Dynamic Frequency Scaling for Energy Consumption Reduction in Distributed MPI Programs

Jean-Claude Charr, Raphaël Couturier, Ahmed Fanfakh and Arnaud Giersch
FEMTO-ST Institute
University of Franche-Comté
IUT de Belfort-Montbéliard, Rue Engel Gros, BP 27, 90016 Belfort, France
Fax : (+33) 3 84 58 77 32
Email: {jean-claude.charr, raphael.couturier, ahmed.fanfakh, arnaud.giersch}@univ-fcomte.fr

***Abstract*—Dynamic Voltage Frequency Scaling (DVFS) can be applied to modern CPUs. This technique is usually used to reduce the energy consumed by a CPU while computing . Indeed, power consumption by a processor at a given instant is exponentially related to its frequency. Thus, decreasing the frequency reduces the power consumed by the CPU. However, it can also significantly affect the performance of the executed program if it is compute bound and a low CPU frequency is selected. The performance degradation ratio can even be higher than the saved energy ratio. Therefore, the chosen scaling factor must give the best possible tradeoff between energy reduction and performance.**

In this paper we present an algorithm that predicts the energy consumed with each frequency gear and selects the one that gives the best ratio between energy consumption reduction and performance. This algorithm works online without training or profiling and has a very small overhead. It also takes into account synchronous communications between the nodes that are executing the distributed algorithm. The algorithm has been evaluated over the SimGrid simulator while being applied to the NAS parallel benchmark programs. The results of the experiments show that it outperforms other existing scaling factor selection algorithms.

I. Introduction

The need and demand for more computing power have been increasing since the birth of the first computing unit and it is not expected to slow down in the coming years. To satisfy this demand, researchers and supercomputers constructors have been regularly increasing the number of computing cores and processors in supercomputers (for example in November 2013, according to the TOP500 list [1], the Tianhe-2 was the fastest supercomputer. It has more than 3 millions of cores and delivers more than 33 Tflop/s while consuming 17808 kW). This large increase in number of computing cores has led to large energy consumption by these architectures. Moreover, the price of energy is expected to continue its ascent according to the demand. For all these reasons energy reduction became an important topic in the high performance computing field. To tackle this problem, many researchers used DVFS (Dynamic Voltage Frequency Scaling) operations which reduce dynamically the frequency and voltage of cores and thus their energy consumption. Indeed, modern CPUs offer a set of acceptable frequencies which are usually called gears, and the user or the operating system can modify the frequency of the processor according to its needs. However, DVFS also degrades the performance of computation. Therefore researchers try to reduce the frequency to minimum when processors are idle (waiting for data from other processors or communicating with other processors). Moreover, depending on their objectives they use heuristics to find the best scaling factor during the computation. If they aim for performance they choose the best scaling factor that reduces the consumed energy while affecting as little as possible the performance. On the other hand, if they aim for energy reduction, the chosen scaling factor must produce the most energy efficient execution without considering the degradation of the performance. It is important to notice that lowering the frequency to minimum value does not always give the most energy efficient execution due to energy leakage. The best scaling factor might be chosen during execution (online) or during a pre-execution phase. In this paper, we present an algorithm that selects a frequency scaling factor that simultaneously takes into consideration the energy consumption by the CPU and the performance of the application. The main objective of HPC systems is to execute as fast as possible the application. Therefore, our algorithm selects the scaling factor online with very small footprint. The proposed algorithm takes into account the communication times of the MPI program to choose the scaling factor. This algorithm has ability to predict both energy consumption and execution time over all available scaling factors. The prediction achieved depends on some computing time information, gathered at the beginning of the runtime. We apply this algorithm to seven MPI benchmarks. These MPI programs are the NAS parallel benchmarks (NPB v3.3) developed by NASA [2]. Our experiments are executed using the simulator SimGrid/SMPI v3.10 [3] over an homogeneous distributed memory architecture. Furthermore, we compare the proposed algorithm with Rauber and Rünger methods [4]. The comparison's results show that our algorithm gives better energy-time tradeoff.

This paper is organized as follows: Section II presents related works from other authors. Section III shows the execution of parallel tasks and sources of idle times. It resumes the energy model of homogeneous platform. Section IV evaluates the performance of MPI program. Section V presents the energy-performance tradeoffs objective function. Section VI demonstrates the proposed energy-performance algorithm. Section VII verifies the performance prediction model and presents the results of the proposed algorithm. Also, It shows the comparison results. Finally, we conclude in Section VIII.

II. Related works

AG: Consider introducing the models sec. III maybe before related works

In this section, some heuristics to compute the scaling factor are presented and classified into two categories: offline and online methods.

