many core systems, by modeling the objective system. Our proposal starts from a homogeneous workload distribution and proceeds to redistribute after evaluating the performance of each process in the parallel application. This approach shares some characteristics with the works presented in *ULL_Calibrate_lib* [1], ADITHE [17] and E-ADITHE [9]. In the first iterations of the algorithm, these techniques redistribute a homogeneous load distribution according to the speed of the processes. We apply this principle in our proposal with a generic approach for resource usage. We then dynamically reallocate the work in order to minimize resource usage without a training phase or a known model of the application. While multiple models and generalizations have been made over the last decades, our generalization provides a formalization for iterative problems with unknown architecture, network and, principally, resource to optimize. While previous contributions, including our work to improve energy efficiency [6], suppose a great effort to increase the performance of applications or to reduce energy consumption, this new approach isolates the objective resource to optimize from the load balancing technique. We also provide a new algorithmic structure that can be filled with different objective functions. In this work, we have focused our efforts in applying this formalization to minimize energy consumption. However our approximation can be replicated for different objectives, such as maximizing performance or minimizing communications.

3 Dynamic load balancing for energy efficiency

In parallel computing, load balancing is a technique that reallocates workload between processes to minimize inefficiencies caused by the different computational capabilities of the underlying architecture or by the irregularities of the algorithm used to solve a problem. Architectures can be heterogeneous because all the computing elements are not uniform and the algorithm can have different number of operations to solve per iteration. These inefficiencies should be addressed to optimize the resources available on a given parallel system.

3.1 Load balancing problem formalization

The problem we are facing can be considered as a generic tuning problem in a parallel system, that could be defined as the minimization of a function ϱ that has the following parameters:

- T is the tuning problem space of the function ϱ.
- P is the problem to solve.
- A is the algorithm to use.
- N is the size of the problem.
- p is the number of parallel processes.
- $\mathbf{w}$ is the workload distribution of the parallel processes.
- x is the resource we are tuning.

Using this notation, the problem can be described as:

$$\begin{aligned} &\min \varrho(P, A, N, p, \mathbf{w}) \\ \varrho : &\, T \longrightarrow \mathbb{R} \\ &P, A, N, p, \mathbf{w} \longmapsto x \end{aligned} \tag{1}$$

As we address the problem of load balancing in iterative problems, we will address only a subset of the space T. Once defined P, A and our number of parallel processes p, we can simplify the problem into a function to obtain an optimal workload distribution for the selected parameters, such as:

$$\begin{aligned} \exists F^{\varrho} : \mathbb{R}^{p+1} &\longrightarrow \mathbb{R} \\ N, \mathbf{w} &\longmapsto x \end{aligned} \tag{2}$$

where the function F^{ϱ}, given a workload w and a problem size N for p processes can return a value x of the resource we want to optimize, execution time or energy efficiency.

In the case of iterative problems, we consider a problem to be regular when the iteration j and the iteration $j+1$ spend the same amount of resources x with constant workload w. Assuming that an iterative problem has N iterations, we introduce the concept of $f_j^{\varrho}(\mathbf{w})$, as the N functions that define the resource consumption for each iteration j,

$$\begin{aligned} \exists f_j^{\varrho} : \mathbb{R}^p &\longrightarrow \mathbb{R} \\ \mathbf{w} &\longmapsto x_j \end{aligned} \qquad x = \sum_{j=1}^{N} x_j \qquad 1 \leq j \leq N \tag{3}$$

the total consumption of the resource x is defined as the sum of each x_j. Equation 3 can be used as basis for the definition of a generic iterative regular problem, a function $F^{\varrho}(N, \mathbf{w})$ defined as:

$$\begin{aligned} F^{\varrho}(N, \mathbf{w}) &= f_1^{\varrho}(\mathbf{w}) + f_2^{\varrho}(\mathbf{w}) + \cdots + f_N^{\varrho}(\mathbf{w}) \\ &\text{where} \\ f_j^{\varrho}(\mathbf{w}) &= f_k^{\varrho}(\mathbf{w}) \qquad \forall\, j,\ k \\ \text{thus, } \min f_j^{\varrho}(\mathbf{w}) &\implies \min F^{\varrho}(N, \mathbf{w}) \end{aligned} \tag{4}$$

