2014 IEEE International Symposium on Parallel and Distributed Processing with Applications

Dynamic Frequency Scaling for Energy Consumption Reduction in Synchronous Distributed Applications

Jean-Claude Charr, Raphaël Couturier, Ahmed Fanfakh and Arnaud Giersch
FEMTO-ST Institute
University of Franche-Comté
IUT de Belfort-Montbéliard, 19 avenue du Maréchal Juin, BP 527, 90016 Belfort cedex, France
Email: {jean-claude.charr,raphael.couturier,ahmed.fanfakh_badri_muslim,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. 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 if a low CPU frequency is selected. Therefore, the chosen scaling factor must give the best possible trade-off 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 million of cores and delivers more than 33 Tflop/s while consuming 17,808 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 has become an important topic in the high performance computing field. To tackle this problem, many researchers use 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 the 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 the 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 overhead. The proposed algorithm takes into account the communication times of the MPI program to choose the scaling factor. This algorithm has the 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 the NAS parallel benchmarks (NPB v3.3) [2]. Our experiments are executed using the simulator SimGrid/SMPI v3.10 [3] over a 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 trade-off.

This paper is organized as follows: Section II presents some related works from other authors. Section III presents an energy model for homogeneous platforms. Section IV describes how the performance of MPI programs can be predicted. Section V presents the energy-performance objective function that maximizes the reduction of energy consumption while minimizing the degradation of the program's performance. Section VI details the proposed energy-performance algorithm. Section VII verifies the accuracy of the performance prediction model and presents the results of the proposed algorithm. It also shows the comparison results between our method and other existing methods. Finally, we conclude in Section VIII with a summary and some future works.

II. Related works

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

978-1-4799-4293-0/14 $31.00 © 2014 IEEE
DOI 10.1109/ISPA.2014.38

IEEE computer society

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 the 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 multi-task 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 has already finished its computation, has 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. The whole program or, a part of it, is usually executed over all the available frequency gears and the execution time and the energy consumed with each frequency gear are measured. Then a heuristic or an exact method uses the retrieved information to compute the values of the scaling factor for the processors. In [7] , Rountree et al. use a linear programming algorithm, while in [8], Cochran et al. use a multi-logistic regression algorithm for the same goal. The main drawback of these methods is that they all require executing the whole program or, a part of it, 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, a lot of information is 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. Peraza, Yu-Liang et al. [9], [10] used varied heuristics to select the appropriate scaling factor values to eliminate the slack times during runtime. However, as seen in [11], machine learning method takes a lot of time to converge when the number of available gears is big. To reduce the impact of slack times, in [12], 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 can predict the consumed energy for every frequency gear after measuring the consumed energy. They maintain the performance as mush as possible by setting the highest frequency gear to the slowest task.

The primary contribution of our paper is to present a new online scaling factor selection method which has the following characteristics:
1) It is based on Rauber and Rünger analytical model to predict the energy consumption of the application with different frequency gears. 2) It selects the frequency scaling factor for simultaneously optimizing energy reduction and maintaining performance. 3) It is well adapted to distributed architectures because it takes into account the communication time. 4) It is well adapted to distributed applications with imbalanced tasks. 5) It has a very small overhead when compared to other methods (e.g., [11]) and does not require profiling or training as in [8].

III. Energy model for a homogeneous platform

Many researchers [13], [4], [14], [15] divide the power consumed by a processor into 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 (1).

$$P_{dyn} = \alpha \cdot C_L \cdot V^2 \cdot f \tag{1}$$

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

$$P_{static} = V \cdot N_{trans} \cdot K_{design} \cdot I_{leak} \tag{2}$$

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 given program can be computed as:

$$E_{ind} = P_{dyn} \cdot T_{comp} + P_{static} \cdot T \tag{3}$$

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

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 [16]. 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 (4).

$$S = \frac{F_{max}}{F_{new}} \tag{4}$$

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. 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 [17]. 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 and Rünger [4], can be written as a function of the scaling factor S, as in EQ (5).