A. *Offline scaling factor selection methods*

The offline scaling factor selection methods are executed before the runtime of the program. They return static scaling factor values to the processors participating in the execution of the parallel program. On one hand, the scaling factor values could be computed based on information retrieved by analyzing the code of the program and the computing system that will execute it. In [5], Azevedo et al. detect during compilation the dependency points between tasks in a parallel program. This information is then used to lower the frequency of some processors in order to eliminate slack times. A slack time is the period of time during which a processor that have already finished its computation, have to wait for a set of processors to finish their computations and send their results to the waiting processor in order to continue its task that is dependent on the results of computations being executed on other processors. Freeh et al. showed in [6] that the communication times of MPI programs do not change when the frequency is scaled down. On the other hand, some offline scaling factor selection methods use the information gathered from previous full or partial executions of the program. A part or the whole program is usually executed over all the available frequency gears and the the execution time and the energy consumed with each frequency gear are measured. Then an heuristic or an exact method uses the retrieved information to compute the values of the scaling factor for the processors. In [7], Xie et al. use an exact exponential breadth-first search algorithm to compute the scaling factor values that give the optimal energy reduction while respecting a deadline for a sequential program. They also present a linear heuristic that approximates the optimal solution. In [8] , Rountree et al. use a linear programming algorithm, while in [9], [10], Cochran et al. use multi logistic regression algorithm for the same goal. The main drawback for these methods is that they all require executing a part or the whole program on all frequency gears for each new instance of the same program.

B. *Online scaling factor selection methods*

The online scaling factor selection methods are executed during the runtime of the program. They are usually integrated into iterative programs where the same block of instructions is executed many times. During the first few iterations, many informations are measured such as the execution time, the energy consumed using a multimeter, the slack times, ... Then a method will exploit these measurements to compute the scaling factor values for each processor. This operation, measurements and computing new scaling factor, can be repeated as much as needed if the iterations are not regular. Kimura, Peraza, Yu-Liang et al. [11], [12], [13] used learning methods to select the appropriate scaling factor values to eliminate the slack times during runtime. However, as seen in [14], [15], machine learning methods can take a lot of time to converge when the number of available gears is big. To reduce the impact of slack times, in [16], Lim et al. developed an algorithm that detects the communication sections and changes the frequency during these sections only. This approach might change the frequency of each processor many times per iteration if an iteration contains more than one communication section. In [4], Rauber and Rünger used an analytical model that after measuring the energy consumed and the execution time with the highest frequency gear, it can predict the energy consumed and the execution time for every frequency gear . These predictions may be used to choose the optimal gear for each processor executing the parallel program to reduce energy consumption. To maintain the performance of the parallel program , they set the processor with the biggest load to the highest gear and then compute the scaling factor values for the rest of the processors. Although this model was built for parallel architectures, it can be adapted to distributed architectures by taking into account the communications. The primary contribution of this paper is presenting a new online scaling factor selection method which has the following characteristics :

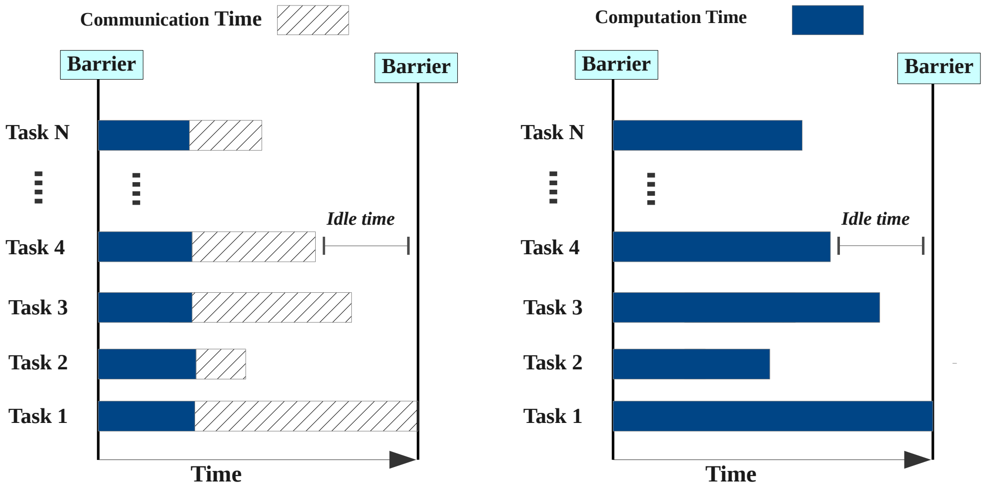
1) Based on Rauber's analytical model to predict the energy consumption and the execution time of the application with different frequency gears.
2) Selects the frequency scaling factor for simultaneously optimizing energy reduction and maintaining performance.
3) Well adapted to distributed architectures because it takes into account the communication time.
4) Well adapted to distributed applications with imbalanced tasks.
5) Has very small footprint when compared to other methods (e.g., [15]) and does not require profiling or training as in [9], [10].

III. Execution and energy of parallel tasks on homogeneous platform

A. *Parallel tasks execution on homogeneous platform*

A homogeneous cluster consists of identical nodes in terms of hardware and software. Each node has its own memory and at least one processor which can be a multi-core. The nodes are connected via a high bandwidth network. Tasks executed on this model can be either synchronous or asynchronous. In this paper we consider execution of the synchronous tasks on distributed homogeneous platform. These tasks can exchange the data via synchronous message passing. Therefore, the execution time of a task consists of the computation time and the communication time. Moreover, the synchronous communications between tasks can lead to slack times while tasks wait at the synchronization barrier for other tasks to finish their tasks (see figure (1a)). The imbalanced communications happen when nodes have to send/receive different amount of data or they communicate with different number of nodes. Another source of idle times is the imbalanced computations. This happens when processing different amounts of data on each processor (see figure (1b)). In this case the fastest tasks have to wait at the synchronization barrier for the slowest ones to begin the next task. In both cases the overall execution time of the program is the execution time of the slowest task as in EQ (1).

$$\mathit{Program\ Time} = \max_{i=1,2,\dots,N} T_i \qquad (1)$$



(a) Sync. imbalanced communications

(b) Sync. imbalanced computations

Figure 1: Parallel tasks on homogeneous platform

where T_i is the execution time of task i and all the tasks are executed concurrently on different processors.

