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

Energy efficiency of load balancing for data-parallel applications in heterogeneous systems

Borja Pérez · Esteban Stafford · José Luis Bosque · Ramón Beivide

Received: date / Accepted: date

Abstract The use of heterogeneous systems in supercomputing is on the rise as they improve both performance and energy efficiency. However, the programming of these machines requires considerable effort to get the best results in massively data-parallel applications. Maat is a library that enables OpenCL programmers to efficiently execute single data-parallel kernels using all the available devices on a heterogeneous system. It offers a set of load balancing methods, which perform the data partitioning and distribution among the devices, exploiting more of the performance of the system and consequently reducing execution time. Until now, however, a study of the implications of these on the energy consumption has not been made. Therefore, this paper analyses the energy efficiency of the different load balancing methods compared to a baseline system that uses just a single GPU. To evaluate the impact of the heterogeneity of the system, the GPUs were set to different frequencies. The obtained results show that in all the studied cases there is at least one load balancing method that improves energy efficiency.

Keywords Heterogeneous systems · load balancing · energy efficiency · OpenCL

1 Introduction

One of the current efforts towards increasing the computing power of high performance computers is to include GPUs alongside general purpose processors. Such heterogeneous systems are used for a wide spectrum of scientific applications [1–3]. This is a solution adopted by desingers facing the challenge of the Exa-scale computing milestone, as the high processing power of GPUs

Borja Pérez · Esteban Stafford · José Luis Bosque · Ramón Beivide
Department of Ingeniería Informática y Electrónica
University de Cantabria
E-mail: perezpavonb@unican.es,stafforde@unican.es,bosquejl@unican.es,beividej@unican.es

allows these systems to increase the FLOPs per Watt ratio. Recently Nvidia has announced its DGX-1 HPC server, which includes two Haswell processors with 32 cores and 8 NVIDIA Tesla P100 GPUs. The combined computing power of the DGX-1 is 85 TFlops. Only four of these systems put together would grant a place in the Top500 list of supercomputers (June 2016).

Harnessing the computing power of these systems is a considerable challenge by itself. The most common programming model, being used by CUDA and OpenCL is the host-device model. It dictates that the host processor starts the execution of an application, and offloads highly parallel parts to the GPU. In general, the host waits for the completion of these code sections. Consequently, this model does not exploit the computing power of the host, as it remains idle while the GPU is in operation. Interestingly, the CPUs are still consuming a noticeable amount of power during these periods, significantly reducing the energy efficiency of the system as a whole. This model does not exploit the computing power of several GPUs by default either, as load distribution to the different GPUs is in the hands of the programmer.

Maat is an OpenCL library that hides from the developers all the different computing elements of a heterogeneous system. This guarantees code portability and improves productivity [4]. With Maat, massively data-parallel applications can be programmed using the host-device model, by using a load balancing algorithm it distributes the workload among all the available computing elements. Maat offers a set of load balancing algorithms that can be used for applications of different nature. Choosing and adequately configuring the algorithm is fairly simple and can considerably improve the performance of the system. This is because all devices perform useful work all the time, and none remains idle waiting for another.

However, using all devices simultaneously to solve a single problem affects the energy consumption in two ways. First it increases the power drawn by all the devices. And second, it reduces the execution time of the application. Since energy is calculated as the product of power and time, these two facts alone do not determine if there is an overall energy saving or not. Therefore, it is necessary to carry out an analysis of the impact of the load balancing algorithms on real systems. To the best of the authors' knowledge, such an analysis has not been yet performed.

To better understand the implications of different load-balancing strategies, this paper proposes a study of the energy efficiency of a whole heterogeneous system composed by several CPUs and GPUs. In order to widen the study, the GPUs are operated at different frequencies, thus varying their apparent performance. To evaluate the energy efficiency, the execution time of several benchmarks and the energy consumed by the system are measured. The execution of each benchmark is repeated with different load balancing algorithms and GPU frequencies. These results are compared with the baseline scenario that consists of a heterogeneous system in which only one GPU is used.

