J Supercomput
https://doi.org/10.1007/s11227-018-2318-5

Load balancing in a heterogeneous world: CPU-Xeon Phi co-execution of data-parallel kernels

Raúl Nozal[1] · **Borja Perez**[1] · **Jose Luis Bosque**[1] · **Ramón Beivide**[1]

Abstract Heterogeneous systems composed by a CPU and a set of different hardware accelerators are very compelling thanks to their excellent performance and energy consumption features. One of the most important problems of those systems is the workload distribution among their devices. This paper describes an extension of the Maat library to allow the co-execution of a data-parallel OpenCL kernel on a heterogeneous system composed by a CPU and an Intel Xeon Phi. Maat provides an abstract view of the heterogeneous system as well as set of load balancing algorithms to squeeze the performance out of the node. It automatically performs the data partition and distribution among the devices, generates the kernels and efficiently merges the partial outputs together. Experimental results show that this approach always outperforms the baseline with only a Xeon Phi, giving excellent performance and energy efficiency. Furthermore, it is essential to select the right load balancing algorithm because it has a huge impact in the system performance and energy consumption.

Keywords Heterogeneous computing · Co-execution CPU-Xeon Phi · Load balancing · OpenCL · Performance portability · Energy efficiency

✉ Raúl Nozal
raul.nozal@unican.es

Borja Perez
borja.perez@unican.es

Jose Luis Bosque
joseluis.bosque@unican.es

Ramón Beivide
ramon.beivide@unican.es

[1] Computer Science and Electronics Department, University of Cantabria, Santander, Spain

1 Introduction

One of the most important challenges of high-performance computing today is to reach Exascale computers. Nowadays one of the most promising ways is the use of cluster-based architectures with nodes that have great computing capacity. These nodes are based on multi-core processors and include hardware accelerators, such as GPUs or Intel Xeon Phi.

Nonetheless, these nodes introduce new challenges, because the use of accelerators turns them heterogeneous. Hence, the software development to efficiently exploit all the available resources has to take into account this heterogeneity. Therefore, the use of a portable programming language on heterogeneous platforms is mandatory. Open Computing Language (OpenCL) [14] is a framework for developing programs to be executed across heterogeneous platforms such as many-core and multi-core architectures.

However, when using OpenCL, the programmer is responsible for explicitly selecting and managing the devices as well as for partitioning the data among them. Therefore, load balancing becomes one of the most challenging problems, having a tremendous impact on performance and programmability. The objective of load balancing algorithms is distributing the workload proportionally to the devices' computational power. This problem is more acute in heterogeneous systems with irregular applications.

This paper extends Maat [12] to allow the co-execution of massively data-parallel kernels on heterogeneous systems composed by a multi-core CPU and an Intel Xeon Phi accelerator. Maat is an OpenCL library that provides the programmer with a unified and abstract view of the heterogeneous system that guarantees code portability. Furthermore, Maat provides a set of load balancing algorithms that allow the programmer to choose the most appropriate depending on the behaviour of the application at hand. In this paper, a diverse set of applications is considered. They are grouped as regular, when every work unit represents the same running time, and irregular, if different work units have different running times. This decision is critical in order to achieve the best performance and energy efficiency, with different applications, as will be shown in Sect. 6. Experiments show results close to the optimum achievable, reaching an efficiency of at least 0.97 and 0.85 per benchmark for both regular and irregular applications, respectively. The power consumption show savings up to 25 and 50% for regular and irregular ones.

The remainder of the paper is structured as follows. Section 2 introduces some basic concepts that are important to the article. Section 3 describes the main characteristics of the three load balancing techniques. Section 4 explains the extensions developed on Maat library. Next, in Sects. 5 and 6 the methodology and evaluation of the algorithms are explained with a performance and energy evaluation of the three load balancing algorithms with different benchmarks. Then, in Sect. 7 the state of the art is presented, and multiple studies have been analysed and compared with our work. Finally, Sect. 8 summarises the main conclusions and future work.

2 Background

OpenCL is an open standard for portable parallel programming on heterogeneous architectures, so that applications written in OpenCL can run on different architectures without code modification. A major benefit of using OpenCL is that the same code can be easily executed on different platforms.

A *context* in OpenCL is a set of devices of the same vendor that can run OpenCL. Each device comprises a set of *compute units* (cores in a CPU and MIC, stream multiprocessors in GPUs), being the minimum element that can execute work. The code to be executed on the devices is encapsulated as a data-parallel C-like function known as *kernel*. OpenCL launches multiple instances of a kernel when it is offloaded to a device, each one with a portion of the data. Each instance is called a *work-item*. The total number of work-items to be executed is set by the programmer with the value *global work size*.