B. Energy model for homogeneous platform

Many researchers [17], [4], [18], [19] divide the power consumed by a processor to two power metrics: the static and the dynamic power. While the first one is consumed as long as the computing unit is on, the latter is only consumed during computation times. The dynamic power P_{dyn} is related to the switching activity α, load capacitance C_L, the supply voltage V and operational frequency f, as shown in EQ (2).

$$P_{dyn} = \alpha \cdot C_L \cdot V^2 \cdot f \tag{2}$$

The static power P_{static} captures the leakage power as follows:

$$P_{static} = V \cdot N_{trans} \cdot K_{design} \cdot I_{leak} \tag{3}$$

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. Energy consumed by an individual processor E_{ind} is the summation of the dynamic and the static powers multiplied by the execution time [20], [18].

$$E_{ind} = (P_{dyn} + P_{static}) \cdot T \tag{4}$$

DVFS is a process that is allowed in modern processors to reduce the dynamic power by scaling down the voltage and frequency. Its main objective is to reduce the overall energy consumption [21]. The operational frequency f depends linearly on the supply voltage V, i.e., $V = \beta \cdot f$ with some constant β. This equation is used to study the change of the dynamic voltage with respect to various frequency values in [4]. The reduction process of the frequency can be expressed by the scaling factor S which is the ratio between the maximum and the new frequency as in EQ (5).

$$S = \frac{F_{max}}{F_{new}} \tag{5}$$

The value of the scaling factor S is greater than 1 when changing the frequency of the CPU to any new frequency value (*P-state*) in the governor. The CPU governor is an interface driver supplied by the operating system's kernel to lower a core's frequency. This factor reduces quadratically the dynamic power which may cause degradation in performance and thus, the increase of the static energy because the execution time is increased [20]. If the tasks are sorted according to their execution times before scaling in a descending order, the total energy consumption model for a parallel homogeneous platform, as presented by Rauber et al. [4], can be written as a function of the scaling factor S, as in EQ (6).

$$E = P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot S_1 \cdot N \tag{6}$$

where N is the number of parallel nodes, $T_i\ and\ S_i\ for\ i = 1, ..., N$ are the execution times and scaling factors of the sorted tasks. Therefore, $T1$ is the time of the slowest task, and S_1 its scaling factor which should be the highest because they are proportional to the time values T_i. The scaling factors are computed as in EQ (7).

$$S_i = S \cdot \frac{T_1}{T_i} = \frac{F_{max}}{F_{new}} \cdot \frac{T_1}{T_i} \tag{7}$$

In this paper we depend on Rauber and Rünger energy model EQ (6) for two reasons: (1) this model is used for any number of concurrent tasks, and (2) we compare our algorithm with Rauber and Rünger scaling factor selection method which is based on EQ (6). The optimal scaling factor is computed by minimizing the derivation for this equation which produces EQ (8).

$$S_{opt} = \sqrt[3]{\frac{2}{N} \cdot \frac{P_{dyn}}{P_{static}} \cdot \left(1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^3} \right)} \tag{8}$$

AG: The following 2 sections can be merged easily

IV. Performance evaluation of MPI programs

The performance (execution time) of parallel synchronous MPI applications depend on the time of the slowest task as in figure (1). If there is no communication and the application is not data bounded, the execution time of a parallel program is linearly proportional to the operational frequency and any DVFS operation for energy reduction increases the execution time of the parallel program. Therefore, the scaling factor S is linearly proportional to the execution time. However, in most of MPI applications the processes exchange data. During these communications the processors involved remain idle until the communications are finished. For that reason any change in the frequency has no impact on the time of communication [6]. The communication time for a task is the summation of periods of time that begin with an MPI call for sending or receiving a message till the message is synchronously sent or received. To be able to predict the execution time of MPI program, the communication time and the computation time for the slower task must be measured before scaling. These times are used to predict the execution time for any MPI program as a function of the new scaling factor as in EQ (9).

$$T_{new} = T_{Max\ Comp\ Old} \cdot S + T_{Max\ Comm\ Old} \tag{9}$$

In this paper, this prediction method is used to select the best scaling factor for each processor as presented in the next section.

