Exploration of Load Balancing Thresholds to Save Energy on Iterative Applications

Edson L. Padoin[1], Laércio L. Pilla[2], Márcio Castro[2], Philippe O. A. Navaux[3] and Jean-François Méhaut[4]

[1] Department of Exact Sciences and Engineering
Regional University of Northwest of Rio Grande do Sul (UNIJUI) – Ijuí, RS – Brazil
[2] Department of Informatics and Statistics
Federal University of Santa Catarina (UFSC) – Florianpolis, SC – Brazil
[3] Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS) – Porto Alegre, RS – Brazil
[4] Laboratoire d'Informatique de Grenoble (LIG)
Grenoble University – Grenoble – France
padoin@unijui.edu.br,laercio.pilla@ufsc.br,marcio.castro@ufsc.br,
navaux@inf.ufrgs.br,jean-francois.mehaut@imag.fr

Abstract. The power consumption of High Performance Computing systems is an increasing concern as large-scale systems grow in size and, consequently, consume more energy. In response to this challenge, we proposed two variants of a new energy-aware load balancer that aim at reducing the energy consumption of parallel platforms running imbalanced scientific applications without degrading their performance. Our research combines Dynamic Load Balancing with Dynamic Voltage and Frequency Scaling techniques in order to reduce the clock frequency of underloaded computing cores which experience some residual imbalance even after tasks are remapped. This work presents a trade-off evaluation between runtime, power demand and total energy consumption when applying these two energy-aware load balancer variants on real-world applications. In this way, we can define which is the best threshold value for each application under the total energy consumption, total execution time or the average power demand focus.

1 Introduction

Several load balancers are able to reduce the total energy consumption of an application by reducing its total execution time (as energy=time$\times$power). Load balancers can improve the performance of imbalanced iterative applications by making a better load distribution among the available processors. However, they can take suboptimal decisions that result in some load imbalance still remaining after task migrations. This can happen due to characteristics of the application that prevent a perfectly balanced mapping to be achieved, and due to limitations of load balancing heuristics, as the problem that they are trying to solve is NP-Hard [15].

Our proposed algorithms, FG-EnergyLB and CG-EnergyLB [17] try to reduce the total energy consumption by exploiting residual imbalance left by load balancing algorithm. The first one, called *Fine-Grained Energy Load Balancer (*FG-EnergyLB*)*, is suitable for platforms composed of few tens of cores that allow per-core Dynamic Voltage and Frequency Scaling (DVFS). The second one, called *Coarse-Grained Energy Load Balancer (*CG-EnergyLB*)* is suitable for current HPC platforms composed of several multi-core processors that feature per-chip DVFS. They identify the possibility of reducing the processors clock to achieve better gains better than other load balancing algorithms that they employ. In this form, energy improvements are achieved due to the reduction of average power during the runtime and also by reducing the application execution time by reducing the number of tasks migrated. The main idea of the EnergyLB is to exploit the existence of residual imbalances on iterative applications to adjust the clock frequency of underloaded cores/processors through DVFS.

Nevertheless, the definition of the interval between calls to the load balancer is decisive to reduce the load balancing overhead. If the load balancer is invoked in long time periods, the load imbalance may increase too much and result in loss of performance, which consequently increases the total energy consumption. On the other hand, if the strategy is performed very frequently, it also may incur in a reduction of performance, since the load balancing overhead may exceed its benefits. In this context, aiming to decrease the load balancing overhead, recent strategies have adopted a threshold value to determine if load balancing or DVFS must be performed.

In this context, in this paper we focus on a trade-off evaluation between runtime, power demand, and total energy consumption when using different threshold values in the two variants of our energy-aware load balancer (EnergyLB) on two imbalanced real-world applications. Our results show that FG-EnergyLB can achieve energy savings of up to 17.1% with an average of 16.3%, and CG-EnergyLB of up to 31% with an average of 23% through the reduction of the average power demand. However, we observed that the total execution time of the applications may be reduced or increased according to threshold value chosen.