An OpenCL kernel describes the behaviour of a single work-item, and the host application specifies the total work-items to express the parallelism of the application. A *work-group* is a team of work-items that can synchronise and cooperate with each other. All the work-items in a work-group are launched to the same compute unit. However, work-groups are run concurrently in the compute units, as a device may not have enough resources to execute them all at once. Work-group size can be defined through the *local work size*. The OpenCL platform consists of a host and a list of compute devices. The host is responsible for managing resources on compute devices. A compute device, such as a CPU, GPU or MIC, has a separate device memory and a list of compute units. By this abstraction, OpenCL enables portable execution.

Computing accelerator devices usually have a differentiated memory address space. For this reason, kernel launches must be preceded by an input data copy phase, from the main memory to the device memory, and followed by another in the opposite direction for the results. For these operations, OpenCL uses the concept of *buffers*, which are a host representation of the memory of the devices in a context. These copy phases must be explicitly instructed by the programmer, which constitutes a tedious and error-prone task.

The accelerator that has been integrated and supported in Maat in this work is the Intel Xeon Phi Knights Corner. It is a coprocessor to speed up parallel applications; therefore, it requires at least one processor in the system. It is a cache-coherent shared memory many-core processor (SMP) based on the x86 architecture. The coprocessor is connected via the PCI express bus to other devices and the host, and they are not hardware cache coherent with other devices in the node. It consists of up to 61 in-order 1.2 GHz x86 cores, capable of running up to 244 hardware threads and with wide vectorisation capability via 512-bit vector registers, connected by a high-performance on-die bidirectional ring interconnect and 8 memory controllers supporting 16 GB of GDDR5 channels at most, with up to 352 GB/s bandwidth.

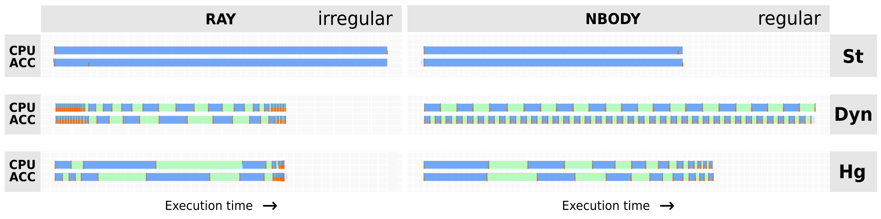

Fig. 1 Ray and NBody package distribution for the three load balancing algorithms

3 Load balancing algorithms

Since there is no load balancing strategy for data parallelism that can improve the execution of all applications, a set of load balancing algorithms is proposed. Figure 1 depicts a comparison among the three load balancing algorithms in real executions. The Y-axis shows every algorithm and the package distribution per device (CPU and ACC), while the X-axis reflects the execution time per benchmark (Ray and NBody). Every rectangle is a work package launched to a specific device. As can be seen, every algorithm balances perfectly the load.

3.1 Static algorithm

This algorithm works before the kernel is executed by dividing the data set in as many *packages* as devices are in the system, as it is shown in *St* in Fig. 1. The division relies on knowing the computing power of the devices in advance. Then, the execution time of each device can be equalised by proportionally dividing the data set among the devices.

Considering a heterogeneous system with n devices. Each device i has *computational power* P_i, which is defined as the amount of work that a device can complete per time unit, including the communication overhead. These powers are parameters that must be given to the algorithm and can be extracted by profiling. Then, the total computational power of the heterogeneous system is the sum of the individual powers of the devices $P_{\text{H}} = \sum_{i=1}^{n} P_i$.

The application will execute a kernel over W work-items, grouped in G work-groups of fixed size $L_s = \frac{W}{G}$. Since the work-groups cannot communicate among themselves, it makes sense to distribute the workload taking the work-group as the atomic unit. Each device will have an execution time of T_i. Then, the execution time of the heterogeneous system will be that of the last device to finish its work, or $T_{\text{H}} = \max_{i=1}^{n} T_i$.

The goal of the Static algorithm is to determine the number of work-groups to assign each device, so that all the devices finish their work at the same time. This means finding a tuple $\{\alpha_1, \ldots \alpha_n\}$, where α_i is the number of work-groups assigned to the device i. Therefore, the expression used by the algorithm is:

$$\alpha_i = \left\lfloor \frac{P_i\, G}{\sum_{j=1}^{n} P_j} \right\rfloor \tag{1}$$

If there is not an exact solution with integers, then $\sum_{i=1}^{n} \alpha_i < G$. In this case, the remaining work-groups are assigned to the most powerful device.