V. Performance to energy competition

This section demonstrates our approach for choosing the optimal scaling factor. This factor gives maximum energy reduction taking into account the execution times for both computation and communication. The relation between the energy and the performance is nonlinear and complex, because the relation of the energy with scaling factor is nonlinear and with the performance it is linear see [6]. Moreover, they are not measured using the same metric. For solving this problem, we normalize the energy by calculating the ratio between the consumed energy with scaled frequency and the consumed energy without scaled frequency:

$$\begin{aligned} E_{Norm} &= \frac{E_{Reduced}}{E_{Original}} \\ &= \frac{P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2}\right) + P_{static} \cdot T_1 \cdot S_1 \cdot N}{P_{dyn} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2}\right) + P_{static} \cdot T_1 \cdot N} \end{aligned} \tag{10}$$

By the same way we can normalize the performance as follows:

$$P_{Norm} = \frac{T_{New}}{T_{Old}} = \frac{T_{Max\ Comp\ Old} \cdot S + T_{Max\ Comm\ Old}}{T_{Max\ Comp\ Old} + T_{Max\ Comm\ Old}} \tag{11}$$

The second problem is that the optimization operation for both energy and performance is not in the same direction. In other words, the normalized energy and the performance curves are not in the same direction see figure (2b). While the main goal is to optimize the energy and performance in the same time. According to the equations (10) and (11), the scaling factor S reduce both the energy and the performance simultaneously. But the main objective is to produce maximum energy reduction with minimum performance reduction. Many researchers used different strategies to solve this nonlinear problem for example see [15], [22], their methods add big overhead to the algorithm for selecting the suitable frequency. In this paper we present a method to find the optimal scaling factor S for optimizing both energy and performance simultaneously without adding big overheads. Our solution for this problem is to make the optimization process have the same direction. Therefore, we inverse the equation of normalize performance as follows:

$$P^{-1}_{Norm} = \frac{T_{Old}}{T_{New}} = \frac{T_{Old}}{T_{Max\ Comp\ Old} \cdot S + T_{Max\ Comm\ Old}} \tag{12}$$

Then, we can modelize our objective function as finding the maximum distance between the energy curve EQ (10) and the inverse of performance curve EQ (12) over all available scaling factors. This represent the minimum energy consumption with minimum execution time (better performance) at the same time, see figure (2a). Then our objective function has the following form:

$$MaxDist = \max_{j=1,2,\dots,F} (\overbrace{P^{-1}_{Norm}(S_j)}^{\text{Maximize}} - \overbrace{E_{Norm}(S_j)}^{\text{Minimize}}) \tag{13}$$

where F is the number of available frequencies. Then we can select the optimal scaling factor that satisfy EQ (13). Our objective function can work with any energy model or static power values stored in a data file. Moreover, this function works in optimal way when the energy curve has a convex form over the available frequency scaling factors as shown in [18], [4], [15].

VI. Optimal scaling factor for performance and energy

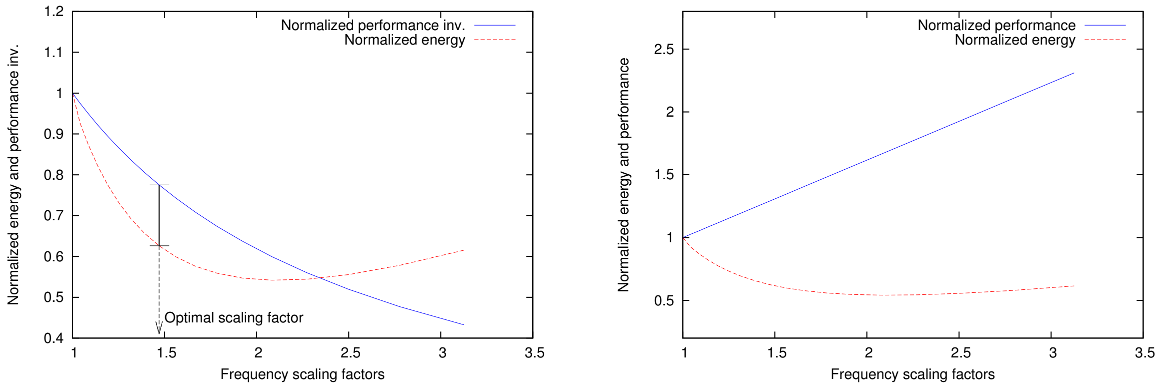
Algorithm 1 compute the optimal scaling factor according to the objective function described above.

The proposed algorithm works online during the execution time of the MPI program. It selects the optimal scaling factor after gathering the computation and communication times from the program after one iteration. Then the program changes the new frequencies of the CPUs according to the computed scaling factors. This algorithm has small execution time (between 0.00152 ms for 4 nodes to 0.00665 ms for 32 nodes). The algorithm complexity is O(F·N), where F is the number of available frequencies and N is the number of computing nodes. The algorithm is called just once during the execution of the program. The DVFS algorithm (2) shows where and when the algorithm is called in the MPI program.

After obtaining the optimal scaling factor, the program calculates the new frequency F_i for each task proportionally to its time value T_i. By substitution of EQ (5) in EQ (7), we can calculate the new frequency F_i as follows:

$$F_i = \frac{F_{max} \cdot T_i}{S_{optimal} \cdot T_{max}} \tag{14}$$

According to this equation all the nodes may have the same frequency value if they have balanced workloads, otherwise, they take different frequencies when having imbalanced workloads. Thus, EQ (14) adapts the frequency of the CPU to the nodes' workloads to maintain performance.



(a) Converted relation.

(b) Real relation.

