Noname manuscript No.
(will be inserted by the editor)

Energy Consumption Reduction for Asynchronous Message Passing Applications

Ahmed Fanfakh · Jean-Claude Charr · Raphaël Couturier · Arnaud Giersch

the date of receipt and acceptance should be inserted later

Abstract It is widely accepted that the asynchronous parallel methods are more suitable than the synchronous ones on a grid architecture. Indeed, they outperform the synchronous methods because they overlap the communications of the synchronous methods with computations. However, they also usually execute more iterations than the synchronous ones and thus consume more energy. To reduce the energy consumption of the CPUs executing such methods, the Dynamic voltage and frequency scaling (DVFS) technique can be used. It lowers the frequency of a CPU to reduce its energy consumption, but it also decreases its computing power. Therefore, the frequency that gives the best trade-off between energy consumption and performance must be selected.

This paper presents a new online frequency selecting algorithm for parallel iterative asynchronous methods running over grids. It selects a vector of frequencies that gives the best trade-off between energy consumption and performance. New energy and performance models were used in this algorithm to predict the execution time and the energy consumption of synchronous, asynchronous or hybrid iterative applications running over grids. The proposed algorithm was evaluated on the SimGrid simulator. The experiments showed that synchronously applying the proposed algorithm to the asynchronous version of the application reduces on average its energy consumption by 22% and speeds it up by 5.72%. Finally, the proposed algorithm was also compared to a method that uses the well known energy and delay product and the comparison results showed that it outperforms this method in terms of energy consumption and performance.

Keywords Energy optimization, DVFS, Asynchronous computing, Grid computing

Ahmed Fanfakh · Jean-Claude Charr · Raphaël Couturier · Arnaud Giersch
FEMTO-ST Institute, CNRS, Univ. Bourgogne Franche-Comté (UBFC)
IUT de Belfort-Montbéliard, 19 avenue du Maréchal Juin, 90016 Belfort, France
Email: ahmed.fanfakh_badri_muslim,jean-claude.charr,raphael.couturier,arnaud.giersch@univ-fcomte.fr

1 Introduction

Since the start of the digital revolution, the demand for computing power has been growing exponentially to solve bigger and more complex problems. At first, to meet these demands, the frequency of the CPUs was regularly increased until reaching the thermal limit. Then, many parallel and distributed architectures, such as multi-cores, clusters and grids, were implemented in order to obtain more computing power. This approach consists in using simultaneously many computing nodes to solve a big problem that cannot be solved on a single node. The computing nodes can collaborate by exchanging data through shared memory or by sending messages synchronously or asynchronously. Therefore, up to now the two most common approaches to get more computing power have been increasing the frequency of the processor and using more and more processors. Both approaches increase the energy consumption of the resulting computing architecture. Indeed, the power consumed by a processor polynomially increases when its frequency increases and a platform consisting of N computing nodes consumes as much as the sum of the power consumed by each computing node. Moreover, both approaches increase the heat generated by the platform and therefore a cooling infrastructure [40] which also consumes a lot of energy, must be implemented to keep the platform from overheating. High CPU's temperatures can also drastically increase its energy consumption, see [39] for more details.

Many techniques have been developed to reduce the energy consumption of a CPU while computing. The DVFS is a widely used process to reduce the energy consumption of a processor by lowering its frequency [31]. However, reducing the frequency of a CPU reduces its computing power (FLOPS) and thus the execution time of the applications running over it might be increased. Therefore, the trade-off between the energy reduction and the degradation in the execution time of the applications becomes an important optimization problem. In [6] and [7], two frequency selecting algorithms were proposed to reduce respectively the energy consumption of synchronous message passing iterative applications running over homogeneous and heterogeneous platforms. In this paper, a new frequency selecting algorithm for asynchronous iterative message passing applications running over grids is presented. An adaptation for hybrid methods, with synchronous and asynchronous communications, is also proposed. The algorithm and its adaptation select the vector of frequencies which simultaneously offers a maximum energy reduction and minimum performance degradation ratio. The algorithm has a very small overhead and works online without requiring any training nor any profiling.

This paper is organized as follows: Section 2 presents some related works from other authors. Section 3 describes the characteristics of the considered grid architecture. Models for predicting the performance and the energy consumption of both synchronous and asynchronous message passing programs running over a grid are explained in Section 4. Section 5 presents the objective function used to select the frequencies that maximize the reduction of energy consumption while minimizing the performance degradation of the program. Section 6 details the proposed frequency selecting algorithm. The iterative multi-splitting application which is a hybrid method and was used as a benchmark to evaluate the efficiency of the proposed algorithm, is described in Section 7. Section 8 presents the results of applying the algorithm on the multi-splitting application and executing it on different grid scenarios. It also shows the results of running three different power scenarios and comparing them. Moreover, in the last subsection, the proposed algorithm is compared to the energy and delay product (EDP) method. Finally, the paper ends with a summary and some perspectives.

2 Related works

A message passing application is, in general, composed of two types of sections, which are the computation and the communication sections. The communications can be done synchronously or asynchronously. In a synchronous message passing application, when a process synchronously sends a message to another node, it is blocked until the latter receives the message. During that time, there is no computation on both nodes and that period is called slack time. On the contrary, in an asynchronous message passing application, the asynchronous communications are overlapped by computations, thus, there is no slack time.

Many techniques have been used to reduce the energy consumption of message passing applications, such as scheduling, heuristics and DVFS. For example, different scheduling techniques which switch off the idle nodes to save their energy consumption, were presented in [35,12,25] and [11]. In [9] and [4], a heuristic to manage the workloads between the computing resources of the cluster and reduce their energy consumption, was published. However, the dynamic voltage and frequency scaling (DVFS) is the most popular technique to reduce the energy consumption of computing processors. In the next two subsections, some works on using DVFS to reduce the energy consumption of synchronous and asynchronous message passing applications, are presented.

2.1 Reducing the energy consumption of synchronous message passing applications

Most of the works in this field target the synchronous message passing applications because they are more common than the asynchronous ones and easier to work on. Some researchers have tried to reduce slack times in synchronous applications running over homogeneous clusters. These slack times can happen on such architectures if the distributed workloads over the computing nodes are imbalanced. Many works [43,37,27] reduce the frequency of each processor with DVFS operations to adapt its computing power to the size of its assigned task. In [6], we have presented a new energy model to predict the energy consumed by the processors in a homogeneous cluster and proposed an algorithm that selects the frequency providing the best tradeoff between energy consumption and performance.

Other works focus on reducing the energy consumption of synchronous applications running over heterogeneous architectures such as heterogeneous clusters or grids. When executing synchronous message passing applications on these architectures, slack times are generated when fast nodes have to communicate with slower ones. Indeed, the fast nodes have to wait for the slower ones to finish their computations to be able to communicate with them. In this case, some energy was saved in [7,18,32] and [8] by reducing the frequencies of the fast nodes with DVFS operations while minimizing slack times. The higher the initial slack times are, the more energy can be saved. In [7] and [16], we have adapted the work presented in [6] to the heterogeneity of the computing platform. The new energy and performance models take into consideration the heterogeneous characteristics of the processors and the frequency selection algorithm selects a new frequency for each processor according to its characteristics.

Heterogeneous platforms are not only composed of heterogeneous CPUs, they can also use GPUs. Many papers have tackled the subject of reducing the energy consumption of synchronous applications running on the latter architecture, such as [21] and [14]. Their results showed that using architectures mixing GPUs and CPUs and applying DVFS operations, is more energy efficient than only using CPUs. To reduce the energy consumption, all these works used online methods which are executed during the runtime of the application. However, some methods, such as those presented in [33] and [15], require some training or information about the application before executing the application and reducing its energy consumption.

2.2 Reducing the energy consumption of asynchronous message passing applications

To our knowledge, no work has been conducted yet on the optimization of the energy consumption of asynchronous message passing applications. Some works use asynchronous communications when applying DVFS operations on synchronous applications. For example, Hsu et al. [17] proposed an online adaptive algorithm that divides the synchronous message passing application into several time periods and selects the suitable frequency for each one. The algorithm asynchronously applies the new computed frequencies to overlap the multiple DVFS switching times with computation. Similarly, Zhu et al. [41] studied the difference between a synchronous or an asynchronous application of the frequency changing algorithm during the execution time of the program. The comparison showed that the proposed asynchronous scheduler is more energy efficient than the synchronous one. In [36], Vishnu et al. presented an energy efficient asynchronous agent that reduces slack times in a parallel program to reduce energy consumption. They used asynchronous communications in the proposed algorithm, which calls the DVFS algorithm many times during the execution time of the program. The three previous presented works were applied on applications running over homogeneous platforms.

In [1], the energy consumption of an asynchronous iterative linear solver running over a heterogeneous platform, is evaluated. The results showed that the asynchronous version of the application had less execution time than the synchronous one. Therefore, according to their energy model the asynchronous method consumes less energy. However, in their model they do not consider that during synchronous communications only static power which is significantly lower than dynamic power, is consumed.

This paper presents the following contributions:

1. new models to predict the energy consumption and the execution time of both synchronous and asynchronous iterative message passing applications running over a grid platform.
2. a new online algorithm that selects a vector of frequencies which gives the best trade-off between energy consumption and performance for asynchronous iterative message passing applications running over a grid platform. The algorithm has a very small overhead and does not require any training or profiling. The new algorithm can be applied synchronously or asynchronously on an iterative message passing application.

3 Characteristics of a grid architecture

A computing grid usually consists of many heterogeneous clusters which are geographically distant. Therefore, the clusters are interconnected through long distance networks with higher latency and lower bandwidth than the clusters' local networks. Each cluster is composed of homogeneous computing nodes interconnected via a fast local network. Nodes from two clusters can be heterogeneous with different computing power, memory and operating system. Figure 1 is an example of a grid with four different clusters. Inside each cluster, all the nodes are homogeneous but they have different specifications than the nodes of the other clusters.

In this paper, we are interested in running asynchronous iterative message passing applications on such platforms while reducing the energy consumption of the CPUs during the execution. To reduce the energy consumption of these applications while running on a grid, the heterogeneity of the clusters' nodes, such as nodes' computing powers (FLOPS), energy consumptions and CPU's frequency ranges, must be taken into account. To reduce the complexity of the experiments and focus on the heterogeneity of the nodes, the local networks of all the clusters are assumed to be identical, with the same latency and bandwidth. The networks connecting the clusters are also assumed to be homogeneous but they are slower than the local networks.