The remaining sections of this paper are organized as follows. Section 2 discusses related works on DVFS and energy-aware load balancing. Then, Section 3 presents the evaluation methodology and the applications used to evaluate the efficiency of our energy-aware load balancer. Our experimental results are discussed in Section 4. Finally, Section 5 concludes this paper.

2 Related Work on Energy Consumption

Different techniques have been proposed to reduce the runtime and power demand and thus improve the energy efficiency of platforms while running parallel applications. Among them, we highlight in this section DVFS and load balancing strategies.

Dynamic voltage and frequency scaling (DVFS). Recent studies demonstrate that an idle host may consume more than half of its peak power [7]. Because of that, DVFS has been used in different contexts as a means to save energy. Gerards *et al.* [4] analyze the use of global DVFS in the context of multi-core processors. They proposed a theoretical method to transform the problem of finding an optimal clock frequency on global DVFS systems to a single core problem by using the amount of parallelism of applications. Their main goal is to minimize the energy consumption of nontrivial real-time applications. Spiliopoulos *et al.* [19] extended the gem5 simulator to support full-system DVFS modeling. This extended version is then used to study the behavior of different DVFS governors (interactive, on-demand and performance). They concluded that the interactive governor is faster than on-demand to adapt to the workload changes and thus achieves better performance at about the same energy consumption. Kin *et al.* [14] proposed a realistic DVFS performance prediction method and a practical DVFS control policy (eDVFS) that aims to minimize total energy consumption in multi-core processors. Their experimental results show that eDVFS can save a substantial amount of energy compared with Linux on-demand. Isci *et al.* [9] proposed to fine-tune the processor's clock frequency by using workload characteristics to maintain a chip-level power below a specified budget without degrading the performance significantly. The proposed approach can come within 1% of the performance of an ideal oracle, while meeting a given chip-level power budget.

Energy-aware load balancing. Load balancing is a challenging problem and has been studied extensively in the past to improve the performance of parallel applications [11, 21]. However, few works have made some efforts to further improve the energy consumption. Aupy [1] proposed energy-aware scheduling models to schedule tasks under reliability and makespan constraints. They designed and evaluated them using simulations with different heuristics based on the failure probability, the task weights, and the processor speeds. These heuristics aim at minimizing the energy consumption while enforcing reliability and deadline constraints. Sarood *et al.* [18] proposed a load balancing strategy that limits the processors' temperatures to reduce the energy spent in cooling and to prevent hot spots. Their results achieved energy savings of up to 63%, with a timing penalty from 2% to 23%. Goel *et al.* [5] proposed a model that uses CPU performance counters and CPU temperature to generate accurate per-core power estimates in real-time. They showed that the model can be used to guide scheduling decisions in power-aware resource managers. Hartog *et al.* [6] studied the relationship between CPU temperature and energy consumption in clusters and provided a method of estimating the power consumption of the system. This method was then used to implement a MapReduce framework that can evaluate the current status of each node and dynamically react to estimated power usage without having to rely on readings from expensive power monitoring hardware affixed to each node in the cluster.

As opposed to these works, our energy-aware approach performs load balancing along with DVFS to improve the performance and to reduce the energy consumption by exploiting residual imbalances of parallel applications [17]. In addition, we also reduce the cost of task migrations, since we only migrate tasks between processors when necessary. The performance, power demand and total energy consumption of our energy-aware load balancers are here analyzed on a set of real-world application running on top of a real platform without the need of simulations.

3 Evaluation Methodology

This section describes the methodology used in our trade-off study. We first present the execution environment, followed by the applications used in our experiments.

3.1 Experimental Environment

The experiments were conducted on an Altix UV 2000 platform designed by SGI. The platform is composed of 24 NUMA nodes. Each node has an Intel Xeon E5-4640 Sandy Bridge-EP x86-64 processor with 8 physical cores running at 2.40 GHz. There are 14 clock frequency levels available in this processor, allowing us to vary the clock frequency of the processor from 1.2 GHz up to 2.4 GHz.