Figure 2: The energy and performance relation

Algorithm 1 Scaling factor selection algorithm

```
Initialize the variable $Dist = 0$
Set dynamic and static power values.
Set $P_{states}$ to the number of available frequencies.
Set the variable $F_{new}$ to max. frequency, $F_{new} = F_{max}$
Set the variable $F_{diff}$ to the difference between two successive frequencies.
for $j := 1$ to $P_{states}$ do
    - $F_{new} = F_{new} - F_{diff}$
    - $S = \frac{F_{max}}{F_{new}}$
    - $S_i = S \cdot \frac{T_1}{T_i} = \frac{F_{max}}{F_{new}} \cdot \frac{T_1}{T_i} \; for \; i = 1, ..., N$
    - $E_{Norm} = \frac{P_{dyn} \cdot S_1^{-2} \cdot \left( T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot S_1 \cdot N}{P_{dyn} \cdot \left( T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot N}$
    - $P_{NormInv} = T_{old}/T_{new}$
    if $(P_{NormInv} - E_{Norm} > Dist)$ then
        $S_{opt} = S$
        $Dist = P_{NormInv} - E_{Norm}$
    end if
end for
Return $S_{opt}$
```

Algorithm 2 DVFS

```
for $k := 1$ to $Some - Iterations$ do
    -Computations Section.
    -Communications Section.
    if $(k = 1)$ then
        -Gather all times of computation and
          communication from each node.
        -Call algorithm 1 with these times.
        -Compute the new frequency from the
          returned optimal scaling factor.
        -Set the new frequency to the CPU.
    end if
end for
```

VII. Experimental results

Our experiments are executed on the simulator SimGrid/SMPI v3.10. We configure the simulator to use a homogeneous cluster with one core per node. The detailed characteristics of our platform file are shown in the table (I). Each node in the cluster has 18 frequency values from 2.5 GHz to 800 MHz with 100 MHz difference between each two successive frequencies. The simulated network link is 1 GB Ethernet (TCP/IP). The backbone of the cluster simulates a high performance switch.

Table I: Platform file parameters

Max Freq.	Min Freq.	Backbone Bandwidth	Backbone Latency	Link Bandwidth	Link Latency	Sharing Policy
2.5 GHz	800 MHz	2.25 GBps	0.5 μs	1 GBps	50 μs	Full Duplex

A. Performance prediction verification

In this section we evaluate the precision of our performance prediction method based on EQ (9) by applying it the NAS benchmarks. The NAS programs are executed with the class B option for comparing the real execution time with the predicted execution time. Each program runs offline with all available scaling factors on 8 or 9 nodes (depending on the benchmark) to produce real execution time values. These scaling factors are computed by dividing the maximum frequency by the new one see EQ (5). In our cluster there are 18 available frequency states for each processor. This lead to 18 run states for each program. We use seven MPI programs of the NAS parallel benchmarks: CG, MG, EP, FT, BT, LU and SP. Figure (3) presents plots of the real execution times and the simulated ones. The maximum normalized error between the predicted execution time and the real time (SimGrid time) for all programs is between 0.0073 to 0.031. The better case is for CG and the worse case is for LU.

B. The experimental results for the scaling algorithm

The proposed algorithm was applied to seven MPI programs of the NAS benchmarks (EP, CG, MG, FT, BT, LU and SP) which were run with three classes (A, B and C). For

