EnergyLB - saving energy while load balancing on HPC systems [*]

Edson L. Padoin[1,2], Laércio L. Pilla[3], Márcio Castro[3], Philippe O. A. Navaux[1], Jean-François Mehaut[4]
{elpadoin,navaux}@inf.ufrgs.br, {laercio.pilla,marcio.castro}@ufsc.br, jean-francois.mehaut@imag.fr

[1]Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS) – Porto Alegre, RS – Brazil
[2]Department of Exact Sciences and Engineering
Regional University of Northwest of Rio Grande do Sul (UNIJUI) – Ijui, RS – Brazil
[3]Department of Informatics and Statistics
Federal University of Santa Catarina (UFSC) – Florianópolis, SC – Brazil
[4]Grenoble Informatics Laboratory (LIG)
University of Grenoble – Grenoble – France

1. Introduction and Motivation

The main focus of HPC systems has been performance and today HPC community works toward building Exascale systems (i.e., EFlops), which will provide unprecedented computation, allowing to solve even larger scientific problems. Conceiving Exascale supercomputers by scaling the current technology would demand over a GigaWatt of power [7, 6], which is equivalent to the entire production of a medium size nuclear power plant [17]. With this problem in mind, a global research effort has risen to try to break the Exascale barrier while avoiding such high energy consumption. In 2008, specialists alerted on an official DARPA report [10, 2] that the acceptable power budget to reach Exascale is 20 MW. So, an Exascale system has a limit of $\frac{1\ EFlops}{20\ MW}$, i.e., 50 GFlops/W. Tianhe-2, the current number 1 supercomputer in the Top500 rank, produces 33.8 PFlops consuming 17.8 MW, thus Tianhe-2's energy efficiency is 1.91 GFlops/W. Using Tianhe-2 as a reference, we would yet to increase the energy efficiency 26-fold to match the DARPA recommendation. Thus, to get to Exascale we need to find alternatives to solve the power consumption problem [1, 18].

The first challenge to the development of energy efficient HPC systems and applications lies on increasing their performance while reducing their power consumption. Indeed, saving power has become one of the main concerns of HPC community. To build future Exascale systems we need to take into account power demand and energy consumption constraints and ways to improve energy efficiency [1, 18, 14, 13].

A second challenge arises as HPC systems grow in number of processors and this complicates efficiency once that scientific applications may not be easily equally distributed due to dynamic or irregular characteristics. These characteristics are present in several scientific applications and contribute to a reduced energy efficiency on parallel systems, since the most heavily loaded processor determines the application's performance. Several load balancing strategies have been used to improve the load distribution across processors and to achieve an efficient use of all available resources of a parallel machine.

In this context, state-of-the-art research focuses on either power consumption or load balancing strategies separately. Several works have used DVFS (dynamic voltage and frequency scaling), however this technique may cause performance degradation and an increase in execution time. Other works have used load balancing strategies to reduce the overall execution time, and, consequently, save energy. However, these strategies have used only computational load ignoring power demand or total energy consumption.

In this context, we plan to improve the energy efficiency of parallel systems when running load imbalanced applications. We intend to provide new load balancing strategies that collect system information from each node of the parallel machine and collect information from tasks of the application, in order to manage power demand while the application is running.

[*] This work was partially supported by CNPq, FAPERGS, FINEP, CAPES (under grants 3471-13-6, 5854-11-3, and 5847-11-7) and by the HPC-GA research project, which has received funding from the European Community's Seventh Framework Programme (FP7-PEOPLE) under grant agreement number 295217. This work was developed in the context of LICIA, an associated international laboratory between UFRGS and University of Grenoble.

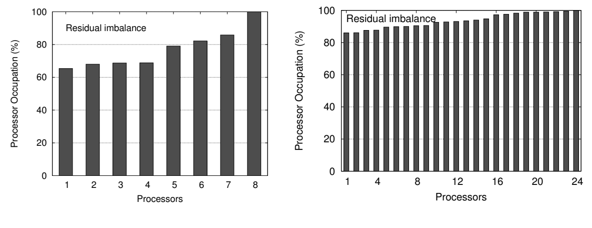
2. ENERGYLB

Reductions in the total execution time are relevant in the perspective of energy consumption, since energy is saved when hardware resources are used for a shorter time. However, it is possible to achieve even greater energy savings if the runtime system is able to exploit the residual imbalances to fine tune the voltage and frequency of cores accordingly. In this case, the challenge lies in reducing the energy consumption of the application while maintaining a similar performance.