However, when we face irregular problems, the cost of a given iteration is not the same as the previous nor the following one, and while the definition of f_j^{ϱ} prevails, we have to consider a different scenario. The scenario for a dynamic load balancing problem needs to find a function for every iteration of the problem since

$$\begin{aligned} F^{\varrho}(N, \mathbf{w}) &= f_1^{\varrho}(\mathbf{w}) + f_2^{\varrho}(\mathbf{w}) + \cdots + f_N^{\varrho}(\mathbf{w}) \\ \text{and, } f_j^{\varrho}(\mathbf{w}) &\neq f_k^{\varrho}(\mathbf{w}) \qquad \text{for any } \ j,\ k \\ \text{thus, } \min f_j^{\varrho}(\mathbf{w}) &\nRightarrow \min f_k^{\varrho}(\mathbf{w}) \\ \min f_j^{\varrho}(\mathbf{w}) &\nRightarrow \min F^{\varrho}(\mathbf{w}) \end{aligned} \tag{5}$$

so in order to minimize our objective function F, we need minimize every function f_j by finding its optimal $\mathbf{w}$. This allows us to formalize our dynamic load balancing problem for p parallel processes as finding

$$\begin{aligned} \mathbf{w}_j &= \left[\, w_{1,j},\ w_{2,j},\ \cdots,\ w_{p,j} \,\right] \qquad 1 \leq j \leq N \\ \text{such that } F^{\varrho}(N, \mathbf{w}) &= \sum_{j=1}^{N} f_j^{\varrho}(\mathbf{w}_j) \text{ be minimized} \end{aligned} \tag{6}$$

Finally, as Eq. 6 considers N different $\mathbf{w}$, we can consider the input of the function F^{ϱ} as a matrix W of dimensions p by N or number of processes by number of iterations, which results in this last expression

$$W = \begin{bmatrix} w_{1,1} & w_{2,1} & & \cdots & w_{p,1} \\ \vdots & \vdots & \ddots & & \vdots \\ w_{1,j} & w_{2,j} & & w_{i,j} & w_{p,j} \\ \vdots & \vdots & & \ddots & \vdots \\ w_{1,N} & w_{2,N} & & \cdots & w_{p,N} \end{bmatrix}$$

$$\mathbf{w}_j = \left[w_{1,j},\, w_{2,j},\, \cdots,\, w_{p,j} \right]$$

$$\text{such that } F^{\varrho}(N, W) = \sum_{j=1}^{n} f_j^{\varrho}(\mathbf{w}_j) \text{ be minimized} \tag{7}$$

As we only have knowledge of the values of the functions f_j^{ϱ} a posteriori, i.e., f_j^{ϱ} is known in iteration $j+1$ after $\mathbf{w}$ is assigned, the generic formulation gathers special importance for defining suitable estimations of the problem performance. This is particularly useful as our objective functions, F^{ϱ} and f_j^{ϱ}, are very difficult to obtain. Estimations of f_j^{ϱ} for different $\mathbf{w}$ are needed for finding the best $W_{p,n}$. Additionally, the generalization could be used as a tool to generate more complex optimization strategies by changing the objective resource at runtime, e.g., from energy to time, with relatively low effort.

4 Heuristic load balancing

4.1 Definition

As the function f_j^{ϱ} described in Sect. 3.1 is very costly to calculate, we have to obtain a function g that models f_j^{ϱ} for a workload $\mathbf{w}$, and use this new function, to predict the behavior of the function f_{j+1}^{ϱ}. g is a heuristic function that takes as input the current workload $\mathbf{w}_j$ and the value of resources consumed, $f_j^{\varrho}(\mathbf{w}_j)$, which were measured after the computation was performed and returns a new workload distribution $\mathbf{w}_{j+1}$. For regular problems, $\mathbf{w}_{j+1}$ will be a valid solution if the result $f_{j+1}^{\varrho}(\mathbf{w}_{j+1})$ for the next iteration j is similar or better than the result that would be obtained using the previous workload distribution $f_j^{\varrho}(\mathbf{w}_j)$. Equation 8 represents formally this conditions,

$$\begin{aligned} &\exists\, g : \mathbb{R}^{p+1} \longrightarrow \mathbb{R}^{p} \\ &\mathbf{w}_j,\, f_j^{\varrho}(\mathbf{w}_j) \mapsto \mathbf{w}_{j+1} \\ \text{such that,}\quad &f_j^{\varrho}(\mathbf{w}_j) \geq f_{j+1}^{\varrho}(\mathbf{w}_{j+1}) \end{aligned} \tag{8}$$

To accomplish the definition of Eq. 8, g requires to have a mechanism to estimate the values of $f_j^{\varrho}(\mathbf{w}_j)$. As previously defined in Sect. 3.1, the function f_j^{ϱ} is unknown to us. Hence, the only viable solution without further knowledge of the problem is to perform a general approximation of the problem behavior. To properly define the estimation function, the following parameters are defined:

- $\mathbf{d}_j$, displacement vector for process workload in iteration j, $\mathbf{d}_j = \mathbf{w}_{j+1} - \mathbf{w}_j$
- $\mathbf{w}'_j$, candidate workload distribution for iteration $j+1$, $\mathbf{w}'_j = \mathbf{w}_j + \mathbf{d}_j$
- $x_{i,j}$ the amount of resources consumed by process i in iteration j, similar to how $\mathbf{w}_j$ is defined in Eq. 6,
- x'_j the estimated resource consumption of f'^{Q}_j,
- $r_{i,j} = \frac{x_{i,j}}{w_{i,j}}$, ratio of resources consumed per unit of work by process i in iteration j. This value is the inverse of the efficiency of a given processor.
- $\mathbf{r}_j$, vector of resource ratios in iteration j, where

$$\mathbf{r}_j = \left[r_{1,j} \; r_{2,j} \; \cdots \; r_{p,j} \right] \tag{9}$$

A function f'^{Q}_j that estimates f^{Q}_j needs to know the previous workload balance. A definition of f'^{Q}_j for energy consumption can be

$$f'^{Q}_j(\mathbf{w}'_j, \mathbf{r}_j) = \sum_{i=1}^{p} \left(r_{i,j} \cdot w'_{i,j} \right) = x'_j \tag{10}$$

where a candidate $\mathbf{w}'_j$ can be any feasible workload distribution in the problem space, $\mathbf{r}_j$ the ratios obtained with the gathered metrics and x'_j the estimated amount of energy consumed. Equation 10 can produce high error if the candidate $\mathbf{w'}_{j+1}$ is much different from $\mathbf{w}_j$.

In regular problems, it is possible to reach an optimal w_j that would yield minimum solutions for $f^{Q}_j(\mathbf{w}_j)$ and $f^{Q}_{j+1}(\mathbf{w}_j)$. When no further improvements are observed, the load is already balanced, and a stop condition has been reached for the heuristic g. Equation 11 defines this behavior:

$$\begin{aligned} &\text{let be } \; \mathbf{w}_{j+1} = \; g\Big(\mathbf{w}_j, \; f^{Q}_j\left(\mathbf{w}_j\right)\Big) \\ &\quad \text{if, } \; f^{Q}_j(\mathbf{w}_j) \cong f^{Q}_{j+1}(\mathbf{w}_{j+1}) \implies \text{local minimum for } \; f^{Q}_j \text{ has been found} \end{aligned} \tag{11}$$

This stop condition is used to define when two vectors $\mathbf{w}_j$ and $\mathbf{w}_{j+1}$ are similar, $\mathbf{w}_j \sim \mathbf{w}_{j+1}$:

$$\text{if, } \; f^{Q}_j(\mathbf{w}_j) \cong f^{Q}_{j+1}(\mathbf{w}_{j+1}) \implies \mathbf{w}_j \sim \mathbf{w}_{j+1} \tag{12}$$

Nevertheless, the nature of irregular problems forces g to lack a permanent stop condition, as defined previously in Eq. 5. These constraints require to address the trade-off between the quality of the workload distribution $\mathbf{w}$ obtained and the resources spent to obtain a good solution. In order to minimize the resource consumption of a given problem, our heuristic g has to achieve good results for both cases presented in Eqs. 4 and 5.

In practice, only the values for iteration j are needed in order to calculate those of iteration $j+1$ and the matrix parameters are only used as formal notation. Using these characteristics, g has to reassign the workload between processors on each step of a given iterative algorithm to minimize the resources x required per unit or work, increasing the efficiency of the system for that given resource. By finding a suitable $\mathbf{d}_j$, we can calculate a $\mathbf{w}_{j+1}$ that accomplishes this requirement.

4.2 Heuristic algorithm proposal

We propose to define the function g by applying techniques inspired by different metaheuristics. This naturally leads to introduce specific parameters, based in the former formalization:

– D_j^*, set of feasible displacements $\mathbf{d}_j$ in a given iteration.
– $P(g)$, probability of restarting the load balancing heuristic. This value is inspired by the acceptance probability of the widely known metaheuristic *Simulated Annealing* [14].

With this notation, we propose to calculate g following the steps defined in Algorithm 1. The index j has been eliminated for the algorithm variables, as it is implicit when g is called. When we find a valid vector of resources consumed, we calculate the consumption for each processor per amount of work performed. We then calculate the set D_j^* of feasible displacements. Since the cardinal of D_j^* increases rapidly with the amount of processes, this function should generate a subset of movements that should be scattered at the beginning of the problem and narrowed as the heuristic progresses. This property is based on the variable neighborhood search metaheuristic [19]. After that, we use the real metric x, execution time or energy consumption, to simulate the possible outcome of the current iteration with every displaced workload distribution w_{aux} and proceed with the best one, w'.