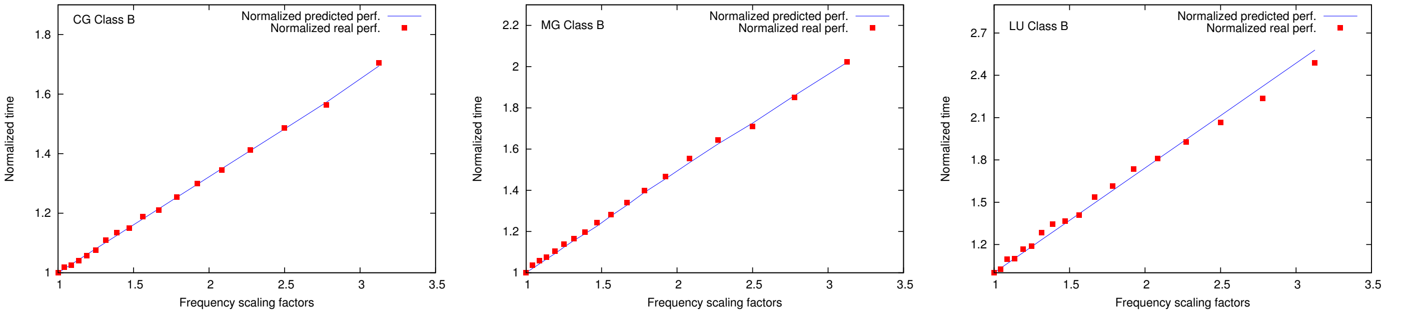


Figure 3: Comparing predicted to real execution time

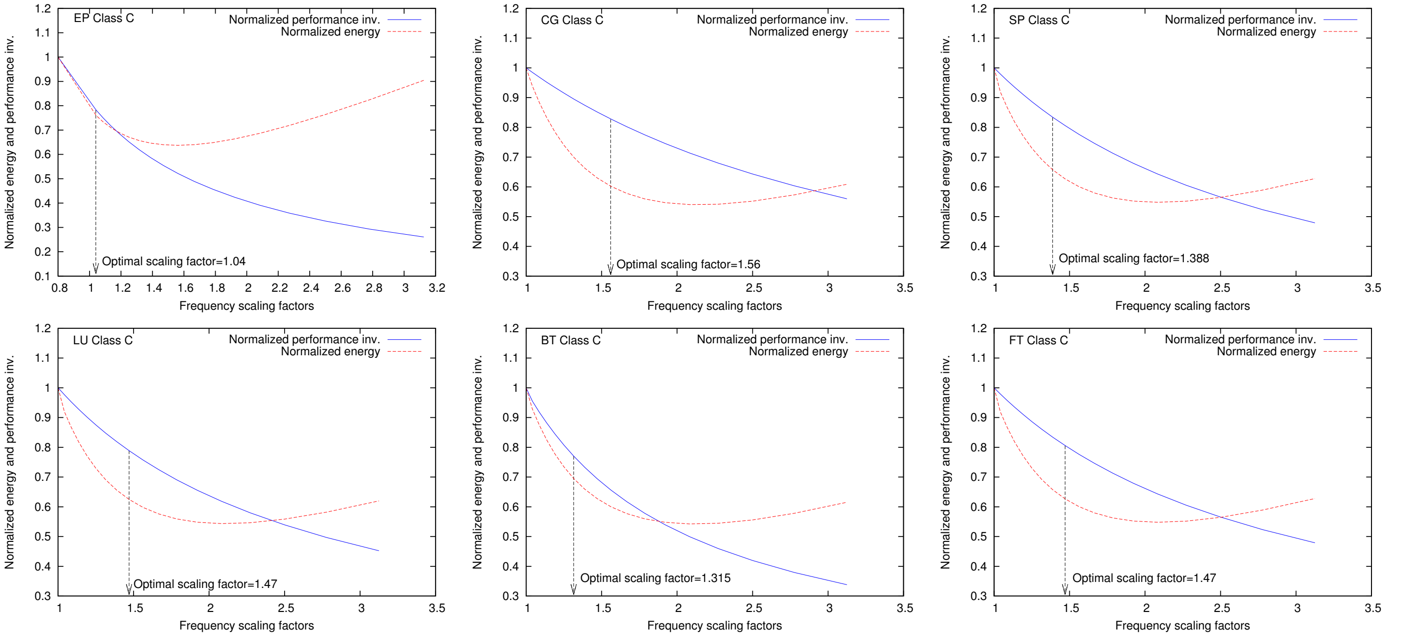


Figure 4: Optimal scaling factors for the predicted energy and performance of NAS benchmarks

each instance the benchmarks were executed on a number of processors proportional to the size of the class. Each class represents the problem size ascending from the class A to C. Additionally, depending on some speed up points for each class we run the classes A, B and C on 4, 8 or 9 and 16 nodes respectively. Depending on EQ (6), we measure the energy consumption for all the NAS MPI programs while assuming the power dynamic with the highest frequency is equal to 20 W and the power static is equal to 4 W for all experiments. These power values were also used by Rauber and Rünger in [4]. The results showed that the algorithm selected different scaling factors for each program depending on the communication features of the program as in the plots (4). These plots illustrate that there are different distances between the normalized energy and the normalized inverted performance curves, because there are different communication features for each benchmark. When there are little or not communications, the inverted performance curve is very close to the energy curve. Then the distance between the two curves is very small. This leads to small energy savings. The opposite happens when there are a lot of communication, the distance between the two curves is big. This leads to more energy savings (e.g. CG and FT), see table (II). All discovered frequency scaling factors optimize both the energy and the performance simultaneously for all NAS benchmarks. In table (II), we record all optimal scaling factors results for each benchmark running class C. These scaling factors give the maximum energy saving percent and the minimum performance degradation percent at the same time from all available scaling factors. As shown in the table (II), when the optimal scaling factor has big value we can gain more energy savings for example as in CG and

Table II: The scaling factors results

Program Name	Optimal Scaling Factor	Energy Saving %	Performance Degradation %	Energy-Perf. Distance
CG	1.56	39.23	14.88	24.35
MG	1.47	34.97	21.70	13.27
EP	1.04	22.14	20.73	1.41
LU	1.38	35.83	22.49	13.34
BT	1.31	29.60	21.28	8.32
SP	1.38	33.48	21.36	12.12
FT	1.47	34.72	19.00	15.72

FT. The opposite happens when the optimal scaling factor is small value as example BT and EP. Our algorithm selects big scaling factor value when the communication and the other slacks times are big and smaller ones in opposite cases. In EP there are no communications inside the iterations. This make our algorithm to selects smaller scaling factor values (inducing smaller energy savings).

C. Results comparison

In this section, we compare our scaling factor selection method with Rauber and Rünger methods [4]. They had two scenarios, the first is to reduce energy to the optimal level without considering the performance as in EQ (8). We refer to this scenario as R_E. The second scenario is similar to the first except setting the slower task to the maximum frequency (when the scale $S = 1$) to keep the performance from degradation as mush as possible. We refer to this scenario as R_{E-P}. While we refer to our algorithm as EPSA. The comparison is made in tables (III,IV,V). These tables show the results of our method and Rauber and Rünger scenarios for all the NAS benchmarks programs for classes A,B and C.

As shown in tables III, IV and V, the (R_{E-P}) method outperforms the (R_E) method in terms of performance and energy reduction. The (R_{E-P}) method also gives better energy savings than our method. However, although our scaling factor is not optimal for energy reduction, the results in these tables prove that our algorithm returns the best scaling factor that satisfy our objective method : the largest distance between energy reduction and performance degradation.