In this context, iterative imbalanced applications are potential candidates for energy consumption improvements. These parallel applications are present in different scientific fields. Molecular dynamics, structural dynamics, weather forecast [12], cosmological modeling simulation [4, 8], seismic wave propagation simulations [5], physics [16], and oil exploitation [15], are examples of applications that can benefit from load balancing schemes to improve performance. Our point is that these schemes still leave space for energy optimization.

In an experimental evaluation, tests were performed using 8 and 24 cores (one on each processor) of the parallel machine presented in [13]. We ran a small test with a version of the seismic wave simulator *ondes 3D* [5] using the CHARM++ parallel programming [9, 3].

Load balancers improve the performance of imbalanced applications by making a better load distribution among the available processors. However, they can take sub optimal decisions, which may result in some imbalance remaining. This may happen due to characteristics of the application that prevent a perfectly balanced mapping to be achieved, or due to limitations of the load balancing heuristics, as the problem that they are trying to solve is NP-Hard [11]. In this work we call this remaining load imbalance *residual imbalance*.



(a) 8 processors. (b) 24 processors.

Figure 1. Processor utilization and residual imbalance after the first call of the load balancer GREEDYLB with different processors counts.

Figure 1 shows the residual imbalance when we executed the application with a load balancer named GREEDYLB and different processors counts. In the execution of the application in 8 processors, (Figure 1(a)), we have a residual imbalance of up to 34%. When we ran the application with the same workload in 24 processors, (Figure 1(b)), we obtained a different load distribution, with a remaining residual imbalance of 14% after load balancing. We notice that the residual imbalance is considerably high in several scenarios, as most of the processors are underused. In this case, DVFS techniques can be used to save energy by reducing the processor's frequency and, consequently, their power demand.

In this context, we introduce a new energy-aware load balancing algorithm called **ENERGYLB**, to save energy on iterative applications that present imbalanced loads. Our strategies can reduce the power demand during execution, saving energy and improving the application performance according to its load characteristic. We have implemented our proposed ENERGYLB, using the existing CHARM++ framework, which implements a model of migratable objects [13].

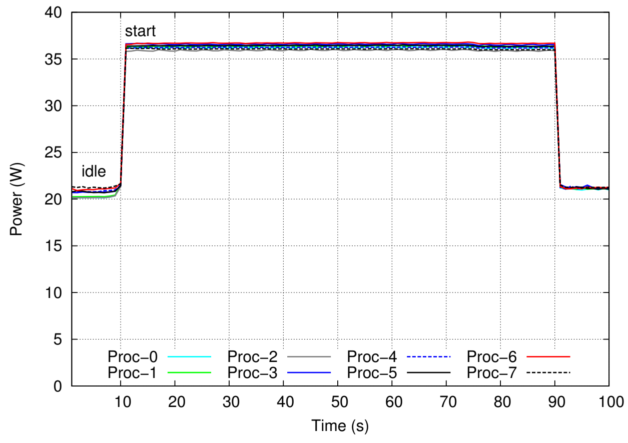
3. ENERGYLB Evaluation

We conducted our experimental evaluation on an SGI Altix UV 2000 system composed of 8 Intel Xeon E5-4640 Sandy Bridge-EP processors with 8 physical cores each. In these processors, there are 14 clock frequency levels available, varying from 1.2 GHz to 2.4 GHz. However, current processors do not allow DVFS on each core individually. So, in this work, tests were performed using 8 cores only, one on each socket.

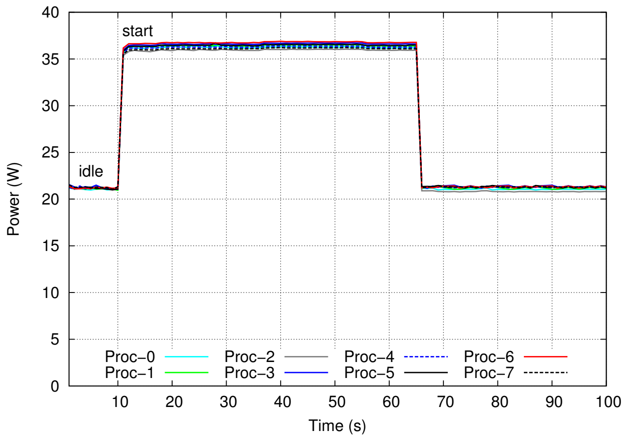
The operating system used in the machine is a UV2000 GNU/Linux distribution with kernel version 3.0.74. The application benchmark was compiled using gcc version 4.3 and the version 6.5.1 of Charm++. Results show the average of a minimum of 10 runs. The relative error is less than 5% using a 95% statistical confidence with Student's t-distribution.