The experimental results show some interesting conclusions. The first is that for all the benchmarks and all the frequencies analysed, there is a load balancing algorithm that improves the baseline results, both in execution time

and energy consumption. Second, the higher frequency on the GPU, greatly reduces the total energy consumption for all benchmarks analysed. The higher power consumption throughout the execution is compensated by the reduction in execution time, thereby reducing the total energy and thus duplicating energy efficiency.

The rest of the paper is organised as follows. A description of the different load balancing methods implemented in Maat is presented in Section 2. Followed by Section 3 that describes the experimental environment and measurement tools employed, and Section 4 presents the obtained results. Section 5 presents some previous works related to this paper. Finally, Section 6 shows the most interesting conclusions of this study and proposes future work.

2 Load Balancing Algorithms

The goal of a load balancing algorithm is to distribute the workload among the devices proportionally to their computational capabilities. This is specially challenging in heterogeneous systems and with irregular applications, as it makes coming with a proportional workload division harder. As a consequence, a single load balancing algorithm will not deliver the best results for every possible system and workload. Therefore Maat provides three algorithms, each of them more suited to a different set of benchmarks [4]. All three algorithms focus on the balancing of data-parallel workloads, in which every device performs the same computation on a disjoint partition of the data.

2.1 Static Balancing

This algorithm is based on dividing the load in as many portions as devices are available in the system. Then, it assigns a single portion to each of them. To obtain good performance, the work-package assigned to each device has to have a size proportional to its *computational speed*. This is defined as the amount of work that each device can complete in a time unit, including the necessary communication overhead.

Let there be a heterogeneous system $H = \{D, P\}$, where $D = \{d_1, \cdots d_n\}$ is the set of devices and $S = \{S_1, \cdots S_n\}$ are their computational speeds. Additionally, consider a work-load that needs to process W *work-items* grouped in G *work-groups*. In OpenCL, work-groups can be executed concurrently. For this reason they should be considered as the atomic distribution unit. All the work-groups have the same size, called *local work size*.

The response time of the heterogeneous system is that of the last device to finish its work. Therefore, the goal of the static method is to find a tuple $\{\alpha_1, \cdots \alpha_n\}$, where α_i is the number of work-groups assigned to the device d_i, such that all the devices finish their work at the same time, and then the system response time is minimized:

$$T_H = T_{d_1} = \cdots = T_{d_n} \Rightarrow \frac{\alpha_1}{S_1} = \cdots = \frac{\alpha_n}{S_n}$$

Following the optimal algorithm proposed by O. Beaumont in [5] this can be done with complexity $O(n^2)$ in two steps:

- First, α_i is calculated so that $\frac{\alpha_i}{S_i}$ is almost constant $\forall\ i \in [1, \cdots n]$, and $\alpha_1 + \alpha_2 + \cdots + \alpha_n \leq G$:

$$\alpha_i = \left\lfloor \frac{S_i}{\sum_{i=1}^{n} S_i} \cdot G \right\rfloor$$

- Second, if $\sum_{i=1}^{n} \alpha_i < G$, then the remaining work-groups are assigned to the most powerful device. This amount of work is practically negligible, and does not disturb the load distribution.

This algorithm guarantees that the number of synchronization points is minimized, and performs well when facing regular loads, provided that the computational powers of the devices are accurately known. However, it is not adaptable, so performance is not so good for irregular loads.

2.2 Dynamic Balancing

This algorithm divides the load in small, equally-sized packages, many more than the amount of available devices. The runtime orchestration is carried out by a *master thread* that follows the next algorithm:

1. The master splits the number of work-groups G, in a set of p packages, all of them with the same size and launches one package on each device.
2. The master waits for the completion of any package.
3. When device d_i completes the execution of a package:
   (a) The master stores results returned by the device.
   (b) If there are outstanding packages the next package is assigned to d_i.
   (c) Else, if d_i is a GPU and there is a busy CPU, d_i steals the package from the CPU.
   (d) If none of this conditions is met, the master proceeds to step 4.
   (e) The master returns to step 2.
4. The master ends when all the packages have been processed and their results are stored in the host.