For regular problems, a simple execution of the algorithm is enough. However, an irregular one this approach requires additional work in order to perform the load balancing dynamically. Algorithm 2 illustrates the additional modifications required in an iterative one to complete the load balancing process. After the initial homogeneous distribution is performed, a small value is selected as $P(g)$. $State_j$ contains the partial solution of the iterative algorithm up to iteration j; therefore, the final solution for the iterative problem should be $State_N$.

Algorithm 1 g proposal

```
1:  function G                                   ▷ Input: w, x            ▷ Output: w'
2:      for all $p$ do
3:          $r[p] \leftarrow x[p] / w[p]$
4:      $D^* \leftarrow GenerateDistributions(p)$
5:      $w' \leftarrow$ nil
6:      $x_{best} \leftarrow \infty$
7:      for all $d \in D^*$ do
8:          $w_{aux} \leftarrow w + d$                        ▷ Ensure $w_{aux}$ values >= 0
9:          $x' \leftarrow f'^{\mathcal{Q}}(w_{aux}, resource\_ratios)$
10:         if $x' < x_{best}$ then
11:             $w' \leftarrow w_{aux}$
12:             $x_{best} \leftarrow x'$
13:     return $w'$
```

Algorithm 2 Heuristic applied to an iterative algorithm

```
1: procedure ITERATIVE PROBLEM
2:     for all $p$ do
3:         $w[p] \leftarrow 1/p$
4:     $do\_calibrate \leftarrow$ True
5:     $P(g) \leftarrow small\ value$
6:     $x_{last} \leftarrow \infty$
7:     for all $State_j \in Iterative\ Algorithm$ do
8:         $DistributeWork(w)$
9:         $Solve(State_j)$
10:        $GatherResults(w)$
11:        $x \leftarrow ObtainResourcesUsed()$
12:        if $do\_calibrate$ then
13:            $w' \leftarrow g(w, x)$
14:            $do\_calibrate \leftarrow similar(w, w')$        ▷ Similar returns bool and fulfills Equation 12.
15:        else
16:            if $random() < P(g)$ then
17:                $w' \leftarrow g(w, x)$
18:                $P(g) \leftarrow small\ value$
19:                $do\_calibrate \leftarrow$ True
20:            else
21:                $Increase(P(g))$
22:        $w \leftarrow w'$        ▷ $w'$ is assigned at least once
23:        $x_{last} \leftarrow x$
```

Workload is redistributed between all processes based on the metrics gathered after solving the partial solution $State_j$ in iteration j, until the stop condition is reached. Finally, due to the possible irregularity of the iterative problem, the probability to restart the procedure is included in Algorithm 2. Once the load balancing has ended, the probability $P(g)$ is incremented every iteration until the condition in line 16 of Algorithm 2 is met. When accepted, the load balancing process is restarted, and $P(g)$ is restored to its initial value.

5 Computational experience

5.1 Heuristic implementation

We have implemented the proposed heuristic to optimize energy consumption. It has been done in C, using the library known as EML [7] to gather the energy metrics required in the estimation function f'. The *GenerateDistributions* implementation introduces a new value to control the workload variation in each iteration. It is used to determine *similar* workloads, when the new distribution is close enough to the previous one. Finally, the maximum displacement is reduced every time the calibration is performed.

We study four use cases of dynamic programming iterative algorithms. They have been chosen by taking in consideration different parameters, their irregularity during the execution and their computational granularity: The Knapsack Problem (KP), the Resource Allocation Problem (RAP) and the Cutting Stock Problem (CSP). Each prob-