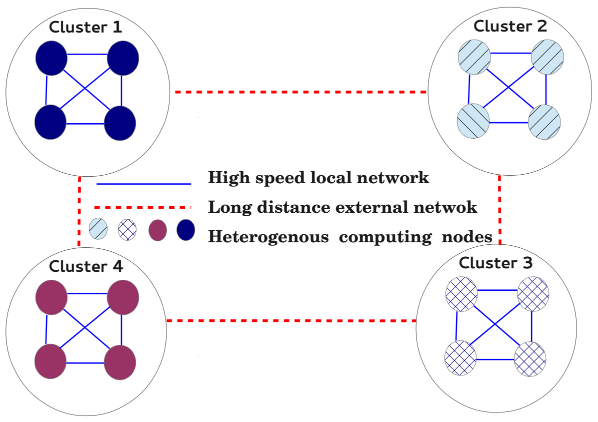


Fig. 1: A grid platform composed of heterogeneous clusters

4 Performance and energy consumption prediction models

4.1 Execution time of iterative synchronous or asynchronous message passing applications

An iterative application consists of a block of instructions that is repeatedly executed until convergence. A distributed iterative application with interdependent tasks requires, at each iteration, exchanging data between nodes to compute the distributed tasks. The communications between the nodes can be done synchronously or asynchronously.

In the synchronous model, each node has to wait to receive data from all its neighbors to compute its next iteration, see Figure 2. Since the tasks are synchronized, all the nodes execute the same number of iterations. Then, the overall execution time of the synchronous iterative message passing program is equal to the execution time of the slowest node which is the sum of all the iteration times executed by that node. An iteration is composed of some computations and some synchronous communications and its execution time is the sum of the execution times of these two parts. The communication time includes the slack time which is produced when the fast nodes have to wait for the slower nodes to finish their computations. The overall communication time for a synchronous application is the summation of periods of time that begin with an MPI call for sending or receiving a message until the message is synchronously sent or received. The overall execution time of an iterative synchronous message passing application with balanced tasks, running on the grid described above, is equal to the execution time of the slowest node in the slowest cluster executing a task.

In the asynchronous model, the fast nodes do not have to wait for the slower nodes to finish their computations to exchange data, see Figure 3. Therefore, there are no idle times between successive iterations. Each node executes the

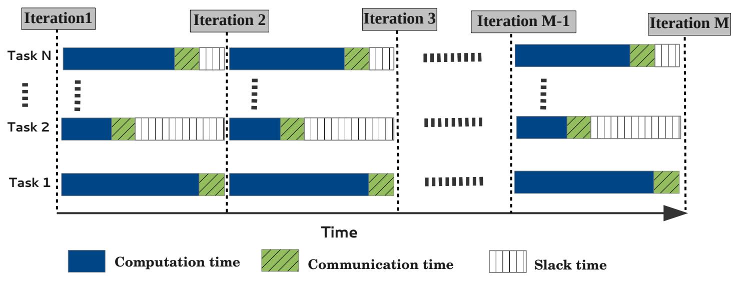


Fig. 2: The synchronous tasks model

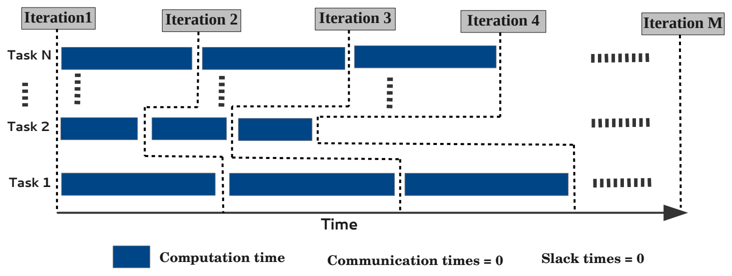


Fig. 3: The asynchronous tasks model

computation of its next iteration with the last received data from its neighbors and the asynchronous communications are overlapped by computations. Since there is no synchronization between the nodes, they do not execute the same number of iterations. The difference in the number of executed iterations between the nodes depends on the heterogeneity of the computing power of the nodes. The execution time of an asynchronous iterative message passing application is not equal to the execution time of the slowest node as in the synchronous mode because each node executes a different number of iterations. Moreover, the overall execution time is directly dependent on the method used to detect the global convergence of the asynchronous iterative application. The global convergence detection method might be synchronous or asynchronous and centralized or distributed. Therefore, in this paper the overall execution time of a message passing asynchronous iterative application is computed as the average of the execution time of all its parallel tasks.

4.2 Dynamic voltage and frequency scaling

Dynamic Voltage and Frequency Scaling (DVFS) is a process, implemented in modern processors, that reduces the energy consumption of a CPU by scaling down its voltage and frequency. Since DVFS lowers the frequency of a CPU and, consequently, its computing power, the execution time of a program running over that scaled down processor may increase, especially if the program is compute bound. The frequency reduction process can be expressed by the scaling factor S which is the ratio between the highest available frequency (F_{max}) for the CPU and the new frequency (F_{new}) applied to the CPU, as in Equation (1).

$$S = \frac{F_{max}}{F_{new}} \tag{1}$$

The execution time of a compute bound sequential program increases proportionally to the applied frequency scaling factor. On the other hand, only the computation sections of a message passing iterative application increase proportionally to the applied frequency scaling factor. The communications which could be synchronous or asynchronous, are not affected when the frequency is scaled down. Indeed, their execution times have not increased and the processors remain idle, for more details see [13]. However, since in the asynchronous communications model the communications are overlapped by computations, the frequency scaling factor affects the performance of the application during the whole execution time.

4.3 Execution times of iterative message passing applications after applying DVFS operations

In a grid, the nodes in each cluster have different characteristics, especially different frequency gears. Therefore, when applying DVFS operations on these nodes, they may get different scaling factors represented by a scaling vector: $(S_{11}, S_{12}, \dots, S_{NM_i})$ where S_{ij} is the scaling factor of processor j in cluster i. To be able to predict the execution time of synchronous iterative message passing applications running over a grid, for different vectors of scaling factors, the communication times and the computation times for all the tasks must be measured during the first iteration before applying any DVFS operation. Then, the execution time for one iteration of the synchronous application with any vector of scaling factors can be predicted using Equation (2).

$$T_{New} = \max_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cpOld_{ij}} \cdot S_{ij}) + \min_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cm_{ij}}) \tag{2}$$

where N is the number of clusters in the grid, M_i is the number of nodes in cluster i, $T_{cpOld_{ij}}$ and $T_{cm_{ij}}$ are the computation time and the communication time of processor j in cluster i during the first iteration. The execution time for one iteration is equal to the sum of the maximum computation time for all the nodes with the new scaling factors and the communication time of the slowest node in the grid. The latter is equal to the minimum local and external communication time which does not include any slack time. The overall execution time of the iterative application is equal to the execution time of one iteration as in Equation (2) multiplied by the number of executed iterations.

The execution time of one iteration of an asynchronous iterative message passing application, running on a grid, after applying the scaling factors is equal to the execution time of the synchronous application minus the communication times. Indeed, The communications are overlapped by computations. The execution time for one iteration in the application can be estimated as in Equation 3.

$$T_{New} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M_i} (T_{cpOld_{ij}} \cdot S_{ij})}{N \cdot M_i} \tag{3}$$

In this work, a hybrid (synchronous/asynchronous) message passing application [28] is being used. It is composed of two loops:

1. In the inner loop, at each iteration, the nodes in a cluster synchronously exchange data between them. There is no communication between nodes from different clusters.
2. In the outer loop, at each iteration, the nodes from different clusters asynchronously exchange their data between them because the network interconnecting the clusters has a high latency.

Therefore, the execution time of one outer iteration of such a hybrid application can be predicted by computing the average of the execution time of the slowest node in each cluster. The overall execution time of the asynchronous iterative applications can be predicted as follows:

$$T_{New} = \frac{\sum_{i=1}^{N} (\max_{j=1,\dots,M_i} (T_{cpOld_{ij}} \cdot S_{ij}) + \min_{j=1,\dots,M_i} (L_{tcm_{ij}}))}{N} \tag{4}$$

In Equation (4), the communication times $L_{tcm_{ij}}$ are only the communications between the local nodes because the communications between the clusters are asynchronous and overlapped by computations. Equations (2) and (4) are based on the equation presented in [7] which predicts the execution time of a message passing application running over a heterogeneous architecture.

4.4 Energy consumption model

The power consumed by a processor can be divided into two power metrics: the static power and the dynamic power, for more details see [22,29,31,42]. While the static power is consumed as long as the computing unit is turned on, the dynamic power is only consumed during computation times. The dynamic power P_d is related to the switching activity α, the load capacitance C_L, the supply voltage V and the operational frequency F, as shown in Equation (5).

$$P_d = \alpha \cdot C_L \cdot V^2 \cdot F \tag{5}$$

The static power P_s captures the leakage power as follows:

$$P_s = V \cdot N_{trans} \cdot K_{design} \cdot I_{leak} \tag{6}$$

where V is the supply voltage, N_{trans} is the number of transistors, K_{design} is a design dependent parameter and I_{leak} is a technology dependent parameter. The energy consumed by an individual processor to execute a sequential program can be computed as follows:

$$E_{ind} = P_d \cdot T_{cp} + P_s \cdot T \tag{7}$$

where T is the execution time of the program, T_{cp} is the computation time and $T_{cp} \leq T$. T_{cp} may be equal to T if there is no communication and no slack time.