The dynamic approach can adapt to different hardware and workload scenarios where the static can not, but it increases the communication overhead, with respect to the static approach.

2.3 H-guided Balancing

The guided algorithm is designed to reduce the amount of synchronization points inherent to the dynamic scheme. Therefore, it revolves about the same basic algorithm, but instead of equal-sized packages, the size diminishes with the number remaining work-groups. Let G_r be the number of pending work-groups and n the number of available devices, the package size si computed as $Package\_size = \lfloor \frac{G_r}{n} \rfloor$. This effectively reduces the number of synchronization points, while maintaining adaptability. However, for heterogeneous systems, a big package may be assigned to a slow device, delaying the whole program.

To avoid this, Maat includes the *h-guided* algorithm that is a refinement for heterogeneous systems. This algorithm takes into account the computational power of the device. Then, considering S_i as the computational power of device d_i, the size of the packages is calculated as:

$$Package\_size = \left\lfloor \frac{S_i}{\sum_{i=1}^{n} S_i} \cdot \frac{G_r}{n} \right\rfloor$$

3 Methodology

To assess the energy efficiency of the different algorithms presented above, a combination of experimental investigation and comparative analysis was employed. First a set of representative applications were selected, and while these were run on a real heterogeneous system, their execution time and power consumption were measured. Second, the enregy efficiency was computed for the different applications and algorithms in order to develop a series of conclusions.

3.1 Experimental Setup

The heterogeneous system is composed of two CPUs, two GPUs and 16 GBs of DDR3 memory. The CPUs are Intel Xeon E5-2620, with six cores that can run two threads each at 2.0 GHz. The CPUs are connected via QPI, which results in OpenCL detecting them as a single device. Therefore, throughout the remainder of this document, any reference to the CPU includes both Xeon E5-2620 processors. The GPUs are NVIDIA K20m with 13 SIMD lanes, 2496 cores and 5 GBytes of VRAM each. These are connected to the system using independent PCI 2.0 slots.

All the experiments have been performed with all the frecuencies supported by the GPU: 324, 614, 640, 666, 705 and 758 MHz. Increasing the frequency naturally escalates the power consumption and reduces the execution times, all having an impact in the energy efficiency of the system. At the lowest frequency, the computing speed of GPU is comparable to that of the CPU, thus making the system less heterogeneous.

For the comparative analysis, a *baseline* configuration was used, that consists of the same heterogeneous system but using only one GPU.

3.2 Benchmarks

Five applications have been chosen for the experiments. Three of them are part of the AMD APP SDK[6]. MatMul, NBody, Binomial, are well-known and regular applications in which different, equal-sized work units have the same running times. The other two applications, which are in-house implementations of known algorithms, are examples of irregular workloads in which different, equal-sized work units may have different running times. First, RAP is an implementation of the Resource Allocation Problem. It must be noted that there is a certain pattern in the irregularity of RAP, as each successive package represents a bigger amount of work than the previous. Second, a raytracing algorithm (RAY) was implemeted as an example of a truly irregular workload. This computes a realistic rendering of a scene by following light rays with independent threads. Thus, each of them represents an unpredictable amount of work, as the number of ray bounces depends on the objects of the scene.

Each application has been run using a problem size big enough to justify its distribution among all the available devices. For MatMul 12800 by 12800 matrices were used. For NBody 51200 elements were considered for simulation. Binomial uses 20480000 options. In RAP, the input was a matrix of 1024 by 1024 resources. Finally, for RAY, a 12000 by 12000 image was generated from a scene with 12 different spheres. Local work size has been set so the performance of the fastest device, namely the GPU, is maximised. Moreover, almost no performance difference was detected when varying local work size for the CPU. The selected values are: 16 by 32 for MatMul, 128 for NBody, 256 for Binomial and 64 for both RAP and RAY.