Each core of the Intel Xeon E5-4640 has 32 KB instruction and 32 KB data L1 caches and 256 KB of L2 cache. All the 8 cores share a 20 MB L3 cache. Each node has 32 GB of DDR3 memory, which is shared with other nodes in a cc-NUMA fashion through SGI's proprietary NUMAlink6. Overall, this platform has 192 physical cores and 768 GB DDR3 memory.

The platform runs an unmodified SUSE Linux Enterprise Server operating system with kernel 3.0.101-0.29 installed. All applications as well as the Charm++ programming model were compiled with GCC 4.8.2. The Charm++ version used in our experiments was linux64-6.5.1. The results presented in Section 4 are the average of at least 10 runs. The relative error was less than 5% using a 95% statistical confidence by Student's t-distribution.

3.2 Applications

To evaluate the trade-off between run time, power demand and total energy consumption of our proposed variants of EnergyLB, we selected different real-world applications. They were chosen due to their varied range of communication patterns and workload characteristics. The description of the applications is given below:

– ***Ondes3D*** is a seismic wave propagation simulator employed to estimate the damage in future earthquake scenarios [3]. In *Ondes3D*, seismic waves are modeled as a set of elastodynamics equations. These equations are then solved by applying a finite difference method. In our experiments, we used a

version recently adapted to Adaptive MPI [8] [10] that profits from Charm++'s load balancing framework [20]. In this version, the application is overdecomposed into multiple virtual MPI processes per core. *Ondes3D* presents load irregularity due to the boundary conditions producing additional work, and load dynamicity from the simulation of waves spreading through space;
- ***Lulesh*** simulates a variety of science and engineering problems requiring hydrodynamics modeling, which describes the motion of materials relative to each other when subject to forces. The Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH) application was originally developed as one of the five challenge problems in the DARPA Ubiquitous High Performance Computing (UHPC) program. *Lulesh* solves one octant of the spherical Sedov problem using Lagrange hydrodynamics [13] [12] [2].

Input parameters Table 1 summarizes the characteristics of the applications and parameters used in our experiments. Different load balancing frequencies have been chosen for different applications in order to strike a balance between the benefits of remapping tasks and the overheads of moving tasks and computing a new task mapping. Deciding the optimal moment to call a load balancer is a challenging problem [16] and is out of the scope of this paper.

Table 1. Summary of the input parameters of applications.

Application	Tasks	Iterations	LB Frequency
Ondes3D	128	500	20
Lulesh	729	1000	50

3.3 Load Balancers

Charm++ provides a set of load balancing algorithms that can be used to migrate tasks among processors and to reduce the load imbalance. Thus, to analyze which is the best threshold value for each application under the total energy consumption, total execution time or the average power demand focus, we have selected the GreedyLB load balancer available on Charm++ platform.

4 Experimental Results

This section presents a trade-off between run time, power demand and total energy consumption achieved by our energy-aware load balancer.

The Intel Xeon E5-4640 processors, available on our experimental platform, there are 14 clock frequency levels available, which allow us to vary the clock frequency of the processor from 1.2 GHz up to 2.4 GHz. So, we vary the threshold (*thrld*) parameter of the algorithm from 0 up to 5 and execute the applications, in order to make a trade-off. In the following sections we first evaluate the results achieved with FG-EnergyLB on applications. Then, we perform a similar evaluation using the CG-EnergyLB.

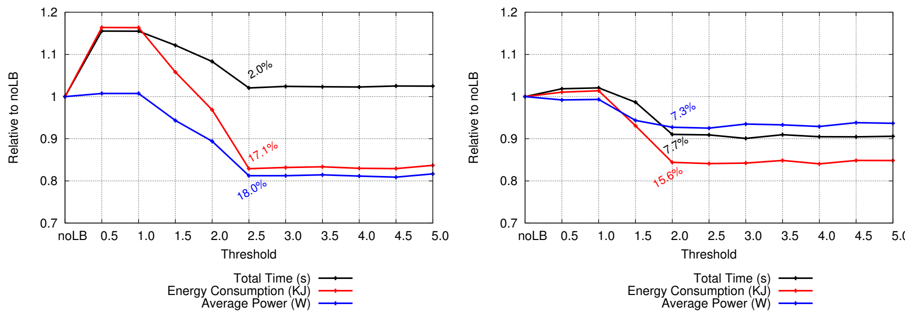
4.1 Fine-Grained EnergyLB Evaluation

Aiming to reduce the effects of load imbalance and load balancing overhead to save energy, this section provides a trade-off between run time, power demand and total energy consumption when used FG-EnergyLB over real applications with different threshold values. The application run time depends on several issues, among them, the number of parallel tasks and their load, the duration of each timestep, and the selected load balancing strategy. The impact of load balancing is directly related to the load balancing frequency once load balancing overhead can overcome the gains achieved with load balancing.