The main objective of a DVFS operation is to reduce the overall energy consumption [20]. The operational frequency F depends linearly on the supply voltage V as follows:

$$V = \beta \cdot F \tag{8}$$

where β is a constant value. Equation (8) is used to study the change of the dynamic voltage with respect to various frequency values in [29]. The reduction process of the frequency can be expressed by the scaling factor S as in Equation (1). The new frequency F_{new} from Equation (1) can be calculated as follows:

$$F_{new} = S^{-1} \cdot F_{max} \tag{9}$$

Replacing equation (8) in equation (5) gives the following equation for the new dynamic power:

$$P_{dNew} = \alpha \cdot C_L \cdot {V_{new}}^2 \cdot F_{new} = \alpha \cdot C_L \cdot \beta^2 \cdot {F_{new}}^3 \tag{10}$$

Replacing F_{new} in equation 10 as in equation 9 gives the following equation for the new dynamic power consumption:

$$\begin{aligned} P_{dNew} &= \alpha \cdot C_L \cdot \beta^2 \cdot {F_{max}}^3 \cdot S^{-3} \\ &= \alpha \cdot C_L \cdot {V_{max}}^2 \cdot F_{max} \cdot S^{-3} = P_{dOld} \cdot S^{-3} \end{aligned} \tag{11}$$

where P_{dNew} and P_{dOld} are the dynamic power consumed with the new frequency and the maximum frequency respectively.

According to Equation 11 the dynamic power is reduced by a factor of S^{-3} when reducing the frequency by a factor of S [29]. Since the number of FLOPS executed by a CPU is proportional to the frequency of that CPU, the computation time is increased proportionally to S. The new dynamic energy is the dynamic power multiplied by the new computation time and is given by the following equation:

$$E_{dNew} = P_{dOld} \cdot S^{-3} \cdot (T_{cp} \cdot S) = S^{-2} \cdot P_{dOld} \cdot T_{cp} \tag{12}$$

The static power is related to the power leakage of the CPU and is consumed during computation and even when the processor is idle. As in [29, 42], the static power of a processor is considered as constant during idle and computation periods, and for all its available frequencies. The static energy is the static power multiplied by the execution time of the program. According to the execution time model for synchronous applications described in (2), the execution time of the program is the sum of the computation and the communication times. The computation time is linearly related to the frequency scaling factor, while this scaling factor does not affect the communication time. The static energy of a processor after scaling its frequency is computed as follows:

$$E_S = P_s \cdot (T_{cp} \cdot S + T_{cm}) \tag{13}$$

In the considered grid platform, each cluster may be composed of homogeneous nodes with processors that consume different dynamic and static powers, from the nodes of the other clusters. The dynamic and static powers for each node j in cluster i are noted as $P_{d_{ij}}$ and $P_{s_{ij}}$ respectively, but in a homogeneous cluster all the nodes have the same dynamic and static powers. Moreover, even though the tasks of the distributed message passing iterative application are load balanced, the computation time of each CPU j of cluster i, noted $T_{cp_{ij}}$, may vary a little bit. Therefore, slightly different frequency scaling factors may be computed for all the nodes of the cluster in order to decrease the overall energy consumption of the application and reduce slack times. The communication time of a processor j of cluster i is noted as $T_{cm_{ij}}$ and could contain slack times when synchronously communicating with slower nodes. In the same way, synchronous communications between clusters with different characteristics could generate slack times, see Figure 2. Consequently, all the nodes do not have equal communication times.

The overall energy consumption of a synchronous message passing application executed over a grid platform during one iteration is equal to the sum of the dynamic and static energies for each processor in each cluster. It can be computed as follows:

$$\begin{aligned} E = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (S_{ij}^{-2} \cdot P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{s_{ij}} \cdot \\ \max_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cp_{ij}} \cdot S_{ij}) + \min_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cm_{ij}})) \end{aligned} \tag{14}$$

The energy consumption of an asynchronous application running over a heterogeneous grid is the summation of the dynamic and static power of each node multiplied by the computation time of that node as in Equation (15). The computation time of each node is equal to the overall execution time of the node because the asynchronous communications are overlapped by computations.

$$E = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (S_{ij}^{-2} \cdot T_{cp_{ij}} \cdot (P_{d_{ij}} + P_{s_{ij}})) \tag{15}$$

It is common for distributed algorithms running over grids to have asynchronous external communications between clusters and synchronous ones between the nodes of the same cluster. In this hybrid communication scheme, the dynamic energy consumption can be computed in the same way as for the synchronous application with equation (12). However, since the nodes of different clusters are not synchronized and do not have the same execution time as in the synchronous application, the static energy consumption is different between them. The cluster's execution time is equal to the execution time of the slowest task in that cluster. The energy consumption of the asynchronous iterative message passing application running on a heterogeneous grid platform during one iteration can be computed as follows:

$$E = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (S_{ij}^{-2} \cdot P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{s_{ij}} \cdot \\ (\max_{j=1,\dots,M_i} (T_{cp_{ij}} \cdot S_{ij}) + \min_{j=1,\dots,M_i} (L_{tcm_{ij}}))) \quad (16)$$

Reducing the frequencies of the processors according to the vector of scaling factors $(S_{11}, S_{12}, \dots, S_{NM_i})$ may degrade the performance of the application and thus, increase the static energy consumed because the execution time is increased [19]. The overall energy consumption for the synchronous iterative application is equal to the energy consumed by one iteration as in (14) multiplied by the number of iterations of that application. While in the asynchronous applications, the overall energy consumption for the application can be computed by multiplying the energy consumption from one iteration of each cluster by the number of the iterations of that cluster, N_{iter_i}, as in equation (17).

$$E = \sum_{i=1}^{N} (\sum_{j=1}^{M_i} (S_{ij}^{-2} \cdot P_{d_{ij}} \cdot T_{cp_{ij}})) \cdot N_{iter_i} + \sum_{i=1}^{N} (\sum_{j=1}^{M_i} (P_{s_{ij}} \cdot \\ (\max_{j=1,\dots,M_i} (T_{cp_{ij}} \cdot S_{ij}) + \min_{j=1,\dots,M_i} (L_{tcm_{ij}})))) \cdot N_{iter_i} \quad (17)$$

5 Energy and performance tradeoff optimization

The main goal behind using the DVFS technique is to lower the frequency of the CPU dynamically so as to reduce the energy consumption of the CPU. Modern operating systems automatically adjust the frequency of the processor according to their needs using DVFS operations. However, the user can scale down the frequency of the CPU using the on-demand governor [38]. To get the best performance, the user should select the maximum frequency. Whereas, to reduce the energy consumption of the CPU, the CPU's frequency should be reduced. However, the lowest frequency does not always give the optimal energy consumption [30] because it increases the execution time and consequently increases the static energy which is linearly related to the execution time. Therefore, it is hard to predict the scaling factor that gives the best tradeoff between energy consumption and performance.

For a scaling factor S, computed as in equation (1), the overall energy consumption of the iterative application is reduced according to the two energy equations (14) and (16) and the execution time is increased as in equations (2) and (4). The relation between the energy consumption and the execution time is non linear and optimizing both terms is complex, for more details refer to [13]. Moreover, the energy and the execution time are not measured using the same metric. To solve this problem, the execution time of a synchronous iterative application is normalized by computing the ratio between the new execution time (after scaling down the frequencies of some processors) and the initial one (with maximum frequency for all nodes) as follows:

$$T_{Norm} = \frac{T_{New}}{T_{Old}} \tag{18}$$

where T_{New} is the new execution time computed as in equation (2) and T_{Old} is the old execution time without frequency scaling and computed as follows:

$$T_{Old} = \max_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cpOld_{ij}}) + \min_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cm_{ij}}) \tag{19}$$

Equation 18 can also be used to normalize the execution time of asynchronous iterative applications. However, T_{New} should be computed as in Equation 4 and T_{Old} computed as follows:

$$T_{Old} = \frac{\sum_{i=1}^{N} (\max_{j=1,\dots,M_i}(T_{cpOld_{ij}}) + \min_{j=1,\dots,M_i}(L_{tcm_{ij}}))}{N} \tag{20}$$

In the same way, the energy consumption for a synchronous or asynchronous application is normalized by computing the ratio between the consumed energy while scaling down the frequency and the consumed energy with maximum frequency for all nodes:

$$E_{Norm} = \frac{E_{Reduced}}{E_{Original}} \tag{21}$$

Where the reduced energy for synchronous and asynchronous applications, $E_{Reduced}$, is computed as in Equations (14) and (16) respectively. While the original energy consumption for synchronous and asynchronous applications, $E_{Original}$, is computed as in equations (22) and (23) respectively.

$$E_{original} = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{d_{ij}} \cdot T_{cpOld_{ij}}) + \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{s_{ij}} \cdot \max_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cpOld_{ij}}) + \min_{\substack{i=1,\dots N \\ j=1,\dots,M_i}} (T_{cm_{ij}}) \tag{22}$$

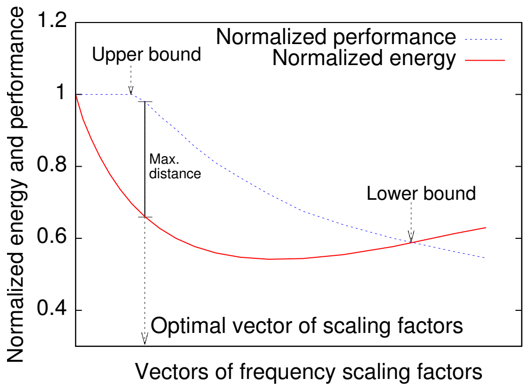


Fig. 4: The energy and performance relation

$$E_{original} = \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{d_{ij}} \cdot T_{cpOld_{ij}}) + \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{s_{ij}} \cdot (\max_{j=1,\dots,M_i}(T_{cpOld_{ij}}) + \min_{j=1,\dots,M_i}(L_{tcm_{ij}}))) \quad (23)$$

The normalized energy and the normalized execution time curves do not evolve (increase/decrease) in the same way. Indeed, according to Equations 21 and 18, the vector of frequency scaling factors $(S_{11}, S_{12}, \dots, S_{NM_i})$ reduces the energy consumption but increases the execution time. To optimize both terms at the same time, both curves should follow the same evolution. Therefore, the normalized execution time is inverted which gives the normalized performance equation, computed as follows:

$$P_{Norm} = \frac{T_{Old}}{T_{New}} \quad (24)$$

Then, the objective function can be modeled as the maximum distance between the normalized energy curve computed by Equation (21) and the normalized performance curve computed by Equation (24) over all available sets of scaling factors. The result of that objective function is the set of scaling factors that gives both the minimum energy consumption and the minimum execution time (maximum performance), see Figure 4. The objective function has the following form:

$$MaxDist = \max_{\substack{i=1,\dots N\\ j=1,\dots,M_i\\ k=1,\dots,F_i}} (\overbrace{P_{Norm}(S_{ijk})}^{\text{Maximize}} - \overbrace{E_{Norm}(S_{ijk})}^{\text{Minimize}}) \quad (25)$$

where N is the number of clusters, M_i is the number of nodes in cluster i and F_i is the number of available frequencies for each node in cluster i. The