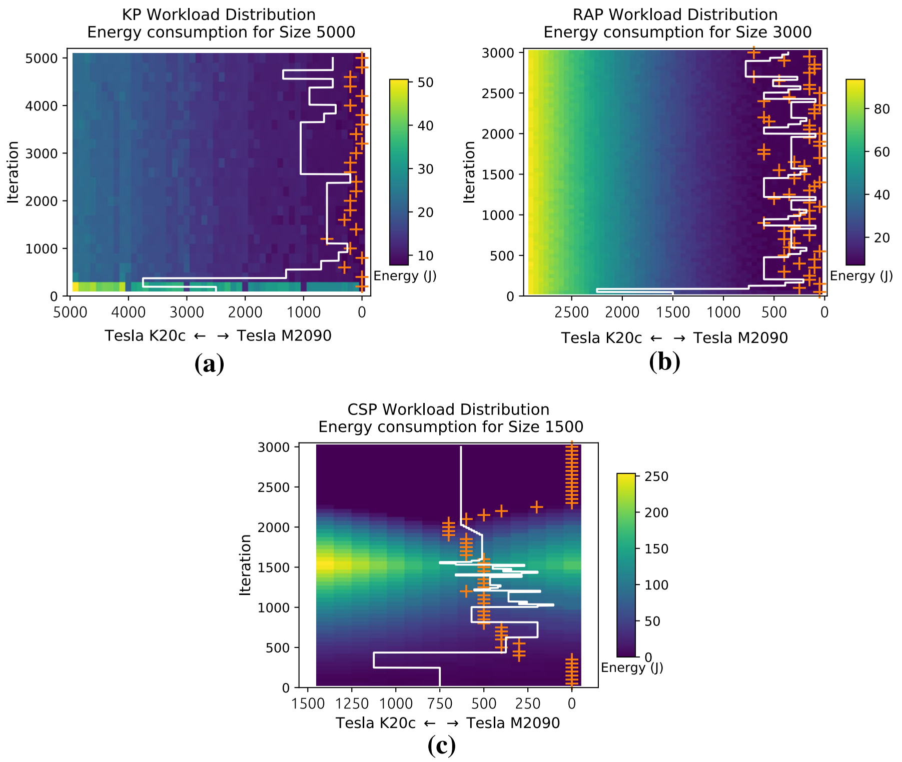

Fig. 1 Energy consumption surface of the workload with 2 processes. **a** KP, **b** RAP and **c** CSP

lem has different properties that allow to examine various computational granularities and memory requirements:

– The Knapsack Problem has small granularity and is regular through all its phases. Communications influence more the execution of this algorithm.
– The Resource Allocation Problem has an increasing granularity, and every value requires the previous ones to be calculated.
– The Cutting Stock Problem is solved by traversing a dynamic programming table diagonally and every iteration has a different amount of workload.

Figure 1 is a representation of executions that solve the presented cases, KP, RAP and CSP using our energy heuristic specific implementation. The mesh represents energy consumption during multiple executions with manual workload distributions using two processes. This data can be used to visualize the behavior of our algorithm in the solution space. The X-axis, labeled as *Tesla K20c* $\longleftrightarrow$ *Tesla M2090*, represents the amount of workload given to each process. The values in the Axis represent the workload of the process with the GPU *Tesla K20c*, while the rest of the workload, *ProblemSize* $-$ X is given to the *Tesla M2090*. The arrows in the label represent where the workload is reallocated when moving along the axis. The Y-axis represents the current *Iteration* of the algorithm. The different colors in the heatmap represent the energy consumed for each step and workload distribution. Additionally, a series of scattered points, represented as the symbol '$+$,' represent the optimal workload distribution for a given iteration, and the white line represents the data movements generated by an execution of our implemented load balancing heuristic for energy consumption.

In Fig. 1a, we can conclude that KP is a regular problem, except for the first iterations, and moreover, we can distinguish the desired space for our workload distribution to the right of X-axis. In Fig. 1b, we observe how the difference in energy consumption along the X-axis is much higher for the RAP. This is related to the data imbalance of every iteration in the problem. Finally, in Fig. 1c, we observe how the optimal workload configuration changes as the problem progresses in the CSP.

It can be observed how the heuristic trace moves toward the wrong workload configurations in some cases because the efficiency of a given process varies with its workload nonlinearly. Our simulated values, used to obtain the distribution $\mathbf{w}'$, are calculated using the current $\mathbf{w}$ and thus translate into wrong decisions delaying the path toward good distributions. However, the trace clearly reaches an area where we consider that the workload distributions are good enough for our execution. It is important to remark that we do not input any information referring to the problem into the heuristic. If required, it would be possible to set the restart probability of the algorithm to 0, reducing overhead if we know the local optima does not move during the execution, as is the case of Fig. 1b. However, it could be detrimental for cases as the one shown in Fig. 1c.

5.2 Results

We gathered computational results in the heterogeneous system in Table 1. The nodes have Debian 9 with the kernel 4.9.0-2-amd64 installed. The build and execution environments have GCC version 4.8.5, OpenMPI 3.0.0, compiled with said GCC compiler, and the CUDA 7.5 sdk version. Every compilation was done with the $-O2$ optimization flag. Energy measurement was performed for the GPUs using the Nvidia Management Library (NVML) driver on the EML library.

To obtain the total energy consumption of the execution, EML was also used to instrument the code by wrapping the *Iterative Problem* procedure described in Algorithm 2. The downside of using energy consumption to perform calibration is that the measurements are subject to the polling rate of the measurement devices and hence the usage of EML to abstract the algorithm from theses issues, which can cause potential losses in the application performance as the granularity is determined by the measurement device.