The beauty of the Static algorithm is that it minimises the number of synchronisation points. This makes it perform well when facing regular loads with known computing powers that are stable throughout the data set. However, it is not adaptable, so its performance might not be as good with irregular loads.

3.2 Dynamic algorithm

Some applications do not present a constant load during their executions. To adapt to their irregularities, the Dynamic algorithm divides the data set in small packages of equal size. The number of packages is well above the number of devices in the heterogeneous system. During the execution of the kernel, a master thread in the host is in charge of assigning packages to the different devices, including the CPU, following the next strategy:

1. The master splits the G work-groups in packages, each with the package size specified by the programmer. This number must be a multiple of the work-group size. If the number of work-items is not divisible by the package size, the last package will be smaller.
2. The master launches one package on each device, including the host itself if it is desired.
3. The master waits for the completion of any package.
4. When device i completes the execution of a package:
   (a) The device returns the partial results corresponding to the processed package.
   (b) The master stores the partial results.
   (c) If there are outstanding packages, a new one is launched on device i.
   (d) If all the devices are idle and there are no more packages, the master jumps to step 5.
   (e) The master returns to step 3.
5. The master ends when all the packages have been processed, and the results have been received.

This algorithm adapts to the irregular behaviour of some applications. However, each completed package represents a synchronisation point between the device and the host, where data are exchanged and a new package is launched. This overhead has a noticeable impact on performance. The Dynamic algorithm takes the size of the packages as a parameter. The time to process a package of equal size is the same in regular applications, while it is not in irregular ones, like it is depicted in *Dyn* in Fig. 1.

3.3 HGuided algorithm

The previous strategies have their strong points and their weak spots. Although neither is the best for every application, both give hints towards an optimal data-division algorithm. The Heterogeneous Guided algorithm (*HGuided*) is an attempt to reduce the synchronisation points of the Dynamic while retaining most of its adaptiveness.

The same algorithm used in the Dynamic approach is applicable to the HGuided, except for how the data set is divided. The HGuided algorithm makes larger packages at the beginning and reduces the size of the subsequent ones as the execution progresses. This reduces the number of synchronisation points and the corresponding overhead, while retaining a small package granularity towards the end of the execution to allow all devices to finish simultaneously, as can be seen in *Hg* in Fig. 1.

Since it is an algorithm for heterogeneous systems, the size of the packets is also dependent on the computing power of the devices. The size of the package for device i is calculated as follows:

$$packet\_size_i = \left\lfloor \frac{G_r\ P_i}{k\ n\ \sum_{j=1}^{n} P_j} \right\rfloor \tag{2}$$

where k is a constant, between 2 and 3, and the smaller the constant, the faster decreases the packet size. Tweaking this constant prevents too large packet sizes when there are only a few devices, with cases such as giving half the workload in the first packet to a device, unbalancing the load. G_r is the number of pending work-groups and is updated with every package launch. The parameters of the HGuided are the computing powers and the minimum package size of the devices to be used. The minimum package size is a lower bound for the $packet\_size_i$, and the minimum package sizes are usually dependent on the computing power of the devices, being bigger package sizes in the most powerful devices.

4 Design and implementation

Maat is a library that acts as an OpenCL wrapper to simplify the programming of heterogeneous devices and squeeze the performance out of them. Maat is specially designed to be used in large data-parallel applications, and it provides the three load balancing algorithms presented previously. While OpenCL forces the programmer to consider the devices as individual entities (Fig. 2 step 1), Maat defines an abstraction layer over all the accelerator devices in a machine. It presents a single virtual device to operate with, hiding the underlying hardware details. Thus, the library effectively divides a single task among all the real devices based on the load balancing algorithm selected by the programmer.

This single virtual device is accessed through a *super context* (Fig. 2 step 2). It is created by the programmer specifying the target devices. In contrast to OpenCL contexts, a *super context* can hold devices from several different manufacturers. Maat offers a set of functions that resemble typical OpenCL calls, to manage the *super context*. The *super context* transparently manages the data structures of all the target devices, like the command queues.