In this way, to FG-EnergyLB, in each call of the load balancer, the algorithm verifies if the weighted load of each processor exceeds or not the *threshold*, makes decisions to adjust the frequencies (determining so that the frequency will be decreased or increased) or invokes other load balancer to migrate tasks. However, the load balancer generates an overhead and when this cost exceeds its benefits, the total execution time is increased, i.e., calling load balancing strategies incurs timing penalties to applications.

Our proposed load balancers take three input parameters in their execution. The first one, is the load balancer that is used to migrate tasks when the imbalance is high. The second one, is the maximum frequency available by processors that can be set to a core, and the last one, is a threshold value, used to decide whether call the load balancer or perform DVFS strategy.

Running the applications with FG-EnergyLB configured with different threshold values, we obtain different amounts of DVFS performed or load balancers called, what determines different frequency settings of cores or migration tasks. In this way, we can analyze which is the best threshold value for each application under the total energy consumption, total execution time or the average power demand focus as shown in the Figure 1.



(a) *Ondes3D*. (b) *Lulesh*.

Fig. 1. FG-EnergyLB comparison with different threshold value on Real Applications

– *Ondes3D* Application

Experiments with *Ondes3D* were performed using 128 tasks, which run for 500 iterations. Total energy spent to run this application without a load balancer is 550.4 kJoules and its total execution time is 645.8 seconds. In this way, during the execution, the average power demand is 35.5 W. These values are taken as reference in the analysis and represent the noLB value in the Figure 2(a).

In the tests with FG-EnergyLB, the load balancer is called at every 20 iterations, resulting in a total of 24 calls. Using threshold values equal to 0.5 and 1.0, FG-EnergyLB does not perform DVFS, calling GreedyLB at every opportunity to migrate tasks. In this context, the average power remains constant around of 35.5 W. However, performing migrations in this application is very costly, which incurs in an increase of 15.5% in run time and total energy consumption. This increase is the result of the overhead of migrations undertaken by GreedyLB.

Using a threshold equal to 1.5, FG-EnergyLB adjusts the clock frequency through DVFS 18× and only 6× calls the other load balancer to migrate tasks. In this way, it is able to reduce the average power in 5.7%. Reducing the number of migrations, the run time suffers a small reduction to 724.38 seconds, which is still 12.2% longer than with no load balancer, and spends 5.82% more energy.

With the threshold value equal to 2.0, gains in both execution time, power demand and, consequently energy consumption, are achieved. DVFS was performed 18× during the execution, which reduced the average power in 10.6%, but the run time was still 8.32% larger than the baseline. Nevertheless, using this threshold, the total energy consumption is reduced in only 3.15%.

For thresholds from 1.0 up to 2.5, the increase of threshold value also increases the number of calls of DVFS. For these values, the run time has a reduction near to linear. The total execution times is reduced from 15% to 2% larger than the baseline. Similarly, for this threshold range, the average power demand of the parallel platform is reduced by up to 19%. In this way, both run time and power demand reductions contribute to reduce the total energy spent.

The greater energy saving for this application is achieved using the threshold value equal or greater than 2.5. Using these values FG-EnergyLB is able to reduce in up to 17.1% the total energy consumption in relation to baseline noLB. These gains are achieved through a reduction in the overhead, which is only 2%, and the average power is reduced in 18%, once that in all calls DVFS was performed, which resulted in a greater amount of energy saving, as shown in Figure 2(a).