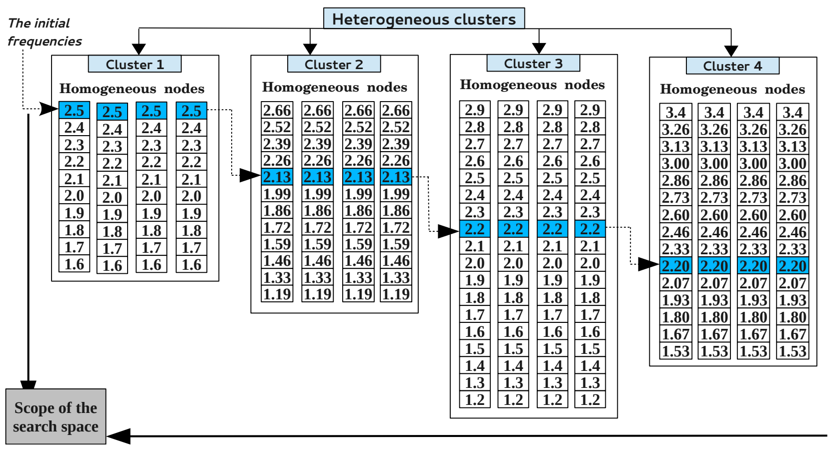


Fig. 5: Selecting the initial frequencies

objective function can work with any energy model or any power values for each node (static and dynamic powers). However, the most important energy reduction gain can be achieved when the energy curve has a convex form as shown in [29,33,42].

6 Frequency selection algorithm

The frequency selection algorithm (1) works online during the first iteration of the synchronous or asynchronous iterative message passing program running over a grid. The algorithm selects the set of frequency scaling factors $S_{opt_{11}}, S_{opt_{12}}, \dots, S_{opt_{NM_i}}$ which maximizes the distance, the tradeoff function (25), between the predicted normalized energy consumption and the normalized performance of the program. The algorithm is called just once in the iterative program and it uses information gathered at the first iteration to approximate the vector of frequency scaling factors that gives the best tradeoff. According to the returned vector of scaling factors, the DVFS algorithm (2) computes the new frequency for each node in the grid. It also shows where and when the proposed scaling algorithm is called in the iterative message passing program.

The proposed algorithm takes into account the heterogeneity of the nodes in the grid when selecting the vector of frequency scaling factors. For synchronous applications, it reduces slack times and for asynchronous applications, it balances the computing power between the nodes to reduce the difference in the number of computed iterations between them. At first, it selects the initial frequency scaling factors $S_{cp_{ij}}$ that increase the execution times of fast nodes and minimize the differences between the computation times of fast and slow nodes. The value of the initial frequency scaling factor for each node is inversely proportional to its computation time that was measured during the first iteration. These initial frequency scaling factors are computed as a ratio between the computation time of the slowest node in the grid and the computation time of the node j in the cluster i as follows:

$$S_{cp_{ij}} = \frac{\max_{i=1,2,\dots,N}(\max_{j=1,2,\dots,M_i} T_{cp_{ij}})}{T_{cp_{ij}}} \tag{26}$$

Using the initial frequency scaling factors computed in Equation (26), the algorithm (1) computes the initial frequencies for all nodes in all clusters as a ratio between the maximum frequency of node j in the cluster i and the computation scaling factor $S_{cp_{ij}}$ for that node as follows:

$$F_{ij} = \frac{F_{max_{ij}}}{S_{cp_{ij}}},\ i = 1,2,\dots,N,\ j = 1,2,\dots,M_i \tag{27}$$

If the computed initial frequency for a node is not available in the gears of that node, it is replaced by the nearest available frequency. In Figure 5, the nodes are sorted by their computing power in ascending order and the frequencies of the faster nodes are scaled down according to the computed initial frequency scaling factors. The resulting new frequencies are highlighted in Figure 5. This set of frequencies can be considered as a higher bound for the search space of the optimal vector of frequencies because selecting scaling factors higher than the higher bound will not improve the performance of the application and it will increase its overall energy consumption. Therefore the algorithm that selects the frequency scaling factors starts the search method from these initial frequencies and takes a downward search direction towards lower frequencies. The algorithm iterates on all the remaining frequencies, from the higher bound until all nodes reach their minimum frequencies or until the normalized computed distance is less than zero (the lower bound). At each iteration, it lowers the frequency of each node by one gear. Both the new overall energy consumption and execution time are computed according to the new scaling factors. The optimal set of frequency scaling factors is the set that provides the biggest distance according to the objective function (25). The lower bound is used to stop the algorithm when the computed distance is below 0, see Figure 4. A negative distance means that the normalized performance degradation is higher than the normalized energy saving.

7 Iterative multi-splitting method

Multi-splitting algorithms have initially been studied to solve linear systems of equations in parallel [24]. Thereafter, they were used to design non linear iterative algorithms and asynchronous iterative algorithms [2]. The principle of multi-splitting algorithms lies in splitting the system of equations, then solving each sub-system using a direct or an iterative method and finally combining the results in order to build a global solution. An iterative multi-splitting method

Algorithm 1 The scaling factors selection algorithm

```
Require:
    $N$ number of clusters in the grid.
    $M$ number of nodes in each cluster.
    $T_{cp_{ij}}$ array of all computation times for all nodes during one iteration and with the
        highest frequency.
    $T_{cm_{ij}}$ array of all communication times for all nodes during one iteration and with the
        highest frequency.
    $F_{max_{ij}}$ array of the maximum frequencies for all nodes.
    $P_{d_{ij}}$ array of the dynamic powers for all nodes.
    $P_{s_{ij}}$ array of the static powers for all nodes.
    $F_{diff_{ij}}$ array of the differences between two successive frequencies for all nodes.
Ensure: $S_{opt_{11}}, S_{opt_{12}} \ldots, S_{opt_{NM_i}}$, a vector of scaling factors that gives the optimal
    tradeoff between energy consumption and execution time
 1: $S_{cp_{ij}} \leftarrow \frac{\max_{i=1,2,\ldots,N}(\max_{j=1,2,\ldots,M_i}(T_{cp_{ij}}))}{T_{cp_{ij}}}$
 2: $F_{ij} \leftarrow \frac{F_{max_{ij}}}{S_{cp_i}},\ i = 1, 2, \cdots, N,\ j = 1, 2, \ldots, M_i.$
 3: Round the computed initial frequencies $F_i$ to the closest available frequency for each
    node.
 4: if (not the first frequency) then
 5:     $F_{ij} \leftarrow F_{ij} + F_{diff_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$
 6: end if
 7: $T_{Old} \leftarrow$ computed as in Equations (19) or (20).
 8: $E_{Original} \leftarrow$ computed as in Equations (22) or (23).
 9: $S_{opt_{ij}} \leftarrow 1,\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$
10: $Dist \leftarrow 0$
11: while (all nodes have not reached their minimum
        frequency or $P_{Norm} - E_{Norm} < 0$) do
12:     if (not the last frequency) then
13:         $F_{ij} \leftarrow F_{ij} - F_{diff_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$
14:         $S_{ij} \leftarrow \frac{F_{max_{ij}}}{F_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$
15:     end if
16:     $T_{New} \leftarrow$ computed as in Equations (2) or (4).
17:     $E_{Reduced} \leftarrow$ computed as in Equations (14) or (16).
18:     $P_{Norm} \leftarrow \frac{T_{Old}}{T_{New}}$
19:     $E_{Norm} \leftarrow \frac{E_{Reduced}}{E_{Original}}$
20:     if ($P_{Norm} - E_{Norm} > Dist$) then
21:         $S_{opt_{ij}} \leftarrow S_{ij},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$
22:         $Dist \leftarrow P_{Norm} - E_{Norm}$
23:     end if
24: end while
25: Return $S_{opt_{11}}, S_{opt_{12}}, \ldots, S_{opt_{NM_i}}$
```

repeatedly executes the same bloc of instructions until the global convergence of the system to a good approximation of the system's solution.

In this paper, we have used an asynchronous iterative multisplitting method to solve a 3D Poisson problem as described in [28]. The problem is divided into small 3D sub-problems and each one is solved by a parallel GMRES method. For more information about multi-splitting algorithms, interested readers are invited to consult the previous references.

Algorithm 2 DVFS algorithm

```
1: for $k = 1$ to *some iterations* do
2:     Computations section.
3:     Communications section
       (asynchronous or synchronous communications).
4:     if ($k = 1$) then
5:         Gather all times of computation and
           communication from each node.
6:         Call Algorithm 1.
7:         Compute the new frequencies from the
           returned optimal scaling factors.
8:         Set the nodes to the new frequencies.
9:     end if
10: end for
```

Table 1: The characteristics of the four types of nodes

node type	Simulated GFLOPS of one node	Max Freq. GHz	Min Freq. GHz	Diff. Freq. GHz	Dynamic power	Static power
A	40	2.50	1.20	0.100	20 W	4 W
B	50	2.66	1.60	0.133	25 W	5 W
C	60	2.90	1.20	0.100	30 W	6 W
D	70	3.40	1.60	0.133	35 W	7 W

8 Experiments

The heterogeneous scaling algorithm (HSA) (1) was applied to the parallel iterative multi-splitting method and executed over the SimGrid/SMPI simulator v3.10 [5] in order to evaluate its performance. The SimGrid simulator offers flexible tools to create a grid architecture and run message passing applications over it. The grid used in the experiments has four different types of nodes. Two types of nodes have different computing powers, frequency ranges, static and dynamic powers. Table 1 presents the characteristics of the four types of nodes. The specifications of the simulated nodes are similar to real Intel processors. Many grid configurations have been used in the experiments where the number of clusters and the number of nodes per cluster are equal to 4 or 8. For the grids composed of 8 clusters, two clusters of each type of nodes were used. The number of nodes per cluster is the same for all the clusters in a given grid.