While in OpenCL the programmer will need to allocate many buffers to communicate with the different target devices, Maat simplifies this task with the *super buffer* (Fig. 2 step 5). When one of these is created through the *super context*, the latter transparently allocates the required buffers on each device. If the data will only flow from

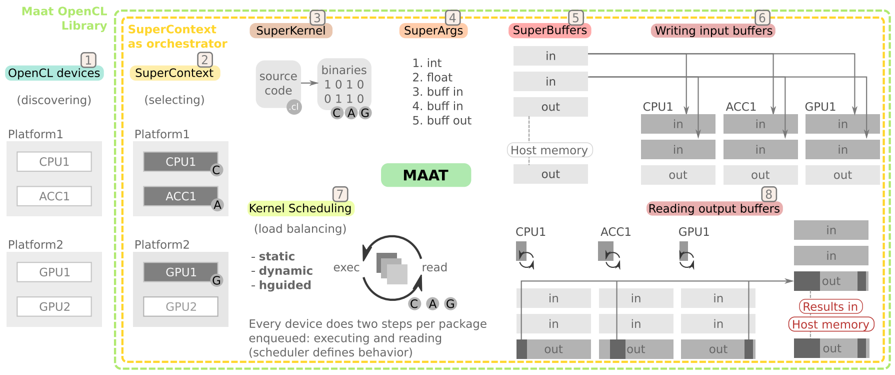

Fig. 2 Maat library from a programmer point of view

the host to the device, it is considered an *in super buffer* (Fig. 2 step 6). Otherwise it is considered an *out super buffer*.

The common reading and writing procedure in *super buffers* assumes that each work-item will use the position indicated by its index. The copy is not performed until the kernel is launched in the device. Such behaviour is necessary in Dynamic and HGuided algorithms where there is no way of knowing in advance which device will compute what package of data. The *out super buffer* (Fig. 2 step 8) creation function requires two extra parameters to be able to automatically copy back the results from the device to the host: a pointer to where the results should be stored in the host memory and the size of the result obtained by each work-group. Therefore, the requirement of writing buffers based on the work-item index denotes the type of applications supported. This may seem a strong requirement, but not only common benchmarks but also many kernels widely used in the industry meet this condition.

Maat simplifies the kernel configuration and execution process by using the idea of *super kernel* (Fig. 2 step 3). When such an entity is created, the *super context* transparently sets up a different kernel instance for each device. Similarly, parameter assignment (Fig. 2 step 4) to the *super kernel* is forwarded to all the kernels in the *super context*. The *super kernel* receives exactly the same global work size, local work size and global work offset it would receive if working with a single device, achieving an easy portability of common OpenCL applications to Maat.

When a *super kernel* is enqueued, Maat transparently performs as many executions of individual kernels as required by the selected load balancing algorithm (Fig. 2 step 7). Each execution will use the adequate OpenCL parameters to represent the correct package of work. The on-demand launch of additional packages in the Dynamic and HGuided algorithms has been implemented using OpenCL callbacks.

The callbacks have been adapted to perform non-blocking readings. Most of the logic has been transferred to functions managed by an independent thread. This increases the performance because it simplifies the callbacks and the synchronisations are deadlock-free (host functions are triggered using *pthread_cond_wait*).

Finally, a function that waits for the completion of every callback guarantees the correctness of the non-blocking calls and improves the performance due to the max-

Table 1 Parameters for each benchmark, local work size (LWS), ACC-CPU execution time ratio used in Static and HGuided, minimum package size for ACC and CPU in HGuided (MinSize) and number of packages in Dynamic (NumPkg)

Bench	Type	Problem size	LWS	Time acc/cpu	MinSize acc, cpu	NumPkg
Binomial	Reg.	40,960,000 samples, 255 steps	256	2.61	103, 40	80
Gaussian	Reg.	13,000×13,000px, 101×101px filter	128	3.92	98, 24	80
NBody	Reg.	1,536,000 bodies	128	4.50	39, 8	96
Taylor	Reg.	1100×1100px	128	1.63	110, 24	96
Mandelbrot	Irreg.	30720×30720px, 6144 rounds	256	4.68	104, 64	96
Rap	Irreg.	2048×2048px, 2,328,576	64	7.16	29, 4	96
Ray1	Irreg.	12,000×12,000px, 3 lights, 10 objs	64	0.60	15, 40	96
Ray2	Irreg.	5000×5000px, 11 lights, 30 objs	64	1.27	51, 40	48

imum overlap between computation with communication. This waiting call serves as a common interface between the load balancing algorithms to monitor every performed callback, follow its event state and wait for its completion, simplifying the API between the algorithms.

Another API simplification and improvement performed in Maat is the automatic selection of the minimum package size for the Phi by giving just the value for the CPU. The best value is proportional to the computing power ratio between Phi/CPU, as shown in Table 1.

The power consumption of the devices is measured by an independent thread. A timer awakes the thread based on the configured sampling interval using a `SIGALARM` *POSIX's* signal as notification mechanism. The timer is tightly coupled to the lifetime of the *super context*. The thread measures the power and energy consumption of the devices used using vendor-specific APIs, calculates the intervals between measures and stores the results in memory to be queried at the end.