$$E = P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot S_1 \cdot N \quad (5)$$

where N is the number of parallel nodes, T_i for $i = 1, \ldots, N$ are the execution times of the sorted tasks. Therefore, T_1 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 S_i are computed as in EQ (6).

$$S_i = S \cdot \frac{T_1}{T_i} = \frac{F_{max}}{F_{new}} \cdot \frac{T_1}{T_i} \quad (6)$$

In this paper we use Rauber and Rünger's energy model, EQ (5), because it can be applied to homogeneous clusters if the communication time is taken in consideration. Moreover, we compare our algorithm with Rauber and Rünger's scaling factor selection method which uses the same energy model. In their method, the optimal scaling factor is computed by minimizing the derivation of EQ (5) which produces EQ (7).

$$S_{opt} = \sqrt[3]{\frac{2}{N} \cdot \frac{P_{dyn}}{P_{static}} \cdot \left(1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^3} \right)} \quad (7)$$

IV. Performance Evaluation of MPI Programs

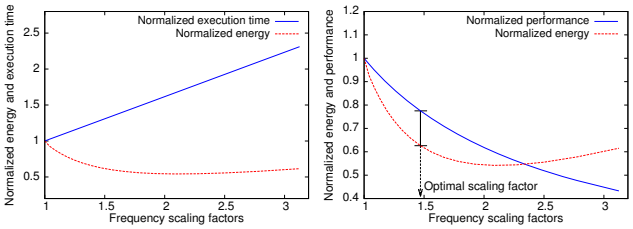
The execution time of a parallel synchronous iterative application is equal to the execution time of the slowest task. If there is no communication and the application is not data bounded, the execution time of a parallel program is linearly proportional to the operational frequency and any DVFS operation 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 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 slowest 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 (8).

$$T_{New} = T_{Max\ Comp\ Old} \cdot S + T_{Max\ Comm\ Old} \quad (8)$$

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 and Energy Reduction Trade-off

This section presents our method for choosing the optimal scaling factor that gives the best tradeoff between energy reduction and performance. This method takes into account the execution times for both computation and communication to compute the scaling factor. Since the energy consumption and the performance are not measured using the same metric, a normalized value of both measurements can be used to compare them. The normalized energy is the ratio between the consumed energy with scaled frequency and the consumed energy without scaled frequency:



(a) Real relation. (b) Converted relation.

Figure 1: The energy and performance relation

$$E_{Norm} = \frac{E_{Reduced}}{E_{Original}} = \frac{P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot S_1 \cdot N}{P_{dyn} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot N} \quad (9)$$

In the same way, the normalized execution time of a program is computed as follows:

$$T_{Norm} = \frac{T_{New}}{T_{Old}} = \frac{T_{Max\ Comp\ Old} \cdot S + T_{Max\ Comm\ Old}}{T_{Max\ Comp\ Old} + T_{Max\ Comm\ Old}} \quad (10)$$

The relation between the execution time and the consumed energy of a program is nonlinear and complex. In consequences, the relation between the consumed energy and the scaling factor is also nonlinear, for more details refer to [6]. Therefore, the resulting normalized energy consumption curve and execution time curve, for different scaling factors, do not have the same direction see Figure 1(a). To tackle this problem and optimize both terms, we inverse the equation of the normalized execution time as follows:

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

Then, we can model our objective function as finding the maximum distance between the energy curve EQ (9) and the inverse of the execution time (performance) curve EQ (11) over all available scaling factors. This represents the minimum energy consumption with minimum execution time (better performance) at the same time, see Figure 1(b). Then our objective function has the following form:

$$Max\ Dist = \max_{j=1,2,\ldots,F} (\overbrace{P_{Norm}(S_j)}^{\text{Maximize}} - \overbrace{E_{Norm}(S_j)}^{\text{Minimize}}) \quad (12)$$

where F is the number of available frequencies. Then we can select the optimal scaling factor that satisfies EQ (12). 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 [14], [4], [11].

Require:

P_{static}	static power value
P_{dyn}	dynamic power value
P_{states}	number of available frequencies
F_{max}	maximum frequency
F_{diff}	difference between two successive freq.