The CPUs' constructors do not specify the amount of static and dynamic powers their CPUs consume. The maximum power consumption for each node's CPU was chosen to be proportional to its computing power (FLOPS). The dynamic power was assumed to represent 80 % of the overall power consumption while the remaining part is the static power. Similar assumptions were made in [6, 7, 29]. The clusters of the grid are connected via a long distance Ethernet network with 1 Gbit/s bandwidth, while inside each cluster the nodes are connected via a high-speed 10 Gbit/s bandwidth local Ethernet network. The

Table 2: The different experiment scenarios

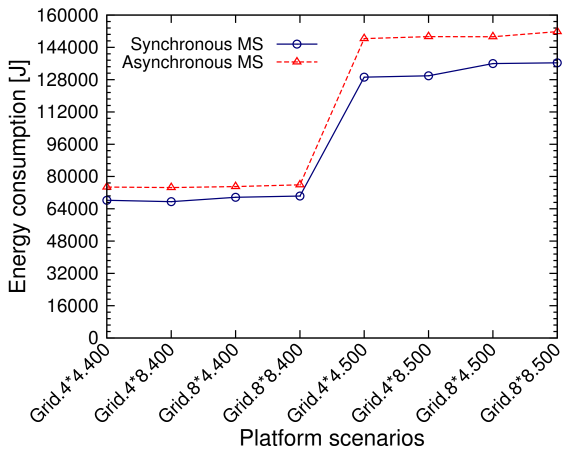
Platform scenario	Clusters number	Number of nodes in cluster	Vector size	Total number of nodes in grid
Grid.4*4.400	4	4	400^3	16
Grid.4*8.400	4	8	400^3	32
Grid.8*4.400	8	4	400^3	32
Grid.8*8.400	8	8	400^3	64
Grid.4*4.500	4	4	500^3	16
Grid.4*8.500	4	8	500^3	32
Grid.8*4.500	8	4	500^3	32
Grid.8*8.500	8	8	500^3	64

local networks have ten times less latency than the network connecting the clusters.

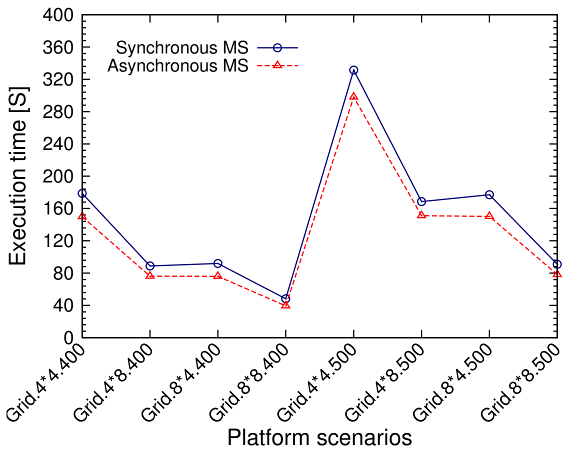
8.1 Energy consumption and execution time of the multi-splitting application

The multi-splitting (MS) method solves a three dimensional problem of size $N = N_x \cdot N_y \cdot N_z$. The problem is divided into equal subproblems which are distributed to the computing nodes of the grid and then solved. The experiments were conducted on problems of size $N = 400^3$ or $N = 500^3$ that require more than 12 and 24 Gigabyte of memory, respectively. Table 2 presents the different experiment scenarios with different numbers of clusters, nodes per cluster and problem sizes. A name, consisting in the values of these parameters was given to each scenario.

This section focuses on the execution time and the energy consumed by the MS application while running over the grid platform without using DVFS operations. The energy consumption of the synchronous and asynchronous MS was computed using the energy Equations 14 and 16 respectively. Figures 6a and 6b show the energy consumption and the execution time, respectively, of the multi-splitting application running over a heterogeneous grid with different numbers of clusters and nodes per cluster. The synchronous and the asynchronous versions of the MS application were executed over all the scenarios described in Table 2. As shown in Figure 6a, the asynchronous MS consumes more energy than the synchronous one. Indeed, the asynchronous application overlaps the asynchronous communications with computations and thus it executes more iterations than the synchronous one and has no slack times. More computations result in more dynamic energy consumption by the CPU in the asynchronous MS method. Since the dynamic power is chosen to be four times higher than the static power, the asynchronous MS method consumes more overall energy than the synchronous one. However, the execution times of the experiments, presented in Figure 6b, show that the execution times of the asynchronous MS are smaller than the execution times of the synchronous one. Indeed, in the asynchronous application, the fast nodes do not have to wait for the slower ones to exchange data. So there are no slack



(a) The energy consumption



(b) The execution time

Fig. 6: The energy consumption and the execution time of the multi-splitting application without applying the HSA algorithm

times and more iterations are executed by the fast nodes which accelerates the convergence to the final solution.

The synchronous and asynchronous MS methods scale well. The execution times of both methods decrease linearly with the increase of the number of computing nodes in the grid, whereas the energy consumption is approximately the same when the number of computing nodes increases. Therefore, the energy consumption of this application is not directly related to the number of computing nodes.

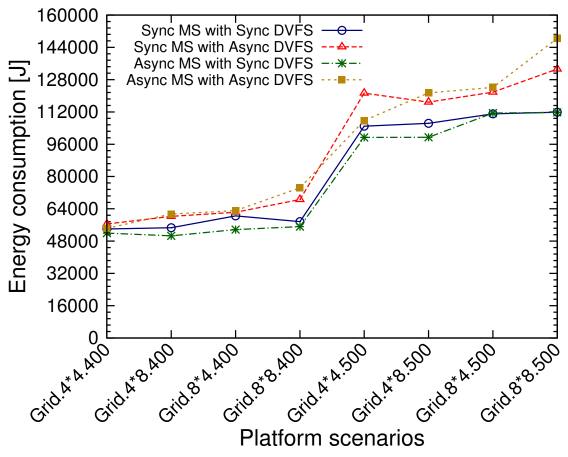
8.2 Experimental results of applying the scaling factor selection algorithm

The scaling factor selection algorithm 1 was applied to both synchronous and asynchronous MS applications which were executed over the 8 possible scenarios presented in table 2. The DVFS algorithm 2 needs to send and receive some information before calling the scaling factor selection algorithm algorithm 1. The communications of the DVFS algorithm can be applied synchronously or asynchronously which results in four different versions of the application: synchronous or asynchronous MS with synchronous or asynchronous DVFS communications. Figures 7a and 7b present the energy consumption and the execution time for the four different versions of the application running on all the scenarios in Table 2.

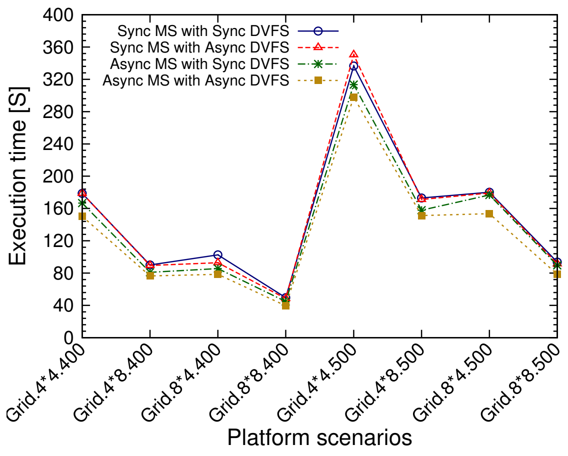
Figure 7a shows that the energy consumption of all four versions of the method, running over the 8 grid scenarios described in Table 2, are not affected by the increase in the number of computing nodes. The MS method without DVFS had the same behavior. On the other hand, Figure 7b shows that the execution time of the MS application with DVFS operations decreases in inverse proportion to the number of nodes. Moreover, it can be noticed that the asynchronous MS with synchronous DVFS consumes less energy than the other versions of the method. Two reasons explain this energy consumption reduction:

1. The asynchronous MS with synchronous DVFS version uses synchronous DVFS communications which allow it to apply the new computed frequencies at the begining of the second iteration. Thus, reducing the consumption of dynamic energy by the application from the second iteration until the end of the application. Whereas in asynchronous DVFS versions where the DVFS communications are asynchronous, the new frequencies cannot be computed at the end of the first iteration and consequently cannot be applied at the beginning of the second iteration. Indeed, since the performance information gathered during the first iteration is not sent synchronously at the end of the first iteration, fast nodes might execute many iterations before receiving the new computed frequencies, based on this information, and applying them. Therefore, many iterations might be computed by CPUs running on their highest frequency and consuming more dynamic energy than scaled down processors.
2. As shown in Figure 6b, the execution time of the asynchronous MS version is lower than the execution time of the synchronous MS version because there is no idle time in the asynchronous version and the communications are overlapped by computations. Since the consumption of static energy is proportional to the execution time, the asynchronous MS version consumes less static energy than the synchronous version.

The energy saving percentage is the ratio between the reduced energy consumption after applying the HSA algorithm and the original energy consumption of the synchronous MS method without DVFS. Whereas, the performance degradation percentage is the ratio between the original execution time of the



(a) The energy consumption with HSA



(b) The execution time with HSA

Fig. 7: The energy consumption and the execution time of different versions of the multi-splitting application after applying the HSA algorithm

synchronous MS method without DVFS and the new execution time after applying the HSA algorithm. Therefore, in this section, the synchronous MS method without DVFS serves as a reference for comparison with the other methods for the following terms: energy saving, performance degradation and the distance between both previous terms.