– *Lulesh* Application

Lulesh was executed with 1000 iterations in each one of its 729 processes mapped in 24 cores. This application spent 100.6 kJoules of energy and takes 84.7 seconds when executed without a load balancer. In this execution the average power demand is 35.1 W. Similar to *Ondes3D*, these values are taken as baseline (noLB) in Figure 2(b)) to examine the threshold variation of the *Lulesh* application.

Load balancing call is configured with a frequency of 50 iterations so, in this test, our load balancer are called 19 times during the execution. When thresholds equal to 0.5 and 1.0 were used, FG-EnergyLB did not perform any time adjustment in clock frequency. So, using these thresholds the average power is not changed. In addition, in these tests the load balancing overhead increases the run time and consequently, the total energy consumption in up to 2%.

A greater amount of DVFS is performed when the value of threshold is increased. For a threshold equal to 1.5, FG-EnergyLB calls DVFS 17×. Thus, it reduces the run time in up to 1.33%, which also contributes for the reduction of the total energy spent. FG-EnergyLB achieves a reduction of up to 5.6% in average power demand. In this way, reducing both, the power demand and run time, the total of energy spent is reduced in up to 6.91%.

The energy saving further increases when using thresholds equal to 2.0 or greater. For these values, the run time reduction is greater than the reduction of the average power demand. In every call of the load balancer, DVFS was performed, which resulted in reductions of 7.3% in average power demand and an average performance improvement of 7.74% compared to noLB. In this way, FG-EnergyLB is able to save energy by up to 15.6% to *Lulesh*.

For this application, the threshold variation from 1.0 up to 2.0 presented the more significant reduction in execution time. When these values were used, the run time was reduced in up to 11%, while the average power demand is reduced in up to 6%. Similarly to *Ondes3D*, both run time and power demand reductions contribute to the reduction of the total energy consumption.

In the tests with threshold values greater than 2.0, a greater amount of DVFS is performed, resulting in lower average power demands. However, such reductions cause an equivalent increase in the total execution time, thus maintaining the energy consumption constant.

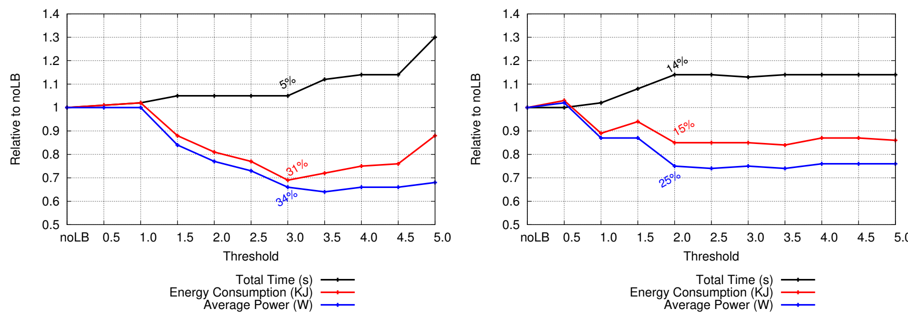
In the execution with threshold equal to 5.0, FG-EnergyLB adjusts the frequency 8×, which resulted in a reduction of the power in 23.36% and reduction of energy in 21.50%. However, the run time exceeds the baseline by 2.42%.

4.2 Coarse-Grained EnergyLB Evaluation

These scientific applications present a dynamic behavior, as the load of theirs tasks change through the iterations, which provides a more challenging scenario for energy aware load balancing. Since all the 192 cores will be used, different parameters are used in the evaluation of the CG-EnergyLB, as shown in Table 2.

Table 2. Summary of the input parameters of real applications.

Application	Tasks	Iterations	LB Frequency
Ondes3D	1024	500	20
Lulesh	5832	1000	50



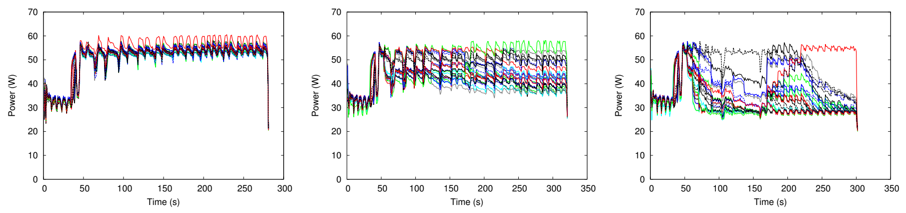
(a) *Ondes3D*. (b) *Lulesh*.