Our strategy employs a load balancing module that benefits from the load balancing framework available in the CHARM++ runtime system. Aiming to evaluate DVFS only on our ENERGYLB proposal, we disable the load balancing functionality of ENERGYLB load balancer. In this context, we ran *lb_test* benchmark with 500 tasks for 250 iterations.

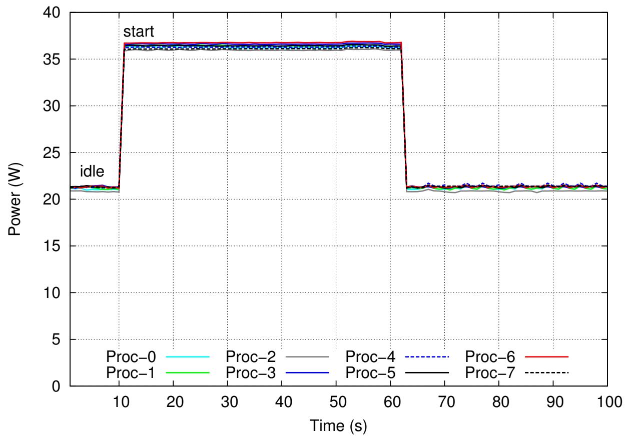
Figure 2 illustrates the instantaneous power demand measured during the test executions, without a load balancer (NOLB) (representing a baseline execution), using different CHARM++ load balancers (GREEDYLB and REFINELB), and our proposed ENERGYLB load balancer.



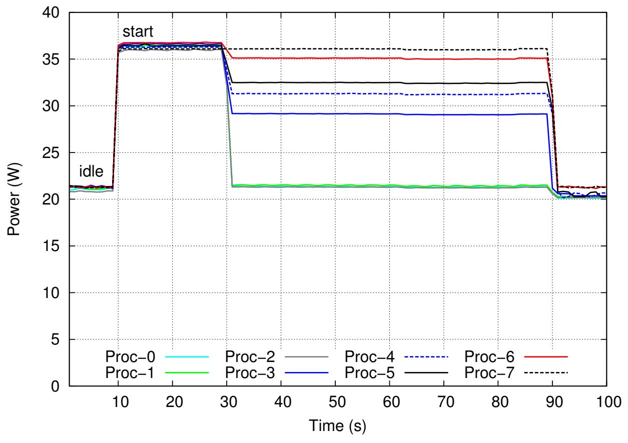
(a) NOLB.



(b) GREEDYLB.



(c) REFINELB.



(d) ENERGYLB- DVFS only.

Figure 2. Instantaneous power measured during execution of the imbalanced *lb_test* benchmark using different load balancers.

Figure 2(a) shows the power measured during the execution of the benchmark without any load balancer strategy. When load imbalanced applications are executed, the kernel governor does not realize the imbalance among processors, running the application using the maximum frequency available in the processors, which maintains the power demand constant (always close to 36W in this experiment).

When tests were performed using GREEDYLB and REFINELB, these load balancers perceived the load imbalance and migrated tasks, reducing the total execution time of the benchmark. Still, similarly to the scenario with NOLB, the highest frequency available in each processor was always employed, resulting in a demand of $\approx$ 36W by each processor. Nevertheless, load was not perfectly balanced among all processors, which lead to energy wasting.

When *lb_test* was run with ENERGYLB, all processors executed their task using the maximum frequency during initial iterations, explaining the power demand similar to NOLB. Nevertheless, after calling ENERGYLB, only one processor (the most loaded one) remained at 2.4 GHz. In other words, all others processors had their clock frequencies reduced to intermediate or to the minimum frequency. Consequently, the instantaneous power demand of each processor was also reduced according to their computational load, reducing the power demand. Figure 2(d) illustrates this phenomenon starting at the 20 seconds mark. As the benchmark does not present load variations, no change in the clock frequency of the processors was necessary during other ENERGYLB calls.