In Figure 8, the energy saving is computed for the four versions of the MS method which are the synchronous or asynchronous MS that apply synchronously or asynchronously the HSA algorithm. The fifth version is the asynchronous MS without any DVFS operations. Figure 8 shows that some versions have positive or negative energy saving percentages which means that

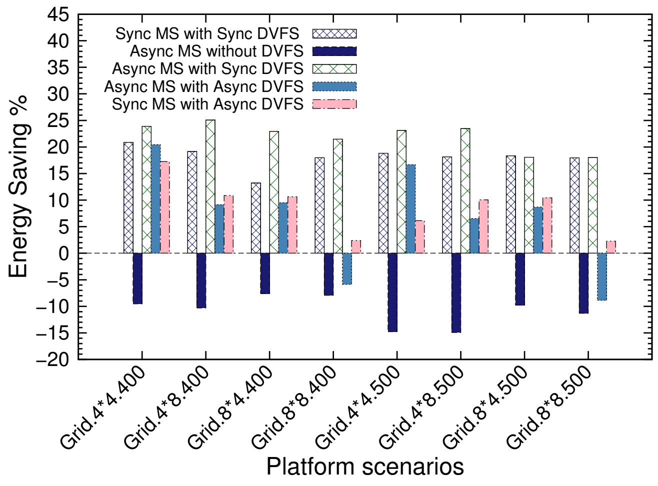


Fig. 8: The energy saving percentages after applying the HSA algorithm to the different versions and scenarios

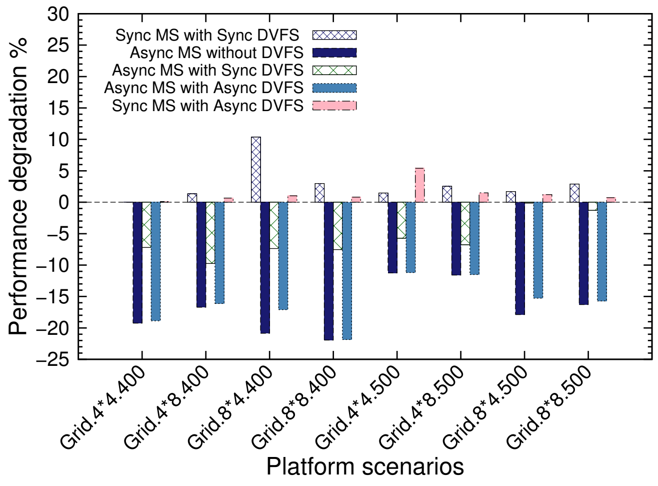


Fig. 9: The results of the performance degradation

the corresponding version respectively consumes less or more energy than the reference method. As in Figure 7a and for the same reasons presented above, the asynchronous MS with synchronous DVFS version gives the best energy saving percentage when compared to the other versions.

Figure 9 shows that some versions have negative performance degradation percentages which means that the new execution time of a given version of the application is less than the execution time of the synchronous MS without DVFS. Therefore, the version with the smallest negative performance degra-

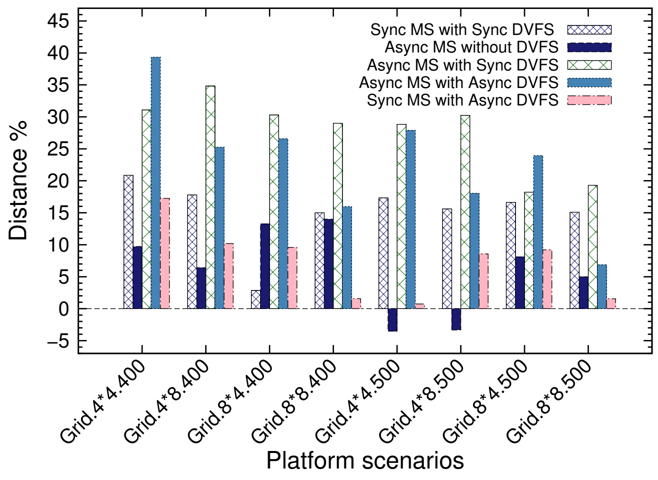


Fig. 10: The results of the tradeoff distance

dation percentage has actually the best speed up when compared to the other versions. The version that gives the best execution time is the asynchronous MS without DVFS which on average outperforms the synchronous MS without DVFS version by 16.9%. While the worst case scenario is the synchronous MS with synchronous DVFS where the performance is on average degraded by 2.9% when compared to the reference method.

The energy consumption and performance tradeoff between these five versions is presented in Figure 10. These distance values are computed as the differences between the energy saving and the performance degradation percentages as in the optimization function (25). Thus, the best MS version is the one that has the maximum distance between the energy saving and performance degradation. The distance can be negative if the energy saving percentage is inferior to the performance degradation percentage. The asynchronous MS, applying the HSA algorithm synchronously, gives the best distance which is on average equal to 27.72%. This version saves, on average, up to 22% of energy and, on average, speeds up the application by 5.72%. This overall improvement is due to combining asynchronous computing and the synchronous application of the HSA algorithm.

Both platform scenarios, Grid 4*8 and Grid 8*4, use the same total number of computing nodes but give different trade-off results. The versions applying the HSA algorithm and running over the Grid 4*8 platform, give higher distance percentages than those running on the Grid 8*4 platform. In the Grid 8*4 platform scenario more clusters are used than in the Grid 4*8 platform and thus the global system is divided into 8 small subsystems instead of 4. Indeed, each subsystem is assigned to a cluster and synchronously solved by the nodes of that cluster. Dividing the global system into smaller subsystems increases the number of outer iterations required for the global convergence of the system because the more the multi-splitting decomposes the system the higher the spectral radius is. For example, the asynchronous MS method, applying synchronously the HSA algorithm, requires on average 135 outer iterations when running over the Grid 4*8 platform and 148 outer iterations when running over the Grid 8*4 platform. The increase in the number of executed iterations over the Grid 8*4 platform justifies the increase in energy consumption by applications running over that platform.

8.3 Comparing the number of iterations executed by the different MS versions

The heterogeneity in the computing power of the nodes has a direct effect on the number of iterations executed by the nodes when running an asynchronous iterative message passing method. The fast nodes execute more iterations than the slower ones because the iterations are not synchronized. On the other hand, in the synchronous versions, all the nodes have the same number of iterations and have to wait for the slowest node to finish its iteration before starting the next iteration because the iterations are synchronized.

When the fast nodes asynchronously execute more iterations than the slower ones, they consume more energy without significantly improving the global convergence of the system. Reducing the frequency of the fast nodes will decrease the number of iterations executed by them. If all the nodes, the fast and the slow ones, execute almost the same number of iterations, the asynchronous application will consume less energy and its performance will not be significantly affected. Therefore, applying the HSA algorithm over asynchronous applications is very promising. In this section, the number of iterations executed by the asynchronous MS method, while solving a 3D problem of size 400^3 with and without applying the HSA algorithm, is evaluated. In Table 3, the standard deviation of the number of iterations executed by the asynchronous application over all the grid platform scenarios, is presented.

Table 3: The standard deviation of the numbers of iterations for different asynchronous MS versions running over different grid platforms

Grid platform	Standard deviation		
	Asyn. MS without HSA	Asyn. MS with Asyn. HSA	Asyn. MS with Syn. HSA
Grid.4*4.400	60.43	13.86	1.12
Grid.4*8.400	58.06	27.43	1.22
Grid.8*4.400	50.97	20.76	1.15
Grid.8*8.400	52.46	48.40	2.38

A small standard deviation value means that there is a very small difference between the numbers of iterations executed by the nodes. This also means that fast nodes did not uselessly execute more iterations than the slower ones and the application does not waste a lot of energy. As shown in Table 3, the asynchronous MS that applies synchronously the HSA algorithm has the best standard deviation value when compared to the other versions. Two reasons explain the advantage of this method:

1. The applied HSA algorithm selects new frequencies that reduce the computation power of the fast nodes while maintaining the computation power of the slower nodes. Therefore, it tries to balance the computation powers of the heterogeneous nodes as much as possible.
2. Applying the HSA algorithm synchronously scales down the frequencies of the CPUs at the end of the first iteration of the application. Therefore the computation power of all the nodes is balanced as much as possible since the beginning of the application. On the other hand, applying the HSA algorithm asynchronously onto the asynchronous MS application only changes the frequencies of the nodes after executing many iterations. Therefore, before the frequencies are scaled down, the fast nodes have enough time to execute many more iterations than the slower ones and consequently increase the overall energy consumption of the application.

Finally, the asynchronous MS version that does not apply the HSA algorithm gives the worst standard deviation values because there is a big difference between the numbers of iterations executed by the heterogeneous nodes. Therefore, this version consumes more energy than the other versions as shown in Figure 8.

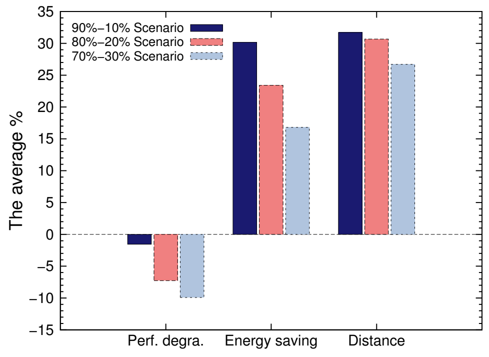
8.4 Comparing different power scenarios

In the previous sections, all the results were obtained by assuming that the dynamic and the static powers are respectively equal to 80% and 20% of the total power consumed by a CPU while computing at its highest frequency. The goal of this section is to evaluate the proposed frequency selection algorithm when these two power ratios are changed. Two new power scenarios are proposed in this section:

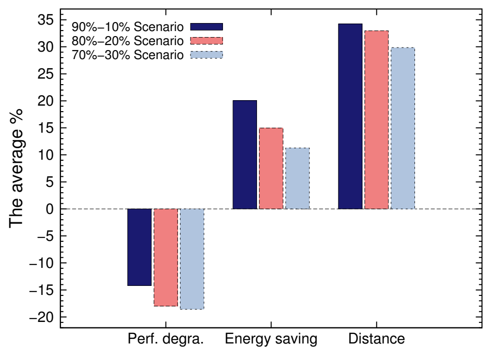
1. The dynamic and the static power are respectively equal to 90% and 10% of the total power consumed by a CPU while computing at its highest frequency.
2. The dynamic and the static power are respectively equal to 70% and 30% of the total power consumed by a CPU while computing at its highest frequency.

The asynchronous MS method was executed over two platform scenarios, the Grid 4*4 and Grid 8*4. Two versions of the asynchronous MS method, with synchronous or asynchronous application of the HSA algorithm, were evaluated on each platform scenario. The energy saving, performance degradation and distance percentages for both versions over both platform scenarios and with the three power scenarios are presented in figures 11a and 11b.