5 Methodology

The machine on which the experimentation was carried out has two processor chips and one Intel Xeon Phi. The CPUs are Intel Sandy Bridge Xeon E5-2620, with six cores that can run two threads each at 2.0 GHz and 16 GB of DDR3 main memory. The CPUs are connected via QPI, which allows OpenCL to detect them as a single device; therefore, any reference to the CPU includes both processors. The Intel Xeon Phi coprocessor is a Knights Corner 7120P which consists of up to 61 cores connected by a high-performance on-die bidirectional ring interconnect and 8 memory controllers supporting 16 GB of GDDR5 channels at most, with up to 352 GB/s bandwidth.

Seven applications have been chosen for the experiments. Two are in-house implementations of well-known algorithms (Gaussian and Taylor), while the rest are part of the AMD APP SDK [2]. While Binomial, Gaussian, NBody and Taylor are regular applications, Mandelbrot, Rap and Ray Tracing are irregular. Two different scenes with different complexities (resolution, lights and objects), referred as Ray1 and Ray2, have been considered.

The parameters for each of the applications are shown in Table 1. Local work size has been set so the performance of the fastest device is maximised. The reason for this is that almost no performance difference was detected when varying local work size for the CPU.

The metric used to evaluate the performance of the algorithms is the total response time, including the input data and results communications. From that, the speedup is calculated as the ratio between the execution time on the Phi and on the heterogeneous system. Due to the heterogeneity of the system and the different behaviour of the benchmarks, the maximum achievable speedups depend on each benchmark. These values were derived from the response time T_i of each device:

$$S_{\text{max}} = \frac{1}{\max_{i=1}^{n}\{T_i\}} \sum_{i=1}^{n} T_i \tag{3}$$

Additionally, the efficiency of the heterogeneous system has been computed as the ratio between the maximum achievable speedup and the empirically obtained speedup for each benchmark. $\text{Eff} = \frac{S_{\text{real}}}{S_{\text{max}}}$. Also, the energy consumption has been calculated for each algorithm normalised to the baseline consumption. This gives an idea of the energy saving that comes through the usage of the whole heterogeneous system, instead of the baseline system that only uses the ACC, while the other devices are idle.

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) [4] for this purpose.

The computational power needed for the Static and HGuided algorithms has been computed for each benchmark. The response times of the benchmark have been measured in both the CPU and Phi. Then, considering the computational power of the CPU equal to 1, the computational power of the Phi is calculated, as the ratio between the CPU time and the Phi time: $P_{\text{Phi}} = \frac{T_{\text{CPU}}}{T_{\text{Phi}}}$.

6 Evaluation

This section presents the performance results and energy savings achieved in the heterogeneous system with different load balancing algorithms. The baseline is the total response time (or energy consumption) of the benchmarks with only one Xeon Phi. Therefore, the benefits presented in this section are due to the co-execution of the benchmarks on the CPU and the Phi simultaneously.

To give an idea of performance of the load balancing algorithms, Fig. 3 shows the speedups reached by the Maat implementation of the benchmarks, compared with the baseline. The theoretical maximum speedup that can be obtained with each benchmark is shown as a horizontal line above the bars of each benchmark. The figures reveal that, for all benchmarks, there is at least one algorithm that gives excellent results close to the maximum, except for Ray1, where the efficiency is up to 0.85. Therefore, Maat can adapt to different kinds of loads obtaining outstanding performance.

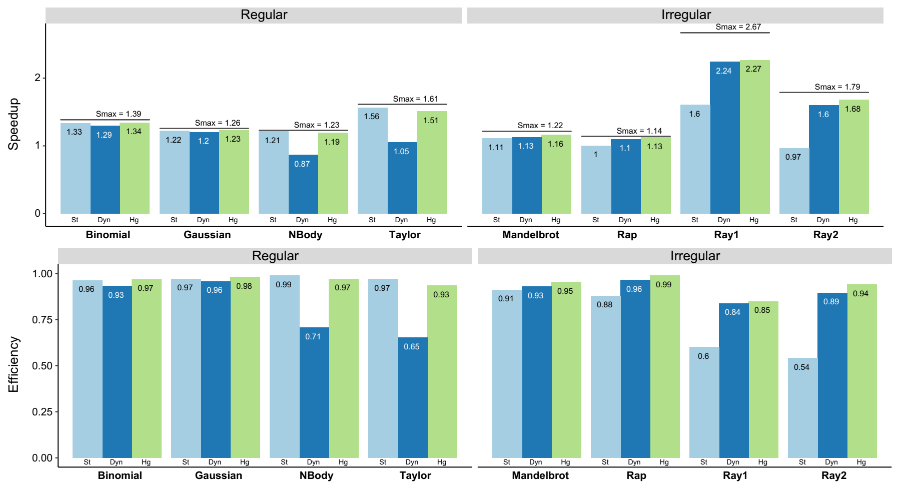

Fig. 3 Speedups and efficiency of the heterogeneous system (more is better)

Analysing the speedups in detail, it can be seen that Static and HGuided deliver excellent results in regular applications. The Dynamic algorithm achieves decent results in most of the applications, but suffers the overhead of communication in benchmarks like NBody and Taylor.