3.3 Performance, Power and Energy Measurements

The performance has been measured as the *speedup* with respect to the baseline execution time for each frequency, using OpenCL and a single GPU. Unlike homogeneous systems, in which the maximum speedup is the number of computing devices, for heterogeneous systems this peak performance value depends on the relative computing power of the different devices as well as their number. Table 1 shows the maximum speedups that can be obtained by each bechmark at the different frequencies considered.

Table 1 Maximum speedup for different benchmarks and frequencies.

Bench./Freq.	324	614	640	666	705	758
MatMul	2.27	2.11	2.10	2.10	2.09	2.09
Nbody	2.33	2.16	2.15	2.14	2.14	2.13
Binomial	2.30	2.16	2.15	2.14	2.14	2.13
RAP	2.65	2.27	2.26	2.25	2.23	2.22
RAY	2.27	2.12	2.12	2.11	2.11	2.10

To measure the energy consumption of the system it is necessary to take into account the power drawn by each of the devices. Modern computing devices include *Performance Management Units (PMU)* that allow applications to measure the current power consumption. This power is associated to the device and not the kernel or process in execution. Together with the fact that it is impractical to add measurement code to applications, this led to the development of a power monitoring application. This tool, named *Sauna* takes a program as its parameter, and is able to configure the different PMUs, run the program and perform periodic power measurements on the different devices. During the execution, it shows the instant power of each device. By default Sauna's measurement is active throughout the execution, but it can also be restricted to a *Region Of Interest(ROI)* as determined by the program itself by special strings written to standard output.

During the measurement Sauna executes calls to the different devices APIs in order to access the PMU data. For the Intel CPUs, recent versions of the Linux kernel provides access to the *Running Average Power Limit (RAPL)* registers [7], which provide accumulative energy readings. On contrast, NVIDIA provides a library to access their PMUs, this *NVIDIA Management Library (NVML)* [8] gives instant power measurements. Naturally, Sauna is able to convert between the two magnitudes.

However it is important to notice that the precision of the power to energy conversion is dependent on the sampling rate. To determine the most adequate sampling rate, a series of test measurements were performed. On one hand, it was observed that the use of the NVML hinders the communication with the GPU, and therefore, with high sampling rates the execution time of the application under test grew significantly. On the other hand, a low sampling rate might miss power consumption bursts and, consequently, reduce the precision of the measurement. Empirically it was decided that for the test system, the best sampling period was 30ms.

The performance and the energy consumption can be combined in a single metric representing the energy efficiency of the system. This paper uses the Energy-Delay-Product (EDP) [9] for this purpose.

4 Experimental Evaluation

This section describes the experimental results obtained and dicusses the most interesting conclusions, extracted from them.

Figure 1 presents the speedup of each load balancing algorithm, benchmark and frequency, referred to the performance of a single GPU using the corresponding frequency. Note that the maximum achievable speedups are shown in Table 1 and that speedup comparisons between frequencies may not be meaningful, as each frequency uses a different baseline and the relative computing power among devices varies. It is important to highlight that for any of the analysed cases, there is always a load balancing algorithm that greatly

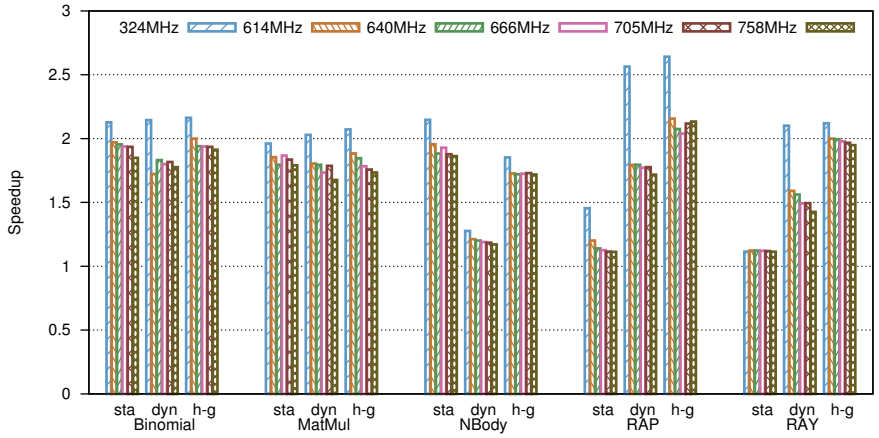

Fig. 1 Speedup for each algorithm relative to a single GPU.

improves the performance of the baseline scenario, getting close to the ideal speedup.