The displayed results are the average of the percentages obtained from multiple runs. Both figures show that the 90 %-10 % power scenario gives the

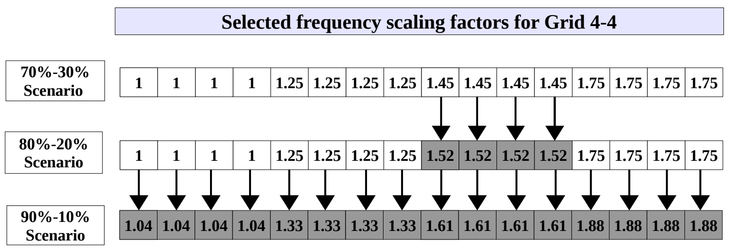


(a) Synchronous application of the HSA algorithm



(b) Asynchronous application of the HSA algorithm

Fig. 11: The results of the three power scenarios



Selected frequency scaling factors for Grid 4-4

70%-30% Scenario	1	1	1	1	1.25	1.25	1.25	1.25	1.45	1.45	1.45	1.45	1.75	1.75	1.75	1.75
80%-20% Scenario	1	1	1	1	1.25	1.25	1.25	1.25	1.52	1.52	1.52	1.52	1.75	1.75	1.75	1.75
90%-10% Scenario	1.04	1.04	1.04	1.04	1.33	1.33	1.33	1.33	1.61	1.61	1.61	1.61	1.88	1.88	1.88	1.88

Fig. 12: Comparison of the selected frequency scaling factors by the HSA algorithm for the three power scenarios

biggest energy saving percentages. The high dynamic power ratio pushes the HSA algorithm to select bigger scaling factors which decreases significantly the dynamic energy consumption. Figure 12 shows that the HSA algorithm selects in the 90 %-10 % power scenario higher frequency scaling factors than in the other power scenarios for the same application. Moreover, the 90 %-10 % power scenario has the smallest static power consumption per CPU which reduces the effect of the performance degradation, due to the scaling down of the frequencies of the CPUs, on the total energy consumption of the application. Finally, the 90 %-10 % power scenario gives higher distance percentages than the other two scenarios which means the difference between the energy reduction and the performance degradation percentages is the highest for this scenario. From these observations, it can be concluded that, in a platform with CPUs that consume low static power and high dynamic power, a lot of energy consumption can be reduced by applying the HSA algorithm but the performance degradation might be significant.

The energy saving percentages are the smallest with the 70 %-30 % power scenario. The high static power consumption in this scenario forces the HSA algorithm to select small scaling factors in order not to decrease the performance of the application significantly. Indeed, scaling down more the frequency of the CPUs will notably increase the total execution time and consequently increase the static energy consumption which will outweigh the reduction of dynamic energy consumption. Finally, since the dynamic power consumption ratio is relatively small in this power scenario, less dynamic energy reduction can be gained by lowering the frequencies of the CPUs than in the other power scenarios. On the other hand, the performance of the 70 %-30 % power scenario suffers the least from the application of the HSA algorithm. From these observations, it can be concluded that in a high static power model, just a small percentage of energy can be saved by applying the HSA algorithm.

The asynchronous application of the HSA algorithm improves on average the performance of the application more than the synchronous application of the HSA algorithm. This difference can be explained by the fact that applying the HSA algorithm synchronously scales down the frequencies of the CPUs after the first iteration, while applying the HSA algorithm asynchronously scales them down after many iterations, depending on the heterogeneity of the platform. However, for the same reasons as above, the synchronous application of the HSA algorithm reduces the energy consumption more than the asynchronous one even though, the method applying the first has a bigger execution time than the one applying the latter.

8.5 Comparing the HSA algorithm to the energy and delay product method

Many methods have been proposed to optimize the trade-off between the energy consumption and the performance of message passing applications. A well known optimization model used to solve this problem is the energy and delay product, $EDP = energy \times delay$. In [10,26,3], the researchers used equal weights for the energy and delay factors. However, others added some weights to the factors in order to direct the optimization towards more energy saving or less performance degradation. For example, in [23] they used the product

$EDP = energy \times delay^2$ which favors performance over energy consumption reduction.

In this work, the proposed scaling factors selection algorithm optimizes both the energy consumption and the performance at the same time and gives the same weight to both factors as in Equation 25. In this section, to evaluate the performance of the HSA algorithm, it is compared to the algorithm proposed by Spiliopoulos et al. [34]. The latter is an online method that selects for each processor the frequency that minimizes the energy and delay product in order to reduce the energy consumption of a parallel application running over a homogeneous multi-core platform. It gives the same weight to both metrics and predicts both the energy consumption and the execution time for each frequency gear as in the HSA algorithm. To fairly compare the HSA algorithm with the algorithm of Spiliopoulos et al., the same energy models, Equation (14) or (16), and execution time models, Equation (2) or (4), are used to predict the energy consumptions and the execution times. Furthermore, the EDP algorithm, as for the HSA algorithm, starts the search process from the initial frequencies that are computed in Equation (27). It stops the search process when it reaches the minimum available frequency for each processor.

The EDP objective function can be equal to zero when the predicted delay is equal to zero. Moreover, this product is equal to zero before applying any DVFS operation. To eliminate the zero values, the EDP function can be modified into the following form:

$$EDP = E_{Norm} \times (1 + D_{Norm}) \tag{28}$$

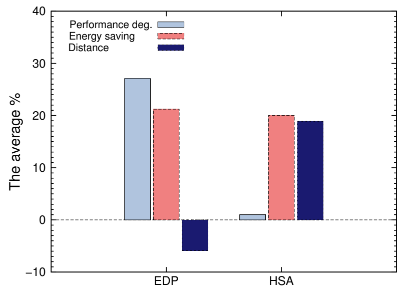
where E_{Norm} is the normalized energy consumption which is computed as in Equation (21) and D_{Norm} is the normalized delay of the execution time which is computed as follows:

$$D_{Norm} = 1 - P_{Norm} = 1 - (\frac{T_{old}}{T_{new}}) \tag{29}$$

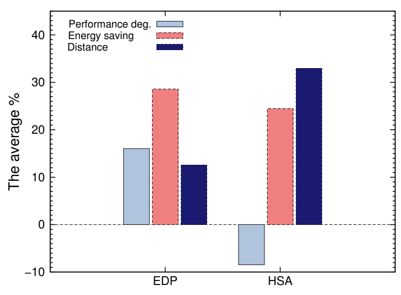
Where P_{Norm} is computed as in Equation (24).

The EDP algorithm was applied to the synchronous and asynchronous MS algorithm solving a 3D problem of size 400^3. Two platform scenarios, Grid 4*4 and Grid 4*8, were chosen for this experiment. The EDP method was applied synchronously and asynchronously to the MS application as for the HSA algorithm. The comparison results of the EDP and HSA algorithms are presented in Figures 13a, 13d,13c and 13d. Each of these figures presents the energy saving, performance degradation and distance percentages for one version of the MS algorithm. The results shown in these figures are also the average of the results obtained from running each version of the MS method over the two platform scenarios described above.

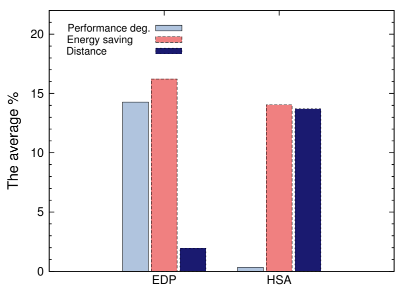
All the figures show that the proposed HSA algorithm outperforms the EDP algorithm in terms of energy saving and performance degradation. EDP gave for some scenarios negative trade-off values which means that the performance degradation percentages are higher than the energy saving percentages,



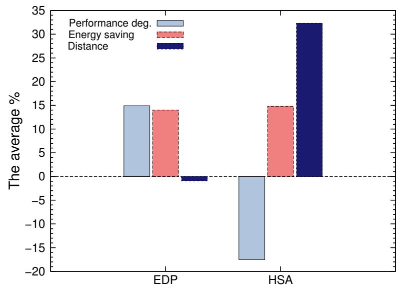
(a) Synchronous application of the frequency scaling selection method on the synchronous MS version



(b) Synchronous application of the frequency scaling selection method on the asynchronous MS version



(c) Asynchronous application of the frequency scaling selection method on the synchronous MS version



(d) Asynchronous application of the frequency scaling selection method on the asynchronous MS version

Fig. 13: The comparison results

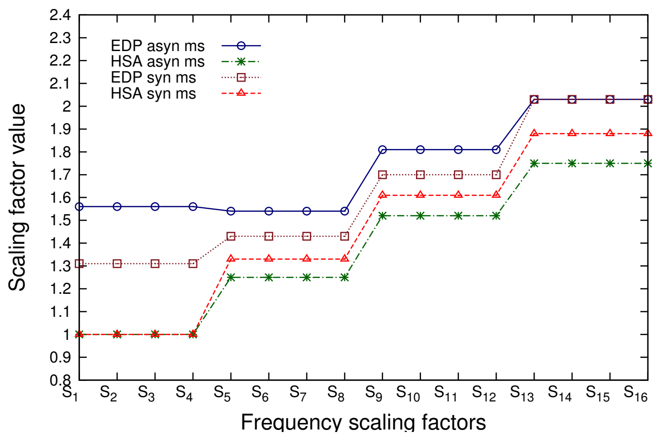


Fig. 14: Comparison of the selected frequency scaling factors by both algorithms over the Grid 4*4 platform scenario

while the HSA algorithm gives positive trade-off values over all scenarios. The frequency scaling factors selected by the EDP are most of the time higher than those selected by the HSA algorithm as shown in Figure 14. The results confirm that higher frequency scaling factors do not always give more energy saving, especially when the overall execution time is drastically increased. Therefore, the HSA method which computes the maximum distance between the energy saving and the performance degradation, is an effective method to optimize both metrics at the same time.

9 Conclusions

This paper presents a new online frequency selection algorithm. It selects the best vector of frequencies that maximizes the distance between the predicted energy consumption and the predicted execution time of an asynchronous message passing iterative application running over a grid. The algorithm uses new energy and performance models to predict the energy consumption and the execution time of a synchronous, asynchronous or hybrid message passing iterative application running over a grid. The proposed algorithm was evaluated on the SimGrid simulator while running a multi-splitting (MS) application. It was applied synchronously and asynchronously on a synchronous and an asynchronous version of the MS application. The results of the experiments show that applying synchronous HSA algorithm on an asynchronous MS application gives the best tradeoff between energy consumption reduction and performance. This scenario saves on average the energy consumption by 22% and reduces the execution time of the application by 5.72%. The HSA algorithm was also evaluated over three power scenarios. As expected, the algorithm selects different vectors of frequencies for each power scenario. The highest energy consumption reduction was achieved in the power scenario with the highest dynamic power and the lowest performance degradation was obtained in the power scenario with the highest static power. Finally, the proposed algorithm was compared to another method that uses the well known energy and delay product as an objective function. The comparison results showed that the proposed algorithm outperforms the latter by selecting a vector of frequencies that gives a better trade-off between the energy consumption reduction and the performance.