On the other hand, the analysis of the irregular benchmarks shows that the HGuided method obtains the best results, followed nearly by the Dynamic. This method achieves a better load balance and reduces the communication overhead, because it divides the workload between a smaller number of packets than in the Dynamic algorithm. The Static is the worst algorithm for irregular applications because it is not adaptive, and benchmarks purely irregular like Ray suffer the most. Coming up with a balanced work distribution is significantly harder for Ray due to the complexity of interaction between the input scene and the ray tracing algorithm.

There is almost no difference between Dynamic and HGuided because the former has been tweaked with near-optimum number of packages, having a small overhead of communication. The Dynamic algorithm is sensible to this parameter, and if the programmer does not choose the number of packages well, the results get worse quickly. This can be seen in NBody and Taylor. These benchmarks have not enough computation compared with the transference of data; therefore, the communication overhead affects more, given sometimes even worse results than only using the Phi. Nevertheless, the HGuided is robust to the computing power given, and even with disparate values the algorithm converges to good results.

Finally, the load balancing efficiency gives an idea of how well a load is balanced. A value of 1.0 represents that all the devices have been working all the time, thus achieving the maximum obtainable speedup. As shown in Fig. 3, the regular applications reach at least 0.97 of efficiency, while the irregular ones achieve an efficiency between 0.85 and 0.99.

However, it should be noted that if the correct load balancing algorithm is not used for each benchmark, the results can be quite bad. Thus, we can see efficiencies around 0.50 in some cases, which imply a very poor use of resources.

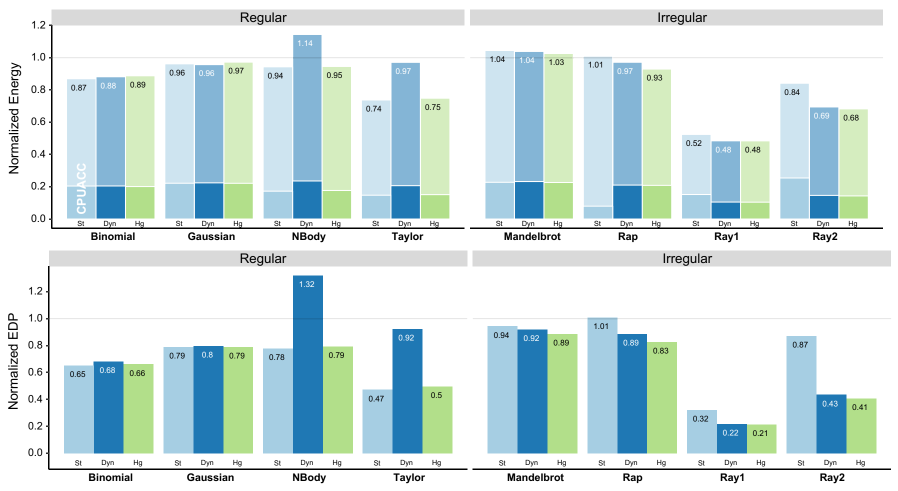

Fig. 4 Normalised energy and EDP of the heterogeneous system (less is better)

Nowadays, performance is not the only figure of merit used to evaluate computing systems. Their energy consumption and efficiency are also very important. Figure 4 gives an idea of the energy saving that comes through the usage of the whole heterogeneous system, instead of the baseline system. Therefore, the figure shows, for each benchmark, the energy consumption of each algorithm normalised to the baseline consumption (less is better.)

Analysing the results, there is a strong correlation between performance and energy saving. Consequently, the best algorithms for regular applications are the Static and HGuided, saving an average of 0.05 and up to 0.25 in benchmarks like Taylor. The saving with irregular applications can be even more further in cases like Ray, being up to 0.50. This is to the high adaptability of HGuided and Dynamic to irregular loads.