Table 2 collects a set of experiments that properly represents the strengths and weaknesses of our implementation. For each problem, CSP, RAP, and KP, we illustrate the average elapsed time and energy consumption for multiple sizes. For each size and metric, we compare the performance and energy consumption of a reference time (*Ref*), an even static distribution; a dynamic load balance performed using the load balancing algorithm of *Ull_Calibrate_lib* (*Calib*); and finally, developed implementation of the proposed heuristic algorithm to minimize energy consumption (*EnerH*). The best option is highlighted using a bold font.

Table 1 GPU cluster

Nodes	CPUs	Memory (GB)	GPU	No. of cores	RAM (GB)	Mem BW (GB/s)	Power (W)
Verode17	E5-2660	64	K20c	2496	5	208	225
Verode18	E5-2660	64	K40m	2880	12	288	235
Verode20	E5-2698v3	128	M2090	512	6	177.6	225

Table 2 Problem experimental data. KP and RAP share problem sizes

KP				RAP			CSP			
Size	*Ref*	*Calib*	*EnerH*	*Ref*	*Calib*	*EnerH*	Size	*Ref*	*Calib*	*EnerH*
Time (s)										
1000	**0.30**	0.52	0.52	1.36	**0.71**	1.08	500	**1.53**	1.68	2.36
2000	**0.93**	1.08	1.36	6.82	2.95	**2.57**	1000	**9.46**	9.49	14.70
3000	**2.01**	2.18	2.29	20.10	7.22	**5.91**	1500	31.02	**30.09**	31.53
4000	**3.46**	3.60	3.76	46.03	12.64	**9.78**	2000	73.64	**65.44**	66.76
5000	5.23	**5.19**	5.38	87.80	16.93	**15.99**	2500	145.9	133.1	**123.1**
6000	7.37	7.08	**6.62**	149.0	**22.76**	22.81	3000	249.3	217.1	**199.8**
7000	9.89	9.49	**8.68**	235.0	34.44	**26.56**	3500	393.8	341.6	**318.3**
Energy (J)										
1000	**35.2**	62.9	63.1	188.0	**93.0**	145.7	500	**213.3**	236.5	327.9
2000	**119.5**	137.3	175.4	994.5	415.7	**360.9**	1000	**1363**	1379	2077
3000	**266.3**	281.9	298.6	2902	1025	**845.5**	1500	4489	**4426**	4563
4000	**463.4**	471.1	494.1	6691	1827	**1418**	2000	10713	**9765**	9876
5000	705.9	**685.0**	711.5	12,843	2511	**2385**	2500	21,369	20,095	**18,573**
6000	1001	940.9	**883.1**	21,885	3449	**3390**	3000	37,022	33,485	**30,758**
7000	1348	1268	**1162**	34,587	5186	**3996**	3500	58,609	53,178	**49,385**

Ull_Calibrate_lib applies a dynamic load balance technique over execution time, by redistributing workload proportionally to the performance achieved during the problem execution. While this is not the objective of the Energy Heuristic, we consider that comparing our proposal to a homogeneous distribution only is not enough as proof of the proposal effectiveness. The usage of a different dynamic load balancing technique algorithm reinforces our claims despite addressing different objectives, as energy and time are heavily correlated variables in the target architecture. The performance impact of the energy consumption workload distributions are in these cases positive, as can be observed in our results. There is only one case where they differ, size 6000 of the RAP problem. This due to the polling rate of the energy consumption measurements and can be considered equal.

The KP has low computational demands for each unit of work. As shown in Table 2, there are performance losses for the executions up to size 5000, in which not performing load balancing at all is more efficient than load balancing. However, when the problem size increases, we can observe that both *Ull_Calibrate_lib* and the Energy Heuristic

Table 3 *UllMF* usage overhead during CSP

CSP	Time (s)			Energy (J)		
	Manual	*UllMF*	Error	Manual	*UllMF*	Error
1000	13.68	14.708	0.075	1800.3	1998.7	0.110
1500	26.64	27.47	0.031	3806.6	3952.8	0.038
2000	64.24	65.17	0.014	9348.9	9504.7	0.017
2500	114.96	116.47	0.013	17,188.6	17,476.6	0.017

present performance gains against a homogeneous distribution. The average gains of the energy heuristic, when it is applicable, is 6.5% of elapsed time and 8.2% of energy consumption, while *Ull_Calibrate_lib* obtains 2.9% and 5.0%, respectively, against the same reference.