Fig. 2. CG-EnergyLB comparison with different threshold value on Real Applications

– *Ondes3D* Application

Experiments with CG-EnergyLB over *Ondes3D* were performed using 1024 tasks mapped on 192 cores, which run 500 iterations each. Total energy spent to run this application without load balancer is 263.1 kJoules and its total execution time is 200.41 seconds, which represent an average power demand of 54.71 W. These values are taken as reference (noLB in Figure 2(a)) in our analysis.

Figure 3(a) depicts the instantaneous power of the execution when CG-EnergyLB uses a threshold value equal to 1.0. In this execution any times DVFS is performed, in all the 24 load balancer calls tasks were migrated by RefineLB. In this way, during execution the power of all processors is always high, resulting in an average power of 54.7 W.



(a) Threshold = 1.0. (b) Threshold = 2.5. (c) Threshold = 5.0.

Fig. 3. Power evaluation to different threshold value on *Ondes3D*

Using threshold equal to 2.5 the processors power is differently reduced as shown in the Figure 3(b). For this threshold, 17 times DVFS is performed and only 7 calls migrate tasks by RefineLB. This form, the power is reduced in great majority of the processors, which result in a total reduction of 16.08%, leaving the average power in 43.4 W.

A different amount of energy is saved when using threshold equal to 5.0 (Figure 3(c)). In this execution in all call (24) adjusts in clock were performed, leaving only one processor using its maximum power. For this threshold value, the power demand follows the increase of application needs, once the increases from the second 160 and reduces again from the second 212. This form, in this test the average power is reduced in 32.39%, resulting in an average of 35.0 W.

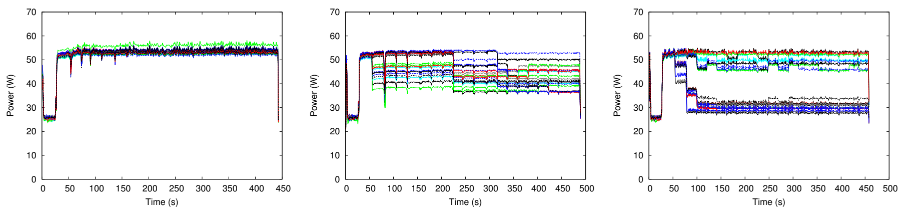
For *Ondes3D*, the least amount of energy spent is achieved using threshold value equal to 3.0. Using this value CG-EnergyLB is able to reduce in up to 31% the total energy consumption in relation to baseline noLB. This reduction is achieved through of the reduction of the average power demand in 34%, which overcome the time overhead of 5%, as shown in the Figure 2(a).

– *Lulesh* Application

Lulesh was executed with 1000 iterations in each one of its 5832 processes mapped in 192 cores. This application spent 840.6 kJoules of energy and takes 688 seconds when executed without load balancer. Thus, in this execution the average power demand is 50.9 W. These values are taken as reference (baseline) and shown in column noLB of the Figure 2(b)) to examine the threshold variation of the *Lulesh* application.

Instantaneous power measured when the application is executed with threshold equal to 0.5 is depicted in the Figure 4(a). Using this value, in all load balancing calls (19 times) were migrated tasks through do RefineLB. Similar to *Ondes3D* execution, for this threshold all processors running using a high power during all execution, which result in an average of power of 50.9 W.

On the other hand, using a threshold equal to 2.5, CG-EnergyLB is able to reduce the power demand to intermediate levels as shown in the Figure 4(b).



(a) Threshold = 0.5. (b) Threshold = 1.5. (c) Threshold = 5.0.

Fig. 4. Power evaluation to different threshold value on *Lulesh*

With this threshold, in this execution are performed 4 times DVFS, which reduced the power of the processors in 12.7%, to 43.4 W.