The speedup of the **regular applications** (Binomial, MatMul and NBody) is limited by both the sensitivity to the number of packages and the performance differences between devices. Regarding the algorithms, observe that the behaviour of these benchmarks is very similar in all frequencies, and that the best algorithms are the static and h-guided with very similar speedups. As a result, we can conclude that for regular applications, in which different work items have equal running times, the main performance limiting factor is overhead. The dynamic algorithm also manages to deliver good performance excluding the case of NBody, which is very sensitive to the number of packages. From the frequency point of view, all the speedups are close to the corresponding maximum speedup values. The execution times of all benchmarks are best with the highest frequency.

The analysis of the **irregular applications** (RAP and RAY) is very similar for all frequencies, from which the h-guided method obtains the best results. This is because the size of the packages assigned is proportional to the computing power of the device and inversely proportional to the number of devices, which results in reduced overheads while maintaining adaptability. Thus, the CPUs can contribute processing small packages, while the majority of the computation is carried out by the GPUs. For the lowest frequency, it can be observed that the dynamic algorithm obtains good results, this is a consequence of the machine becoming less heterogeneous and the CPU contributing a significant amount of work. The static algorithm does not deliver good results for this applications. Consequently, the key performance limiting factor for irregular applications is adaptability, although overhead also plays a role, as shown by the speedup of the dynamic algorithm.

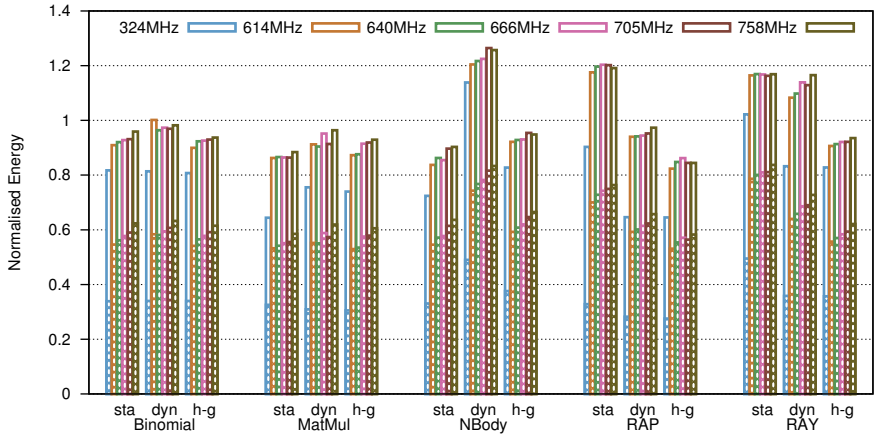

Fig. 2 Energy consumption for each algorithm normalised to the baseline configuration.

Attending to energy considerations, Figure 2 shows the normalised energy consumption of the system for each benchmark, load balancing algorithm and GPU frequency. The absolute energy values were normalised to the baseline configuration for each frequency. The bars on the graph are divided to separately show the energy corresponding to the twelve CPU cores and the two GPUs.

The first conclusion to highlight is that, despite using two GPUs and two CPUs, the total energy consumption of the heterogeneous system is less than that of the baseline. Therefore, there is at least one load balancing algorithm that improves the baseline energy consumption. Again, the most suitable algorithms for regular benchmarks are the static and h-guided, and for irregular ones it is clearly h-guided. This comes as a consequence of two improvements: the reduction in the execution time of the benchmarks, and that all the devices are contributing useful work instead of being idle, improving the energy efficiency of the system. This behaviour is exacerbated with the minimum GPU frequency, because the CPU can contribute a more significant amount of work with respect to the energy it consumes. However, the energy consumption is relatively less with the maximum frequency.