Our RAP implementation using the skeleton shares all the code with the KP, with the exception of the CUDA kernel executed during the main loop. While KP is fine grained, the RAP is very compute intensive. Table 2 displays how both *Ull_Calibrate_lib* and the energy heuristic obtains very good performances, in terms of energy savings and performance. The RAP problem definition generates unbalances in the workload if a homogeneous distribution is used, obtaining more than 70% of gains when using any load balancing method. Comparing both dynamic load balancing methods to obtain a fair comparison, the energy heuristic performs better than or similarly good to *Ull_Calibrate_lib* in every case studied, with an average improvement of 13.6% elapsed time and 13.7% of energy consumption using the heuristic.

Finally, The CSP implementation is an example of a problem where a static load balancing technique cannot reach the optimal solution for the problem since the best workload distribution varies between iterations. Table 2 shows the best strategy for sizes from 1500 up to size 2000 is *Ull_Calibrate_lib* and obtains worse results than the heuristic as problem size increases. In this case, the heuristic obtains 12.5% better execution time and 10.4% energy consumption compared to the reference from size 1500. *Ull_Calibrate_lib*, on the other hand, obtains an average improvement of 9.8% and 7.0%, respectively.

Table 3 shows the overhead of our methodology. The comparison was performed by executing different problem sizes for the CSP and saving every distribution of work $\mathbf{w}$. Then, a different set of executions was performed by removing the code related to the heuristic and using the workload configurations stored earlier. The results gathered for both cases are the best case obtained for each problem size. The overhead starts at 7.5% and 11% for time and energy, respectively, and as the problem size increases this overhead is reduced to 1.4% and 0.8%, respectively. These results indicate that there is a relatively high impact of initialization that disappears as the problem size increases. Despite this shortcoming of the load balancing implementation, the gains in overall energy consumption are greater than the amount of energy wasted to perform the algorithm.

These results, though they could still be improved, illustrate the competitiveness of our approach despite its relative simplicity. They also illustrate the downsides of our implementation overhead and the limitations related to the usage of energy measurement tools. However, the generalization could be used to avoid some of these implementation issues, by supporting the load balancing technique with multiple objective functions.

6 Conclusion

We have presented an heuristic algorithm for parallel applications to perform dynamic load balancing in iterative problems, on heterogeneous systems. This heuristic is formalized using a generalization of the resource to optimize, in order to allow different optimization criteria for the decision making to distribute the workload. As proof of its effectiveness, we have implemented it using C and CUDA with relatively low overhead. The computational results obtained prove that our load balancing technique shows a considerable improvement for compute intensive problems. It also gets better results in some cases when compared against a different load balancing algorithm that has been proven to work. In the future, we plan to include CPU only environments and generate more complex strategies that change the objective resource to optimize using our formalization for iterative problems to cover the weak points of using only energy consumption to calibrate.

Acknowledgements This work was supported by the Spanish Ministry of Science, Innovation and Universities through the TIN2016-78919-R project, the Government of the Canary Islands, with the project ProID2017010130 and the grant TESIS2017010134, which is co-financed by the Ministry of Economy, Industry, Commerce and Knowledge of Canary Islands and the European Social Funds (ESF), operative program integrated of Canary Islands 2014-2020 Strategy Aim 3, Priority Topic 74(85%); the Spanish network CAPAP-H, and the European COST Action CHIPSET.

References

1. Acosta A, Almeida F (2013) Skeletal based programming for dynamic programming on multi-GPU systems. J Supercomput 65(3):1125–1136. https://doi.org/10.1007/s11227-013-0895-x
2. Agullo E, Demmel J, Dongarra J, Hadri B, Kurzak J, Langou J, Ltaief H, Luszczek P, Tomov S (2009) Numerical linear algebra on emerging architectures: the PLASMA and MAGMA projects. J Phys Conf Ser 180(1):012037
3. Almeida F, Arteaga J, Blanco V, Cabrera A (2015) Energy measurement tools for ultrascale computing: a survey. Supercomput Front Innov 2(2):64–76
4. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 28(5):755–768. https://doi.org/10.1016/j.future.2011.04.017 (Special Section: Energy efficiency in large-scale distributed systems)
5. Browne S, Dongarra J, Garner N, Ho G, Mucci P (2000) A portable programming interface for performance evaluation on modern processors. Int J High Perform Comput Appl 14(3):189–204. https://doi.org/10.1177/109434200001400303
6. Cabrera A, Acosta A, Almeida F, Blanco V (2017) Energy efficient dynamic load balancing over multi-GPU heterogeneous systems. In: Parallel Processing and Applied Mathematics—12th International Conference, PPAM 2017, Lublin, Poland, September 10–13, 2017, Revised Selected Papers, Part II, pp 123–132. https://doi.org/10.1007/978-3-319-78054-2_12