The greatest reduction in execution time was achieved when GREEDYLB was used. GREEDYLB load balancer obtain speedup of 1.42 when compared to baseline (NOLB). However, if we compare ENERGYLB results with GREEDYLB, we can observe that the execution time with ENERGYLB was 1.42 times longer, but the total energy consumption was only 1.19 times greater. This is justified due load imbalance present between tasks. In this test, the largest load difference between processors measured was of 9.2 times and ENERGYLB can only reduce the clock frequency by half (the range the processor goes from 2.4 to 1.2 GHz). Therefore, only adjusting the clock is not sufficient to mitigate the effects of load imbalance in energy consumption.

4. Conclusion

In this paper, we present an evaluation of ENERGYLB employing DVFS only (disabling the load balancing functionality of ENERGYLB). The results achieved show that, when load imbalance was greater than 2 times in the testbed platform, the use of ENERGYLB only is not enough to mitigate processor idleness. Therefore, to avoid this problem, the use of load balancing algorithms together with ENERGYLB is necessary. Due to this effect, we have implemented other approaches that automatically call another load balancer to migrate tasks between processors and then perform DVFS.

We showed that applications implemented in CHARM++ can benefit from ENERGYLB to improve energy efficiency, reducing of power demand during the execution of imbalanced applications, without considerably degrading their overall performance.

References

[1] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho. Using performance modeling to design large-scale systems. *Computer*, 42(11):42–49, 2009.

[2] P. Beckman, B. Dally, G. Shainer, T. Dunning, S. C. Ahalt, and M. Bernhardt. On the Road to Exascale. *Scientific Computing World*, (116):26–28, 2011.

[3] CHARM++. Parallel programming laboratory (ppl), 2014. http://charm.cs.illinois.edu/.

[4] M. D. Dikaiakos and J. Stadel. A performance study of cosmological simulations on message-passing and shared-memory multiprocessors. In *International Conference on Supercomputing (ISC)*, pages 94–101. ACM, 1996.

[5] F. Dupros, H. Aochi, A. Ducellier, D. Komatitsch, and J. Roman. Exploiting intensive multithreading for the efficient simulation of 3D seismic wave propagation. In *International Conference on Computational Science and Engineering (CSE)*, pages 253–260, 2008.

[6] W. Feng and K. Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12):50–55, 2007.

[7] C.-H. Hsu, W. chun Feng, and J. S. Archuleta. Towards efficient supercomputing: A quest for the right metric. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8–pp. IEEE, 2005.

[8] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. Quinn. Massively parallel cosmological simulations with ChaNGa. In *International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2008.

[9] L. V. Kalé and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. In *Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 91–108. ACM, 1993.

[10] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, and e. a. Denneau. Exascale Computing Study: Technology Challenges in achieving Exascale Systems. *DARPA IPTO*, pages 1–297, 2008.

[11] J. Y. T. Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. Chapman & Hall/CRC, 2004.

[12] J. Michalakes and M. Vachharajani. Gpu Acceleration of Numerical Weather Prediction. *Parallel Processing Letters*, 18(04):1–8, 2008.

[13] E. L. Padoin, M. Castro, L. L. Pilla, P. O. A. Navaux, and J.-F. Mehaut. Saving energy by exploiting residual imbalances on iterative applications. In *21st International Conference on High Performance Computing (HiPC)*, pages 1–10, Goa, India, 2014.

[14] E. L. Padoin, L. L. Pilla, F. Z. B. Boito, R. V. Kassick, P. Velho, and P. O. A. Navaux. Evaluating application performance and energy consumption on hybrid CPU+GPU architecture. *Cluster Computing*, 16(3):511–525, 2013. 10.1007/s10586-012-0219-6.

[15] J. Panetta, T. Teixeira, P. R. de Souza Filho, C. A. da Cunha Finho, D. Sotelo, F. M. R. da Motta, S. S. Pinheiro, I. P. Junior, A. L. R. Rosa, L. R. Monnerat, L. T. Carneiro, and C. H. de Albrecht. Accelerating Kirchhoff Migration by CPU and GPU Cooperation. *21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2009)*, pages 26–32, 2009.

[16] J. Shiers. The worldwide lhc computing grid (worldwide lcg). *Computer physics communications*, 177(1-2):219–223, 2007.

[17] M. Wehner, L. Oliker, and J. Shalf. A Real Cloud Computer. *IEEE Spectrum*, 46(10):24–29, 2009.

[18] A. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers. Efficient resource management for cloud computing environments. In *International Green Computing Conference (IGCC)*, pages 357–364. IEEE Computer Society, 2010.