For threshold equal to 5.0 (Figure 4(c)) were adjusted 19 times the clock frequency of cores. This way, the power of most of the processors is reduced to minimum levels saving more energy. The total reduction was of 24%, which reduced the average power demand to 38.7 W.

The threshold variation from 0.5 up to 2.0 present the reduction more significant in energy consumption for this application. Differently from *Ondes3D*, when used these threshold values in CG-EnergyLB load balancer, the runtime increases in up to 14%, while that the average power demand reduces in up to 25%. In this way, the least amount of energy spent for *Lulesh*, is achieved using threshold value equal to 2.0. For this value, CG-EnergyLB reduces the total energy consumption in up to 15% if compared to baseline noLB. This reduction is achieved through of the reduction of the average power in 25%, which overcome an overhead of 14%, as shown in the Figure 2(b).

5 Conclusions

The exponential increase in power consumption related to a linear increase in the clock frequency and a higher complexity involved in the processors' design changed the course of development of new processors. Power consumption has become a critical aspect to the development of both large and small scale systems. This concern is now enough to warrant the research on techniques to improve the energy efficiency of parallel applications running on top of HPC platforms.

In this paper, we focused on analyze the trade-off between run time, power demand and total energy consumption of the variants of our energy-aware load balancer which aim to reduce the energy consumption and power demand of parallel applications without considerably degrading their overall performance.

Our results demonstrated that FG-EnergyLB can achieve energy savings of up to 17.1% with an average of 16.3%, and CG-EnergyLB of up to 31% with an average of 23% on real-world applications through the reduction of the average power demand. On the other hand, the total execution time happens to be reduced or increased according to threshold value. In this way, we can analyze which is the best threshold value for each application under the total energy consumption, total execution time or the average power demand focus.

This work can be extended in different directions. One possibility would be to develop a new load balancer that performs load balancing and DVFS at the same time in each load balancing step. For that, it would be necessary to create a heuristic that takes into account the cost of task migrations between cores/processors that operate in different clock frequencies. Another possibility would be to develop a hierarchical energy-aware load balancer that performs task migrations between cores of the same processor and only migrate tasks between processors when needed. In this scheme, only the processors involved in task migrations would need their clock frequencies to be adjusted, reducing overhead of performing DVFS on all processors at each load balancing step. Finally, we also intend to evaluate the benefits of FG-EnergyLB and CG-EnergyLB on other real-world applications and platforms.

Acknowledgments

This work was supported by CNPq, CAPES, FAPERGS and FINEP. This research has received funding from the European Community's Seventh Framework Programme (FP7-PEOPLE) under grant agreement number 295217, funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement number 689772 and STIC-AmSud/CAPES scientific-technological cooperation program under EnergySFE research project grant 99999.007556/2015-02.

References

1. Aupy, G., Benoit, A., Robert, Y.: Energy-aware scheduling under reliability and makespan constraints. In: Proceedings... pp. 1–10. International Conference on High Performance Computing (HiPC), IEEE Computer Society (2012)
2. Dosanjh, S., Barrett, R., Doerfler, D., Hammond, S., Hemmert, K., Heroux, M., Lin, P., Pedretti, K., Rodrigues, A., Trucano, T., et al.: Exascale design space exploration and co-design. Future Generation Computer Systems 30, 46–58 (2014)
3. Dupros, F., Aochi, H., Ducellier, A., Komatitsch, D., Roman, J.: Exploiting intensive multithreading for the efficient simulation of 3d seismic wave propagation. In: Proceedings... pp. 253–260. International Conference on Computational Science and Engineering, IEEE (July 2008)
4. Gerards, M.E., Hurink, J.L., Holzenspies, P.K., Kuper, J., Smit, G.J.: Analytic clock frequency selection for global DVFS. In: Proceedings... pp. 512–519. Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP) (2014)
5. Goel, B., McKee, S.A., Gioiosa, R., Singh, K., Bhadauria, M., Cesati, M.: Portable, scalable, per-core power estimation for intelligent resource management. In: Proceedings... pp. 135–146. International Green Computing Conference (IGCC), IEEE Computer Society (2010)