Figure (5) shows the maximum distance between the energy saving percent and the performance degradation percent. Negative values mean that one of the two objectives (energy or performance) have been degraded more than the other. The positive tradeoffs with the highest values lead to maximum energy savings while keeping the performance degradation as low as possible. Our algorithm always gives the highest positive energy to performance tradeoffs while Rauber and Rünger method (R_{E-P}) gives in some time negative tradeoffs such as in BT and EP.

VIII. Conclusion

In this paper, we have presented a new online scaling factor selection method that optimizes simultaneously the energy and performance of a distributed application running on an homogeneous cluster. It uses the computation and communication times measured at the first iteration to predict energy consumption and the performance of the parallel application at every available frequency. Then, it selects the scaling factor that gives the best tradeoff between energy reduction and performance which is the maximum distance between the energy and the inverted performance curves. To evaluate this method, we have applied it to the NAS benchmarks and it was compared to Rauber and Rünger methods while being executed on the simulator SimGrid. The results showed that our method, outperforms Rauber and Rünger methods in terms of energy-performance ratio.

In the near future, we would like to adapt this scaling factor selection method to heterogeneous platforms where each node has different characteristics. In particular, each

Table III: Comparing results for the NAS class A

Method Name	Program Name	Factor Value	Energy Saving %	Performance Degradation %	Energy-Perf. Distance
$EPSA$	CG	1.56	37.02	13.88	23.14
R_{E-P}	CG	2.14	42.77	25.27	17.50
R_E	CG	2.14	42.77	26.46	16.31
$EPSA$	MG	1.47	27.66	16.82	10.84
R_{E-P}	MG	2.14	34.45	31.84	2.61
R_E	MG	2.14	34.48	33.65	0.80
$EPSA$	EP	1.19	25.32	20.79	4.53
R_{E-P}	EP	2.05	41.45	55.67	-14.22
R_E	EP	2.05	42.09	57.59	-15.50
$EPSA$	LU	1.56	39.55	19.38	20.17
R_{E-P}	LU	2.14	45.62	27.00	18.62
R_E	LU	2.14	45.66	33.01	12.65
$EPSA$	BT	1.31	29.60	20.53	9.07
R_{E-P}	BT	2.10	45.53	49.63	-4.10
R_E	BT	2.10	43.93	52.86	-8.93
$EPSA$	SP	1.38	33.51	15.65	17.86
R_{E-P}	SP	2.11	45.62	42.52	3.10
R_E	SP	2.11	45.78	43.09	2.69
$EPSA$	FT	1.25	25.00	10.80	14.20
R_{E-P}	FT	2.10	39.29	34.30	4.99
R_E	FT	2.10	37.56	38.21	-0.65

Table IV: Comparing results for the NAS class B

Method Name	Program Name	Factor Value	Energy Saving %	Performance Degradation %	Energy-Perf. Distance
$EPSA$	CG	1.66	39.23	16.63	22.60
R_{E-P}	CG	2.15	45.34	27.60	17.74
R_E	CG	2.15	45.34	28.88	16.46
$EPSA$	MG	1.47	34.98	18.35	16.63
R_{E-P}	MG	2.14	43.55	36.42	7.13
R_E	MG	2.14	43.56	37.07	6.49
$EPSA$	EP	1.08	20.29	17.15	3.14
R_{E-P}	EP	2.00	42.38	56.88	-14.50
R_E	EP	2.00	39.73	59.94	-20.21
$EPSA$	LU	1.47	38.57	21.34	17.23
R_{E-P}	LU	2.10	43.62	36.51	7.11
R_E	LU	2.10	43.61	38.54	5.07
$EPSA$	BT	1.31	29.59	20.88	8.71
R_{E-P}	BT	2.10	44.53	53.05	-8.52
R_E	BT	2.10	42.93	52.80	-9.87
$EPSA$	SP	1.38	33.44	19.24	14.20
R_{E-P}	SP	2.15	45.69	43.20	2.49
R_E	SP	2.15	45.41	44.47	0.94
$EPSA$	FT	1.38	34.40	14.57	19.83
R_{E-P}	FT	2.13	42.98	37.35	5.63
R_E	FT	2.13	43.04	37.90	5.14

Table V: Comparing results for the NAS class C

Method Name	Program Name	Factor Value	Energy Saving %	Performance Degradation %	Energy-Perf. Distance
$EPSA$	CG	1.56	39.23	14.88	24.35
R_{E-P}	CG	2.15	45.36	25.89	19.47
R_E	CG	2.15	45.36	26.70	18.66
$EPSA$	MG	1.47	34.97	21.69	13.27
R_{E-P}	MG	2.15	43.65	40.45	3.20
R_E	MG	2.15	43.64	41.38	2.26
$EPSA$	EP	1.04	22.14	20.73	1.41
R_{E-P}	EP	1.92	39.40	56.33	-16.93
R_E	EP	1.92	38.10	56.35	-18.25
$EPSA$	LU	1.38	35.83	22.49	13.34
R_{E-P}	LU	2.15	44.97	41.00	3.97
R_E	LU	2.15	44.97	41.80	3.17
$EPSA$	BT	1.31	29.60	21.28	8.32
R_{E-P}	BT	2.13	45.60	49.84	-4.24
R_E	BT	2.13	44.90	55.16	-10.26
$EPSA$	SP	1.38	33.48	21.35	12.12
R_{E-P}	SP	2.10	45.69	43.60	2.09
R_E	SP	2.10	45.75	44.10	1.65
$EPSA$	FT	1.47	34.72	19.00	15.72
R_{E-P}	FT	2.04	39.40	37.10	2.30
R_E	FT	2.04	39.35	37.70	1.65