Although the Dynamic power consumed at an instant of time is greater in the heterogeneous system, because more devices are used, this increase is compensated by the reduction in execution time. Therefore, a remarkable energy saving is achieved with respect to using the baseline. However, there are types of computations, such as Mandelbrot, with efficiencies between 0.91 and 0.95 but consuming more energy, regardless of whether it runs in less time. On the other hand, the case of NBody with Dynamic occurs because the communication overhead makes it take longer than in the baseline, leading to a higher energy expenditure as well.

Another interesting metric is the energy efficiency, which combines performance with consumption. The EDP is the product of the consumed energy and the execution time of the application. Figure 4 shows the EDP of the algorithms normalised to the EDP of the baseline. Since the EDP is a combination of the two above metrics, the relative advantage of the different algorithms is maintained. Both the Static and HGuided algorithm on regular applications and the HGuided and Dynamic on irregular sensibly reduce the EDP, in relation to the baseline. This leads to an improvement of the energy efficiency of 1.34× for the Static and HGuided in regular applications and 1.53× for HGuided and Dynamic for irregular ones. Considering all the benchmarks, the average improvement observed is 1.43×, if the best algorithm for each benchmark is selected.

7 Related work

This topic has given rise to an interest in both understanding how to efficiently use this kind of devices and making their programming easier. The work presented in [9] and [17] studies the performance of the Xeon Phi through the use offload directive-based approaches in a single heterogeneous node, while this work uses Static and Dynamic approaches with OpenCL. Lastovetsky et al. [6] focus on improving the performance of an application running on a Xeon Phi, but they propose a load imbalancing-based optimisation technique. To the problem of cooperatively executing a single task among the available devices, Zhang et al. [18] proposes an analytical model to identify when co-execution is worth it in terms of performance. Finally, [11] presents a Dynamic load balancing for the co-execution of a single data-parallel kernel. However, both do not consider the Xeon Phi.

Some studies [13,16] analyse many accelerators including the Xeon Phi with different programming models, but not the OpenCL. Notable studies like [3] and [15] are centred on the workload distribution among CPU and GPU. Belviranli et al. provide an online 2-phase algorithm; the adaptive phase calculates the block size, while the completion phase process the most workload with the parameters obtained from the first phase. Its work is centred on CUDA, while we simplify the programming effort and promote portability by using OpenCL. Also, we provide an energy analysis of the load balancing algorithms for multiple benchmarks. Vilches et al. propose two load balancing algorithms in *TBB* for regular and irregular applications when CPU-integrated GPUs are co-executed, while we exploit OpenCL and the Xeon Phi.

Related to the programmability, there are some works [1,8]. Li et al. propose a new programming tool along with a new domain-specific language designed to simplify the development of an application for multiple accelerators but uses a fixed algorithm focused on stencil applications , while Maat allows the programmer to choose three different load balancing algorithms for data-parallel applications and it uses pure C API function calls that follow the same conventions as the OpenCL standard. Aji et al. address the problem of load balancing while abstracting the underlying system, but they focus on task parallelism instead of data parallelism. Related to the Static approach, in [7] the focus is on the distribution of a single kernel execution to the devices, but use code modifications. Zhong et al. [19] use performance models. Both works are not adaptable, and Maat has a simpler API.

Regarding power, GreenGPU dynamically distributes work to GPU and CPU, minimising the energy wasted on idling and waiting for the slower device [10]. SPARTA is a throughput-aware runtime task allocator for heterogeneous many-core platforms [5]. Both studies are not centred in the Xeon Phi.

All the above mentioned works are just partial solutions to the problem of performance portability, which includes load balancing and system abstraction. They all fail to address the importance of irregular applications or loads that are limited in the number or type of devices that they can manage. On the contrary, Maat does not suffer from these constraints and constitutes a performance portable solution.

8 Conclusions and future work

This paper extends the Maat library to allow the co-execution of massively data-parallel OpenCL kernels on heterogeneous systems composed by an Intel Xeon Phi coprocessor. A set of load balancing algorithms has been implemented, in order to give the best performance and energy efficiency to both regular and irregular applications.

From the experimental results presented in this paper, a set of conclusions can be remarked. The use of the whole heterogeneous system is always beneficial, from the performance point of view, at least with one of the load balancing methods. Regarding savings in energy consumption and energy efficiency, generally is advised, but in a few specific cases should be discarded. A second conclusion is that applications with different behaviours, regular or irregular ones, need different load balancing algorithms to get the best efficiency on the heterogeneous system. With respect to the algorithms, the Static and HGuided approach is the most adequate for regular applications as it minimises overheads. In the case of irregular applications, the HGuided and Dynamic method excels because of their adaptiveness, although the HGuided is robust independently of the “regularity” of an application. Lastly, the Dynamic algorithm is an acceptable all-around option when a-priori information of the computing powers of the system is not available, but at the cost of more modest speedups and suffering penalties in some applications.