Finally, Figure 3 shows the results of the energy efficiency in terms of the Energy-Delay Product, again normalised to that of the baseline configuration. This metric combines the performance and energy in a single value, and its reduction is the ultimate goal of the load balancing algorithms. Therefore this graphs shows a combination of the two previous ones, and most of the observations made above are confirmed by the results plotted in this graph.

In general, it can be said that there is always a load-balancing algorithm that halves the EDP of the baseline, meaning that the energy efficiency of the heterogeneous system is doubled. However, failing to choose an adequate load balancing algorithm might only not save energy but also be inefficient. This

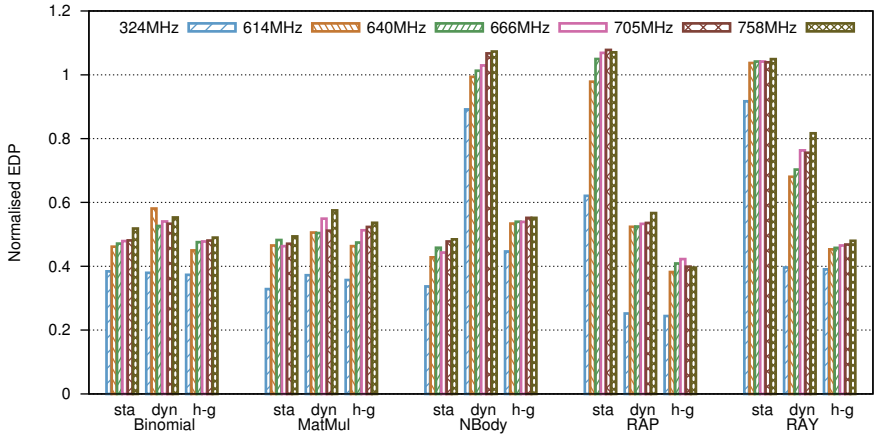

Fig. 3 EDP for each algorithm normalised to the baseline configuration.

can be seen in the static algorithm with irregular benchmarks or the dynamic for NBody, where the normalised EDP is larger than 1. Another aspect to note is that for some cases, the combination of both energy and time results in a reduction of EDP even though the energy for that case is higher than the baseline. This means that the reduction of execution time makes up for the increase in energy. This can be seen in RAY when using the dynamic approach.

Finally regarding the frequencies, an in-depth analysis of the absolute energy and EDP values has shown that the best results are yield with the highest frequencies. This once again implies that the reduced running time overcomes the increased energy consumption, of the highest frequency.

5 Related Work

To the problem of load balancing in heterogeneous environments two main approaches are found in the literature: static and dynamic algorithms, which differ in the time in which the allocation decisions are taken.

Some examples of static load balancing algorithms are [10] and [11]. The first one proposes a library that implements static load balancing by encapsulating standard OpenCL API calls. On the other hand, [11] approaches the problem by automatically modifying OpenCL code, so the load is balanced and each of the available devices works on a subset of the data.

In the dynamic approach, Binotto *et al.* [12] introduce a load balancer that establishes an initial scheduling guess based on different estimation costs and then analyses changes in runtime conditions. On contrast, the aproaches of Boyer *et al.* [13] and Kaleem *et al.* in [14] get information about the execution time of the first portions of the workload, and take the load distributions decisions based on this information.

Power consumption is a hot topic on heterogeneous systems, and therefore several authors have taken it into account. Hong and Kim develop an integrated power and performance prediction model, to estimate the optimal number of active processors for a given application [15]. Abe *et al.* present power and performance characterization and modelling of GPU-accelerated systems [16], while [17] studies how the performance-per-watt of GPUs is affected by temperature, core clock frequency and voltage. Some authors have proposed methods to accurately compute the instant power and the energy consumption of GPUs using NVIDIAs Management Library (NVML) [18] or experimentally study the impacts of Dynamic Voltage and Frequency Scaling (DVFS) on application performance and energy efficiency for GPU computing and compare them with those of DVFS for CPU computing [19].