7. Cabrera A, Almeida F, Arteaga J, Blanco V (2014) Measuring energy consumption using EML (energy measurement library). Comput Sci Res Dev 30(2):135–143. https://doi.org/10.1007/s00450-014-0269-5
8. Dongarra J, Bosilca G, Chen Z, Eijkhout V, Fagg GE, Fuentes E, Langou J, Luszczek P, Pjesivac-Grbovic J, Seymour K, You H, Vadhiyar SS (2006) Self-adapting numerical software (SANS) effort. IBM J Res Dev 50(2/3):223–238
9. Garzón EM, Moreno JJ, Martínez JA (2017) An approach to optimise the energy efficiency of iterative computation on integrated GPU–CPU systems. J Supercomput 73(1):114–125. https://doi.org/10.1007/s11227-016-1643-9
10. Ge R, Feng X, Song S, Chang HC, Li D, Cameron KW (2010) Powerpack: energy profiling and analysis of high-performance systems and applications. IEEE Trans Parallel Distrib Syst 21(5):658–671
11. Guzek M, Kliazovich D, Bouvry P (2015) HEROS: energy-efficient load balancing for heterogeneous data centers. In: Pu C, Mohindra A (eds) 8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27–July 2, 2015, pp 742–749. IEEE. https://doi.org/10.1109/CLOUD.2015.103
12. Hendrickson B, Leland R (1995) An improved spectral graph partitioning algorithm for mapping parallel computations. SIAM J Sci Comput 16(2):452–469. https://doi.org/10.1137/0916028
13. Innovative Computing Laboratory (2011) University of Tennessee: the parallel linear algebra for scalable multi-core architectures (PLASMA) project. http://icl.cs.utk.edu/plasma/. Accessed May 2018
14. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680. https://doi.org/10.1126/science.220.4598.671
15. Kumar V, Grama A, Vempaty N (1994) Scalable load balancing techniques for parallel computers. J Parallel Distrib Comput 22(1):60–79. https://doi.org/10.1006/jpdc.1994.1070
16. Marqués R, Paulino H, Alexandre F, Medeiros PD (2013) Algorithmic skeleton framework for the orchestration of GPU computations. In: Wolf F, Mohr B, an Mey D (eds) Euro-Par 2013 Parallel Processing—19th International Conference, Aachen, Germany, August 26–30, 2013. Proceedings, Lecture Notes in Computer Science, vol 8097, pp 874–885. Springer. https://doi.org/10.1007/978-3-642-40047-6_86
17. Martínez J, Garzón E, Plaza A, García I (2009) Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. J Supercomput. https://doi.org/10.1007/s11227-009-0350-1
18. Meuer H, Strohmaier E, Dongarra J, Simon H Top500 list. http://www.top500.org/. Accessed May 2018
19. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100. https://doi.org/10.1016/S0305-0548(97)00031-2
20. Peláez I, Almeida F, Suárez F (2007) DPSKEL: a skeleton based tool for parallel dynamic programming. In: 7th International Conference on Parallel Processing and Applied Mathematics, PPAM2007. Gdansk, Poland, pp 1104–1113. https://doi.org/10.1007/978-3-540-68111-3_117
21. Reddy R, Lastovetsky A (2017) Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. IEEE Trans Comput 1(1):1–1. https://doi.org/10.1109/TC.2017.2742513
22. Richmond P, Romano D (2010) FLAME: Flexible large-scale agent modelling environment on the GPU. https://www.cs.utexas.edu/~flame/web/. Accessed Dec 2018
23. Steuwer M, Gorlatch S (2014) Skelcl: a high-level extension of opencl for multi-GPU systems. J Supercomput 69(1):25–33. https://doi.org/10.1007/s11227-014-1213-y
24. Takouna I, Rojas-Cessa R, Sachs K, Meinel C (2013) Communication-aware and energy-efficient scheduling for parallel applications in virtualized data centers. In: IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9–12, 2013, pp 251–255. IEEE. https://doi.org/10.1109/UCC.2013.50
25. The FLAME Project (2011) Flame: formal linear algebra methods environment. http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage. Accessed May 2018
26. Willebeek-LeMair MH, Reeves AP (1993) Strategies for dynamic load balancing on highly parallel computers. IEEE Trans Parallel Distrib Syst 4(9):979–993. https://doi.org/10.1109/71.243526
27. Xu C, Lau FC (1997) Load balancing in parallel computers: theory and practice. Kluwer Academic Publishers, Norwell