Future work includes new load balancing algorithms, a completely rewrite of the library to provide a new architecture to support new load balancing algorithms and tweak the implementation design to reduce synchronisation and communication overhead. Also, new abstractions would be included to improve the programmability while preserving performance portability.

Acknowledgements This work has been supported by the Spanish Ministry of Education, FPU grant FPU16/03299, the University of Cantabria, grant CVE-2014-18166, the Spanish Science and Technology Commission under contracts TIN2016-76635-C2-2-R and TIN2016-81840-REDT (CAPAP-H6 network), the European Research Council (G.A. No. 321253) and the European HiPEAC Network of Excellence. The Mont-Blanc project has received funding from the European Unions Horizon 2020 research and innovation programme under Grant Agreement No. 671697.

References

1. Aji AM et al (2016) MultiCL: enabling automatic scheduling for task-parallel workloads in OpenCL. Parallel Comput 58:37–55
2. AMD Accelerated Parallel Processing (APP) Software Development Kit (SDK) V3. Last accessed January 2018. https://developer.amd.com/amd-accelerated-parallel-processing-app-sdk/
3. Belviranli ME, Bhuyan LN, Gupta R (2013) A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures. ACM Trans Archit Code Optim 9(4):1–20
4. Castillo E et al (2014) Financial applications on multi-CPU and multi-GPU architectures. J Supercomput 71(2):729–739

5. Donyanavard B, Mück T, Sarma S, Dutt N (2016) SPARTA: runtime task allocation for energy efficient heterogeneous many-cores bryan. In: Proceedings of the 11th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp 1–10
6. Lastovetsky A, Szustak L, Wyrzykowski R (2017) Model-based optimization of eulag kernel on intel xeon phi through load imbalancing. IEEE Trans Parallel Distrib Syst 28(3):787–797
7. Lee J, Samadi M, Park Y, Mahlke S (2015) Skmd. ACM Trans Comput Syst 33(3):1–27
8. Li P, Brunet E, Trahay F, Parrot C, Thomas G, Namyst R (2015) Automatic OpenCL code generation for multi-device heterogeneous architectures. In: Proceedings of the International Conference on Parallel Processing, pp 959–968
9. Lopez et al (2016) Towards achieving performance portability using directives for accelerators. In: Third workshop on accelerator programming using directives, pp 13–24
10. Ma K, Li X, Chen W, Zhang C, Wang X (2012) GreenGPU: a holistic approach to energy efficiency in GPU-CPU heterogeneous architectures. In: Proceedings of the International Conference on Parallel Processing, pp 48–57
11. Pandit P, Govindarajan R (2014) Fluidic kernels: cooperative execution of opencl programs on multiple heterogeneous devices. In: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization, pp 273–283
12. Pérez B, Bosque JL, Beivide R (2016) Simplifying programming and load balancing of data parallel applications on heterogeneous systems. In: Proceedings of the 9th Annual Workshop on General Purpose Processing using Graphics Processing Unit, ACM, pp 42–51
13. Salehian S, Liu J, Yan Y (2017) Comparison of threading programming models. In: Proceedings IEEE 31st International Parallel and Distributed Processing Sym. Workshops, pp 766–774
14. Stone JE, Gohara D, Shi G (2010) OpenCL: a parallel programming standard for heterogeneous computing systems. IEEE Des Test 12(3):66–73
15. Vilches A, Asenjo R, Navarro A, Corbera F, Gran R, Garzarán M (2015) Adaptive partitioning for irregular applications on heterogeneous CPU–GPU chips. Procedia Comput Sci 51(1):140–149
16. Wienke S, Terboven C, An Mey D, Muller MS (2013) Accelerators, quo vadis? Performance vs. productivity. In: Proceedings of the International Conference on High Performance Computing and Simulation, pp 471–473
17. Xiao X, Hirasawa S, Takizawa H, Kobayashi H (2016) The importance of dynamic load balancing among openmp thread teams for irregular workloads. In: 4th International Symposium on Computing and Networking, pp 529–535
18. Zhang F, Zhai J, He B, Zhang S, Chen W (2017) Understanding co-running behaviors on integrated cpu/gpu architectures. IEEE Trans Parallel Distrib Syst 28(3):905–918
19. Zhong Z, Rychkov V, Lastovetsky A (2015) Data partitioning on multicore and multi-GPU platforms using functional performance models. IEEE Trans Comput 64(9):2506–2518