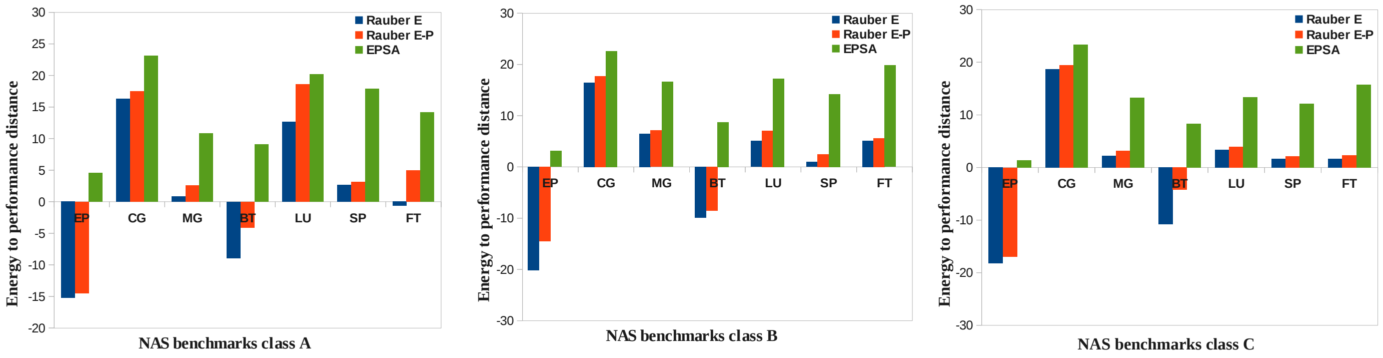


Figure 5: Comparing our method to Rauber and Rünger methods

CPU has different available frequencies, energy consumption and performance. It would be also interesting to develop a new energy model for asynchronous parallel iterative methods where the number of iterations is not known in advance and depends on the global convergence of the iterative system.

Acknowledgment

As a PhD student, M. Ahmed Fanfakh, would like to thank the University of Babylon (Iraq) for supporting his work.

References

[1] "TOP500 Supercomputers Sites." [Online]. Available: http://www.top500.org

[2] NASA Advanced Supercomputing Division, "NAS parallel benchmarks," Mar. 2012. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[3] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a generic framework for large-scale distributed experiments," in *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, ser. UKSIM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 126–131.

[4] T. Rauber and G. Rünger, "Analytical modeling and simulation of the energy consumption of independent tasks," in *Proceedings of the Winter Simulation Conference*, ser. WSC '12. Winter Simulation Conference, 2012, pp. 245:1–245:13.

[5] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, "Profile-based dynamic voltage scheduling using program checkpoints," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 168–175.

[6] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "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 (IPDPS'05) - Papers - Volume 01*, ser. IPDPS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 4.1–.

[7] F. Xie, M. Martonosi, and S. Malik, "Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation," in *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, Aug 2005, pp. 287–292.

[8] B. Rountree, D. Lowenthal, S. Funk, V. W. Freeh, B. De Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, November 2007, pp. 1–9.

[9] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: Adaptive DVFS and thread packing under power caps," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 175–185.

[10] R. Cochran, C. Hankendi, A. Coskun, and S. Reda, "Identifying the optimal energy-efficient operating points of parallel workloads," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '11. Piscataway, NJ, USA: IEEE Press, 2011, pp. 608–615.

[11] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster," in *Cluster Computing, 2006 IEEE International Conference on*, Sept 2006, pp. 1–10.

[12] J. Peraza, A. Tiwari, M. Laurenzano, C. L., and Snavely, "PMaC's green queue: a framework for selecting energy optimal DVFS configurations in large scale MPI applications," *Concurrency Computat.: Pract. Exper.DOI: 10.1002/cpe*, pp. 1–20, 2012.

[13] Y.-L. Chou, S. Liu, E.-Y. Chung, and J.-L. Gaudiot, "An energy and performance efficient DVFS scheme for irregular parallel divide-and-conquer algorithms on the Intel SCC," *IEEE Computer Architecture Letters*, vol. 99, no. RapidPosts, p. 1, 2013.

[14] G. Dhiman and T. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," in *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, Aug 2007, pp. 207–212.

[15] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," in *ISQED*, 2012, pp. 747–754.

[16] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[17] K. Malkowski, "Co-adapting scientific applications and architectures toward energy-efficient high performance computing," Ph.D. dissertation, The Pennsylvania State University, USA, 2009.

[18] J. Zhuo and C. Chakrabarti, "Energy-efficient dynamic task scheduling algorithms for dvs systems," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 2, pp. 17:1–17:25, Jan. 2008.

[19] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *J. Parallel Distrib. Comput.*, vol. 71, no. 8, pp. 1154–1164, Aug. 2011.

[20] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, Dec. 2003.

[21] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10)*, Vancouver, Canada, October 2010.

[22] G. Dhiman and T. Rosing, "System-level power management using online learning," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 5, pp. 676–689, May 2009.