As a future work, it would be interesting to evaluate the HSA algorithm on other message passing iterative methods in order to see how it adapts to the characteristics of the new method. Furthermore, the methods should be executed on a real grid to check if the results obtained by the SimGrid simulator are comparable to those of a real experiment. Finally, it would be interesting to explore if a relation can be found between the numbers of asynchronous iterations required to global convergence and the applied frequencies to the nodes. The number of iterations required by each node for global convergence is not known in advance and the change in CPUs frequencies influences the number of iterations required by each node for global convergence.

Acknowledgment

This work has been partially supported by the Labex ACTION project (contract "ANR-11-LABX-01-01"). Computations have been performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté. As a PhD student, Mr. Ahmed Fanfakh, would like to thank the University of Babylon (Iraq) for supporting his work.

References

1. Anzt, H.: Asynchronous and multiprecision linear solvers. Ph.D. thesis, Karlsruher Institut für Technologie, Bade-Wurtemberg,Germany (2012)
2. Bahi, J., Contassot-Vivier, S., Couturier, R.: Parallel Iterative Algorithms: from sequential to grid computing, *Numerical Analysis and Scientific Computating*, vol. 1. Chapman and Hall/CRC (2007)
3. Baldassin, A., de Carvalho, J., Garcia, L., Azevedo, R.: Energy-performance tradeoffs in software transactional memory. In: Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on, pp. 147–154 (2012)
4. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation:Practice and Experience **24**(13), 1397–1420 (2012)
5. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (2014)
6. Charr, J.C., Couturier, R., Fanfakh, A., Giersch, A.: Dynamic frequency scaling for energy consumption reduction in distributed MPI programs. In: ISPA 2014: 12th IEEE International Symposium on Parallel and Distributed Processing with Applications, pp. 225–230. IEEE Computer Society, Milan, Italy (2014)
7. Charr, J.C., Couturier, R., Fanfakh, A., Giersch, A.: Energy consumption reduction with DVFS for message passing iterative applications on heterogeneous architectures. In: The 16th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing. IEEE Computer Society, INDIA (2015)
8. Chen, J.J., Huang, K., Thiele, L.: Dynamic frequency scaling schemes for heterogeneous clusters under quality of service requirements. Information Science and Engineering **28**(6), 1073–1090 (2012)
9. Cocana-Fernandez, A., Sanchez, L., Ranilla, J.: A software tool to efficiently manage the energy consumption of hpc clusters. In: Fuzzy Systems (FUZZ-IEEE), 2015 IEEE International Conference on, pp. 1–8 (2015)
10. Cochran, R., Hankendi, C., Coskun, A., Reda, S.: Identifying the optimal energy-efficient operating points of parallel workloads. In: Proceedings of the International Conference on Computer-Aided Design, pp. 608–615. IEEE Press, NJ, USA (2011)
11. Da Costa, G., de Assunção, M.D., Gelas, J.P., Georgiou, Y., Lefèvre, L., Orgerie, A.C., Pierson, J.M., Richard, O., Sayah, A.: Multi-facet approach to reduce energy consumption in clouds and grids: The green-net framework. In: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, pp. 95–104. ACM, New York, NY, USA (2010)
12. Da Costa, G., Gelas, J.P., Georgiou, Y., Lefevre, L., Orgerie, A.C., Pierson, J., Richard, O., Sharma, K.: The green-net framework: Energy efficiency in large scale distributed systems. In: Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pp. 1–8 (2009)
13. Freeh, V.W., Pan, F., Kappiah, N., Lowenthal, D.K., Springer, R.: Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, pp. 4a–4a. IEEE Computer Society, Washington, DC, USA (2005)

14. Ge, R., Vogt, R., Majumder, J., Alam, A., Burtscher, M., Zong, Z.: Effects of dynamic voltage and frequency scaling on a k20 gpu. In: Parallel Processing (ICPP), 2013 42nd International Conference on, pp. 826–833 (2013)
15. Guermouche, A., Triquenaux, N., Pradelle, B., Jalby, W.: Minimizing energy consumption of MPI programs in realistic environment. Computing Research Repository (2015)
16. Guérout, T., Monteil, T., Costa, G.D., Calheiros, R.N., Buyya, R., Alexandru, M.: Energy-aware simulation with {DVFS}. Simulation Modelling Practice and Theory **39**(0), 76 – 91 (2013)
17. Hsu, C.H., chun Feng, W.: A power-aware run-time system for high-performance computing. In: Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference, pp. 1–9 (2005)
18. Joshi, K.R., Hiltunen, M.A., Schlichting, R.D., Sanders, W.H.: Blackbox prediction of the impact of DVFS on end-to-end performance of multitier systems. ACM SIGMETRICS Performance Evaluation Review **37**(4), 59–63 (2010)
19. Kim, N.S., Austin, T., Blaauw, D., Mudge, T., Flautner, K., Hu, J.S., Irwin, M.J., Kandemir, M., Narayanan, V.: Leakage current: Moore's law meets static power **36**(12), 68–75 (2003)
20. Le Sueur, E., Heiser, G.: Dynamic voltage and frequency scaling: The laws of diminishing returns. In: Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10) (2010)
21. Ma, K., Li, X., Chen, W., Zhang, C., Wang, X.: Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In: Parallel Processing (ICPP), 2012 41st International Conference on, pp. 48–57 (2012)
22. Malkowski, K.: Co-adapting scientific applications and architectures toward energy-efficient high performance computing. Ph.D. thesis, The Pennsylvania State University, USA (2009)
23. Muralimanohar, N., Ramani, K., Balasubramonian, R.: Power efficient resource scaling in partitioned architectures through dynamic heterogeneity. In: In Proceedings of ISPASS (2006)
24. O'Leary, D.P., White, R.E.: Multi-splittings of matrices and parallel solution of linear systems. SIAM Journal on Algebraic Discrete Methods **6**(4), 630–640 (1985)
25. Orgerie, A.C., Lefevre, L., Gelas, J.P.: Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In: Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on, pp. 171–178 (2008)
26. Peraza, J., Tiwari, A., Laurenzano, M., L., C., Snavely: PMaC's green queue: a framework for selecting energy optimal DVFS configurations in large scale MPI applications. Concurrency Computation: Practice and Experience pp. 1–20 (2012)
27. Pietri, I., Sakellariou, R.: Energy-aware workflow scheduling using frequency scaling. In: Parallel Processing Workshops (ICCPW), 2014 43rd International Conference on, pp. 104–113 (2014)
28. Ramamonjisoa, C., Ziane Khodja, L., Laiymani, D., Giersch, A., Couturier, R.: Simulation of asynchronous iterative algorithms using simgrid. In: High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on, pp. 890–895 (2014)
29. Rauber, T., Rünger, G.: Analytical modeling and simulation of the energy consumption of independent tasks. In: Proceedings of the Winter Simulation Conference, pp. 245:1–245:13. Winter Simulation Conference (2012)
30. Rauber, T., Rünger, G., Schwind, M., Xu, H., Melzner, S.: Energy measurement, modeling, and prediction for processors with frequency scaling. The Journal of Supercomputing **70**(3), 1451–1476 (2014)
31. Rizvandi, N.B., Taheri, J., Zomaya, A.Y.: Some observations on optimal frequency selection in DVFS-based energy consumption minimization. Journal of Parallel and Distributed Computing **71**(8), 1154–1164 (2011)
32. Shelepov, D., Fedorova, A.: Scheduling on heterogeneous multicore processors using architectural signatures. In: Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with ISCA (2008)
33. Shen, H., Lu, J., Qiu, Q.: Learning based DVFS for simultaneous temperature, performance and energy management. In: ISQED, pp. 747–754 (2012)

34. Spiliopoulos, V., Kaxiras, S., Keramidas, G.: Green governors: A framework for continuously adaptive dvfs. In: International Green Computing Conference and Workshops (IGCC), pp. 1–8 (2011)
35. Thiam, C., Da Costa, G., Pierson, J.M.: Energy aware clouds scheduling using anti-load balancing algorithm : EACAB. In: 3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS 2014), pp. pp. 82–89. Barcelona, Spain (2014)
36. Vishnu, A., Song, S., Marquez, A., Barker, K., Kerbyson, D., Cameron, K., Balaji, P.: Designing energy efficient communication runtime systems: a view from pgas models. The Journal of Supercomputing **63**(3), 691–709 (2013)
37. Wang, L., Khan, S.U., Chen, D., Kołodziej, J., Ranjan, R., zhong Xu, C., Zomaya, A.: Energy-aware parallel task scheduling in a cluster. Future Generation Computer Systems **29**(7), 1661 – 1670 (2013)
38. Wen-Yew Liang Po-Ting Lai, C.W.C.: An energy conservation dvfs algorithm for the android operating system. The Journal of Convergence **1**(1), 93–100 (2010)
39. Zapater, M., Ayala, J.L., Moya, J.M., Vaidyanathan, K., Gross, K., Coskun, A.K.: Leakage and temperature aware server control for improving energy efficiency in data centers. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 266–269. San Jose, CA, USA (2013)
40. Zapater, M., Tuncer, O., Ayala, J., Moya, J., Vaidyanathan, K., Gross, K., Coskun, A.: Leakage-aware cooling management for improving server energy efficiency. Parallel and Distributed Systems, IEEE Transactions on **26**(10), 2764–2777 (2015)
41. Zhu, Y., Mueller, F.: Exploiting synchronous and asynchronous DVS for feedback EDF scheduling on an embedded platform. ACM Transactions on Embedded Computing **7**(1) (2007)
42. Zhuo, J., Chakrabarti, C.: Energy-efficient dynamic task scheduling algorithms for dvs systems. ACM Transactions on Embedded Computing **7**(2), 17:1–17:25 (2008)
43. Zong, Z., Manzanares, A., Ruan, X., Qin, X.: Ead and pebd: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. Computers, IEEE Transactions on **60**(3), 360–374 (2011)