Some papers propose algorithms to distribute workload between CPU and GPU taking performance and power into account. For instance, GreenGPU dynamically distributes workloads to GPU and CPU, minimizing the energy wasted on idling and waiting for the slower side to finish [20]. Moreover, to maximize energy savings while allowing marginal performance degradation, it dynamically throttles the frequencies of CPU, GPU cores and memory, based on their utilizations. Wang and Ren [21] propose a power-efficient workload distribution method for single applications on CPU-GPU systems. The method can coordinate inter-processor work distribution and frequency scaling to minimize energy consumption under a given scheduling length constraint.

E-ADITHE is an approach for improving energy efficiency for iterative computation on integrated GPU-CPU systems [22]. It describes an heuristic scheme for processing device selection according to the estimation of energy efficiency to balance the heterogeneous system. In [23], the authors present an Integer Linear Programming (ILP) based framework that maps a given task-set onto an Heterogeneous Multiprocessor System-on-Chip architecture. They use Dynamic Voltage Scaling (DVS) to reduce energy consumption while employing task duplication to maximize reliability. [24] performs an experimental study of the interactions between performance, power and energy in low-power ARM and GPU architectures when executing the RX algorithm.

All these approaches fail to address irregular applications, focus on a certain type or number of devices or schedule different kernels but not a single and massive data-parallel kernel. To the extent of the authors knowledge, this is the first paper that presents a study of the performance and energy efficiency of a set of load balancing methods for data parallel and irregular applications in heterogeneous systems.

6 Conclusions

This paper presents an evaluation of the energy consumption of a set of load balancing algorithms for massively data-parallel applications in heterogeneous systems. By harnessing the computing capability of the whole heterogeneous system, these algorithms not only shorten the execution time of the applica-

tions, but also reduce their energy consumption, thus obtaining an improvement of the energy efficiency of the whole system.

The experimental results presented in this paper show that for both, regular and irregular applications, there is always a load balancing algorithm that reduces the execution time as well as the energy consumption. As a consequence, the EDP is also reduced, showing a duplication in the energy efficiency of the whole heterogeneous system. As is expectable, the best results are obtained with the highest GPU frequencies.

These results allows answering the question as to whether there is an energy saving when using all the devices in the heterogeneous machine simultaneously. The fact that the machine is running during less time overcomes the greater power drawn by the devices, which would still consume a significant amount of energy to no avail if left idle. Thus, a machine with a greater amount of GPU devices should, not only give an enormous computing capability, but it would do so in an energy efficient manner. Nevertheless, this is only achievable with the use of an adequate load balancing algorithm.

The best overall results are obtained with the h-guided algorithm, yet the dynamic also gives very good results in irregular applications. Furthermore, dynamic does not require prior knowledge of the power of the computing devices, thus making it a good first approach to an unknown system. The static algorithm is appropriate for homogeneous environments and regular applications.

Future work will afford the study of more benchmarks in order to confirm the conclusions obtained in this paper. The study could also be broadened by extending the tests to other hardware accelerators, like Systems-On-Chip (SOC) with integrated GPUs. Additionally, new load balancing methods that take into account energy efficiency together with performance will be included in Maat library.

References

1. Peter Benner, Alfredo Remón, Ernesto Dufrechou, Pablo Ezzatti, and Enrique S. Quintana-Ortí. Extending lyapack for the solution of band lyapunov equations on hybrid cpu–gpu platforms. *The Journal of Supercomputing*, 71(2):740–750, 2015.
2. Xianggao Cai, Guoming Lai, and Xiaola Lin. Forecasting large scale conditional volatility and covariance using neural network on gpu. *The Journal of Supercomputing*, 63(2):490–507, 2013.
3. Kyle E. Niemeyer and Chih-Jen Sung. Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. *The Journal of Supercomputing*, 67(2):528–564, 2014.
4. Borja Pérez, José Luis Bosque, and Ramón Beivide. Simplifying programming and load balancing of data parallel applications on heterogeneous systems. In *Proc. of the 9th Workshop on General Purpose Processing using GPU*, pages 42–51, 2016.
5. Olivier Beaumont, Vincent Boudet, Antoine Petitet, Fabrice Rastello, and Yves Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Trans. Computers*, 50(10):1052–1070, 2001.
6. Amd accelerated parallel processing software development kit v2.9. Last accesed November 2015.