Ensure: S_{opt} is the optimal scaling factor

1: $S_{opt} \leftarrow 1$
2: $Dist \leftarrow 0$
3: $F_{new} \leftarrow F_{max}$
4: **for** $j = 2$ to P_{states} **do**
5: $\quad F_{new} \leftarrow F_{new} - F_{diff}$
6: $\quad S \leftarrow F_{max}/F_{new}$
7: $\quad S_i \leftarrow S \cdot \frac{T_1}{T_i} = \frac{F_{max}}{F_{new}} \cdot \frac{T_1}{T_i}$ for $i = 1, \ldots, N$
8: $\quad E_{Norm} \leftarrow \frac{P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2}\right) + P_{static} \cdot T_1 \cdot S_1 \cdot N}{P_{dyn} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2}\right) + P_{static} \cdot T_1 \cdot N}$
9: $\quad P_{Norm} \leftarrow T_{Old}/T_{New}$
10: $\quad$ **if** $(P_{Norm} - E_{Norm} > Dist)$ **then**
11: $\quad\quad S_{opt} \leftarrow S$
12: $\quad\quad Dist \leftarrow P_{Norm} - E_{Norm}$
13: $\quad$ **end if**
14: **end for**
15: Return S_{opt}

Figure 2: Scaling factor selection algorithm

VI. Optimal scaling factor for performance and energy

Algorithm on Figure 2 computes 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. In our experiments over a homogeneous cluster described in Section VII, this algorithm has a small execution time. It takes 1.52 μs on average for 4 nodes and 6.65 μs on average for 32 nodes. The algorithm complexity is $O(F \cdot N)$, 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 on Figure 3 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 (4) in EQ (6), we can calculate the new frequency F_i as follows:

$$F_i = \frac{F_{max} \cdot T_i}{S_{opt} \cdot T_{max}} \tag{13}$$

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 (13) adapts the frequency of the CPU to the nodes' workloads to maintain the performance of the program.

1: **for** $k = 1$ to *some iterations* **do**
2: $\quad$ Computations section.
3: $\quad$ Communications section.
4: $\quad$ **if** $(k = 1)$ **then**
5: $\quad\quad$ Gather all times of computation and communication from each node.
6: $\quad\quad$ Call algorithm from Figure 2 with these times.
7: $\quad\quad$ Compute the new frequency from the returned optimal scaling factor.
8: $\quad\quad$ Set the new frequency to the CPU.
9: $\quad$ **end if**
10: **end for**

Figure 3: DVFS algorithm

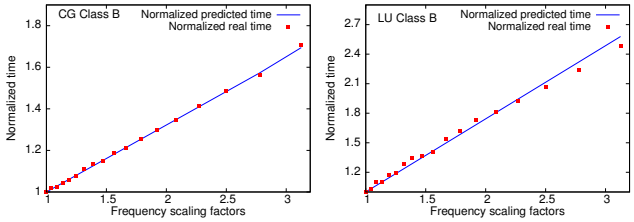


Figure 4: Comparing predicted to real execution times

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. 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 nodes are connected via an ethernet network with 1Gbit/s bandwidth.

A. Execution time prediction verification

In this section we evaluate the precision of our execution time prediction method based on EQ (8) by applying it to the NAS benchmarks. The NAS programs are executed with the class B option to compare 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 (4). In our cluster there are 18 available frequency states for each processor. This leads 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 4 presents plots of the real execution times and the simulated ones. The maximum normalized error between these two execution times varies between 0.0073 to 0.031 dependent on the executed benchmark. The smallest prediction error was for CG and the worst one was 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