6. Hartog, J., Dede, E., Govindaraju, M.: Mapreduce framework energy adaptation via temperature awareness. Cluster Computing 17(1), 111–127 (2013), http://dx.doi.org/10.1007/s10586-013-0270-y
7. Hosseinimotlagh, S., Khunjush, F., Hosseinimotlagh, S.: A cooperative two-tier energy-aware scheduling for real-time tasks in computing clouds. In: Proceedings... pp. 178–182. Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP) (2014)
8. Huang, C., Lawlor, O., Kalé, L.V.: Adaptive mpi. In: Rauchwerger, L. (ed.) Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science, vol. 2958, pp. 306–322. Springer Berlin Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-24644-2_20
9. Isci, C., Buyuktosunoglu, A., Cher, C.Y., Bose, P., Martonosi, M.: An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In: Proceedings... pp. 347–358. International Symposium on Microarchitecture (MICRO), IEEE Computer Society (Dec 2006)
10. Kalé, L.V., Bohm, E., Mendes, C.L., Wilmarth, T., Zheng, G.: Programming Petascale Applications with Charm++ and AMPI. pp. 421–441. Chapman & Hall / CRC Press (2008)
11. Kalé, L., Bhandarkar, M., Brunner, R.: Load balancing in parallel molecular dynamics. In: Ferreira, A., Rolim, J., Simon, H., Teng, S.H. (eds.) Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science, vol. 1457, pp. 251–261. Springer Berlin Heidelberg (1998), http://dx.doi.org/10.1007/BFb0018544
12. Karlin, I., Bhatele, A., Chamberlain, B.L., Cohen, J., Devito, Z., Gokhale, M., Haque, R., Hornung, R., Keasler, J., Laney, D., Luke, E., Lloyd, S., McGraw, J., Neely, R., Richards, D., Schulz, M., Still, C.H., Wang, F., Wong, D.: Lulesh programming model and performance ports overview. Tech. Rep. LLNL-TR-608824 (December 2012), http://www.osti.gov/scitech/servlets/purl/1059462
13. Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B.L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., Still, C.: Exploring traditional and emerging parallel programming models using a proxy application. In: Proceedings... 27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013) (May 2013)
14. Kim, S.g., Eom, H., Yeom, H., Min, S.: Energy-centric DVFS controlling method for multi-core platforms. In: Proceedings... pp. 685–690. High Performance Computing, Networking, Storage and Analysis (SCC), IEEE Computer Society (Nov 2012)
15. Leung, J.Y.T.: Handbook of scheduling: algorithms, models, and performance analysis. Chapman & Hall/CRC (2004)
16. Menon, H., Jain, N., Zheng, G., Kalé, L.: Automated load balancing invocation based on application characteristics. In: Proceedings... pp. 373–381. IEEE International Conference on Cluster Computing (CLUSTER), IEEE Computer Society (2012)
17. Padoin, E., Castro, M., Pilla, L., Navaux, P., Mehaut, J.F.: Saving energy by exploiting residual imbalances on iterative applications. In: Proceedings... pp. 1–10. High Performance Computing (HiPC), 21st International Conference on (Dec 2014)
18. Sarood, O., Meneses, E., Kalé, L.V.: A 'cool' way of improving the reliability of HPC machines. In: Proceedings... pp. 58:1–58:12. International Conference on High Performance Computing, Networking, Storage and Analysis (SC), ACM (2013)

19. Spiliopoulos, V., Bagdia, A., Hansson, A., Aldworth, P., Kaxiras, S.: Introducing DVFS-management in a full-system simulator. In: Proceedings... pp. 535–545. International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE Computer Society (2013)
20. Tesser, R.K., Pilla, L.L., Dupros, F., Navaux, P.O.A., Mehaut, J.F., Mendes, C.: Improving the performance of seismic wave simulations with dynamic load balancing. In: Proceedings... pp. 196–203. Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE Computer Society (Feb 2014)
21. Zheng, G., Bhatelé, A., Meneses, E., Kalé, L.V.: Periodic hierarchical load balancing for large supercomputers. International Journal of High Performance Computing Applications 25(4), 371–385 (2011)