7. Efraim Rotem, Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eli Weissmann. Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge. In *IEEE Int. HotChips Symp. on High-Perf. Chips (HotChips˜2011)*, 2011.
8. NVIDIA. NVIDIA Management Library (NVML). Last accesed April 2016.
9. Emilio Castillo, Cristóbal Camarero, Ana Borrego, and Jose Luis Bosque. Financial applications on multi-cpu and multi-gpu architectures. *J. Supercomput.*, 71(2):729–739, February 2015.
10. Carlos S. de la Lama, Pablo Toharia, Jose Luis Bosque, and Oscar D. Robles. Static multi-device load balancing for opencl. In *Proc. of ISPA*, pages 675–682. IEEE Computer Society, 2012.
11. Janghaeng Lee, Mehrzad Samadi, Yongjun Park, and Scott Mahlke. Transparent CPU-GPU Collaboration for Data-parallel Kernels on Heterogeneous Systems. In *Proc. of PACT*, pages 245–256, Piscataway, NJ, USA, 2013. IEEE Press.
12. A.P.D. Binotto, C.E. Pereira, and D.W. Fellner. Towards Dynamic Reconfigurable Load-balancing for Hybrid Desktop Platforms. In *Proc. of IPDPS*, pages 1–4. IEEE Computer Society, April 2010.
13. Michael Boyer, Kevin Skadron, Shuai Che, and Nuwan Jayasena. Load Balancing in a Changing World: Dealing with Heterogeneity and Performance Variability. In *Proc. of the ACM International Conference on Computing Frontiers*, pages 21:1–21:10, 2013.
14. Rashid Kaleem, Rajkishore Barik, Tatiana Shpeisman, Brian T. Lewis, Chunling Hu, and Keshav Pingali. Adaptive heterogeneous scheduling for integrated GPUs. In *Proc. of PACT*, pages 151–162, New York, NY, USA, 2014. ACM.
15. Sunpyo Hong and Hyesoon Kim. An integrated gpu power and performance model. *SIGARCH Comput. Archit. News*, 38(3):280–289, June 2010.
16. Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres. Power and performance characterization and modeling of gpu-accelerated systems. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 113–122, May 2014.
17. D. C. Price, M. A. Clark, B. R. Barsdell, R. Babich, and L. J. Greenhill. Optimizing performance-per-watt on gpus in high performance computing. *Computer Science - Research and Development*, pages 1–9, 2015.
18. Martin Burtscher, Ivan Zecena, and Ziliang Zong. Measuring gpu power with the k20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*, GPGPU-7, pages 28:28–28:36, New York, NY, USA, 2014. ACM.
19. Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *Proceedings of the 42 Int. Conference on Parallel Processing*, ICPP '13, pages 826–833, 2013.
20. Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. GreenGPU: A holistic approach to energy efficiency in GPU-CPU heterogeneous architectures. In *41st International Conference on Parallel Processing, ICPP 2012*, pages 48–57, 2012.
21. G. Wang and X. Ren. Power-efficient work distribution method for cpu-gpu heterogeneous system. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 122–129, Sept 2010.
22. E. M. Garzón, J. J. Moreno, and J. A. Martínez. An approach to optimise the energy efficiency of iterative computation on integrated gpu–cpu systems. *The Journal of Supercomputing*, pages 1–12, 2016.
23. Suleyman Tosun. Energy- and reliability-aware task scheduling onto heterogeneous mpsoc architectures. *The Journal of Supercomputing*, 62(1):265–289, 2012.
24. G. León, J. M. Molero, E. M. Garzón, I. García, A. Plaza, and E. S. Quintana-Ortí. Exploring the performance–power–energy balance of low-power multicore and manycore architectures for anomaly detection in remote sensing. *The Journal of Supercomputing*, 71(5):1893–1906, 2015.