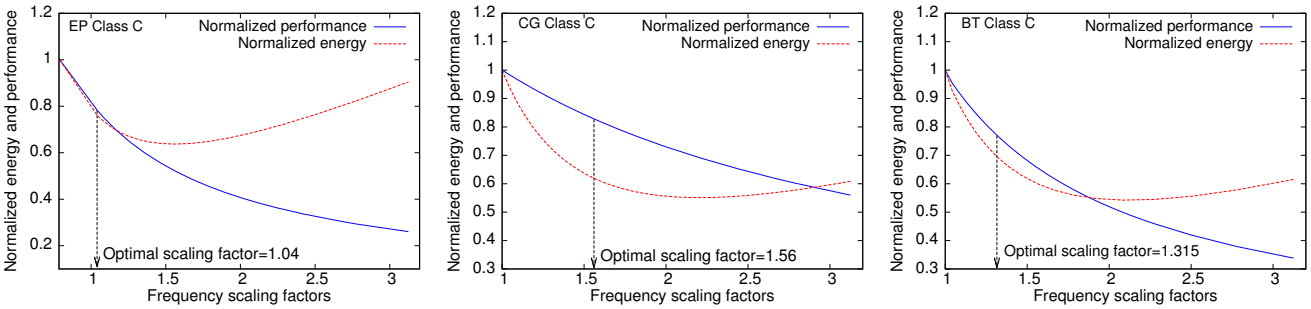


Figure 5: Optimal scaling factors for the predicted energy and performance of NAS benchmarks

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 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 (5), we measure the energy consumption for all the NAS MPI programs while assuming that the dynamic power 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 from Figure 5. These plots illustrate that there are different distances between the normalized energy and the normalized inverted execution time curves, because there are different communication features for each benchmark. When there are little or no communications, the inverted execution time 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 I. All discovered frequency scaling factors optimize both the energy and the execution time simultaneously for all NAS benchmarks. In Table I, we record all optimal scaling factors results for each benchmark running class C. These scaling factors give the maximum energy saving percentage and the minimum performance degradation percentage at the same time from all available scaling factors.

As shown in Table I, when the optimal scaling factor has a big value we can gain more energy savings as in CG and FT benchmarks. The opposite happens when the optimal scaling factor has a small value as in BT and EP benchmarks. Our algorithm selects a 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 communication inside the iterations. This leads our algorithm to select 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 execution time as in EQ (7). 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 (Energy to Performance Scaling Algorithm). The comparison is made in Table I. This table shows the results of our method and Rauber and Rünger scenarios for all the NAS benchmarks programs for class C.

Table I: 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

As shown in Table I, 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 this table prove that our algorithm returns the best scaling factor that satisfy our objective method: the largest distance between energy reduction and performance degradation. Figure 6 illustrates even better the distance between the energy reduction and performance degradation. The negative values mean that one of the two objectives (energy or performance) have been degraded more than the other. The positive trade-offs 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 trade-

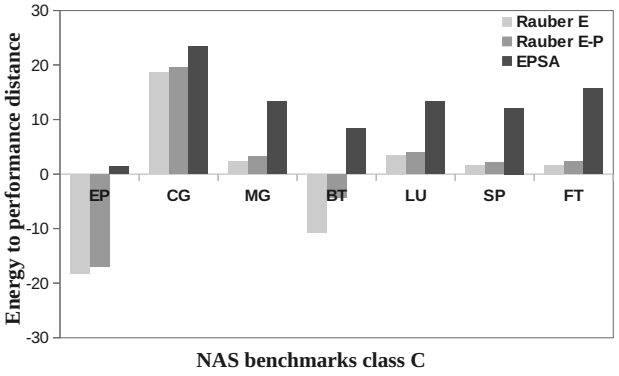


Figure 6: Comparing our method to Rauber and Rünger's methods

offs while Rauber and Rünger's method, (R_{E-P}), gives sometimes negative trade-offs 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 a homogeneous cluster. It uses the computation and communication times measured at the first iteration to predict energy consumption and the execution time of the parallel application at every available frequency. Then, it selects the scaling factor that gives the best trade-off between energy reduction and performance which is the maximum distance between the energy and the inverted execution time 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's 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 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

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] "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. 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, 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. IEEE Computer Society, 2005, pp. 4.1–.

[7] 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.

[8] 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. NJ, USA: IEEE Press, 2011, pp. 608–615.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.