Energy Efficient Parallel K-Means Clustering for an Intel® Hybrid Multi-Chip Package

Matheus A. Souza*, Lucas A. Maciel*, Pedro Henrique Penna*†, Henrique C. Freitas*
*Pontifical Catholic University of Minas Gerais (PUC Minas), Belo Horizonte, Brazil
†Université Grenoble Alpes (UGA), Grenoble, France
{matheus.alcantara,lamaciel,pedro.penna}@sga.pucminas.br, cota@pucminas.br

***Abstract*—FPGA devices have been proving to be good candidates to accelerate applications from different research topics. For instance, machine learning applications such as K-Means clustering usually relies on large amount of data to be processed, and, despite the performance offered by other architectures, FPGAs can offer better energy efficiency. With that in mind, Intel® has launched a platform that integrates a multicore and an FPGA in the same package, enabling low latency and coherent fine-grained data offload. In this paper, we present a parallel implementation of the K-Means clustering algorithm, for this novel platform, using OpenCL language, and compared it against other platforms. We found that the CPU+FPGA platform was more energy efficient than the CPU-only approach from 70.71% to 85.92%, with Standard and Tiny input sizes respectively, and up to 68.21% of performance improvement was obtained with Tiny input size. Furthermore, it was up to 7.2× more energy efficient than an Intel® Xeon Phi™, 21.5× than a cluster of Raspberry Pi boards, and 3.8× than the low-power MPPA-256 architecture, when the Standard input size was used.**

***Index Terms*—K-Means, OpenCL, FPGA, energy efficiency.**

I. Introduction

In order to address the ever-increasing demands for High Performance Computing (HPC), the Computer Architecture community introduced a great variety of massively parallel architectures, such as large-scale multicores, low-power NoC-based manycore processors and heterogeneous architectures. While in the past decade, the former solution was widely exploited, it soon got refutable as a scalable solution, due to restrictions in their interconnection systems. Large-scale multicores are inherently power-hungry and do not deliver a good balance between performance and power consumption, which is primordial to achieve exascale [1], [2]. On the other hand, low-power manycores and heterogeneous architectures may deliver better energy-efficiency and thus are emerging as promising alternatives.

Graphics Processing Units (GPUs) and the accelerators such as Xeon Phi™ have been used in current platforms, since they can significantly accelerate overall performance, although power consumption remained an open-challenge. These architectures can be used in a heterogeneous fashion, introducing new concerns, for instance, in the way the data is exchanged between the host processor and the device. Usually, such accelerators are connected to the host using interfaces such as the Peripheral Component Interconnect Express (PCIe) interface. These devices have their own memories, where copies of host data are put to be processed. Thus, there might be communication problems, which impair performance and increase energy consumption. Also, when specific domains that deal with big data volumes are considered (e.g., Machine Learning), the problem gets worse.

A good alternative for high performance in big data processing is the use of Field Programmable Gate Arrays (FPGAs). As an example, the K-Means clustering is an algorithm in big data domain that have been implemented in different ways for this kind of architecture [3], [4]. FPGA is a device which has a flexible hardware that can be adapted to perform specific computational tasks. This means that the processing is done at hardware level, without general processing units, which might lead to better performance and lower power consumption than other architectures. Current FPGAs can offer about 1/10 of GPUs' power consumption at near or better performance levels [5]. Thus, they are good candidates for big data processing, as well as to reach the exascale era, providing better scalability than GPUs and CPUs when a higher number of devices are used [6].

Despite the advantages, low bandwidth is still a problem in traditional FPGA devices. To avoid this, multi-chip packaging (MCP) technology can be employed. The purpose of such technology is to integrate multiple heterogeneous devices or chips in a single board, approximating them to address performance, communication bandwidth, power consumption, chip area costs, among other constraints [7]. Thinking of offering a device with these characteristics, Intel® has launched a hybrid platform, which integrates an Intel® Xeon® Broadwell processor and an Intel® Arria 10 FPGA, all on a multi-chip package, connected by two PCIe and one Intel® QuickPath Interconnect (QPI). This device is capable to offer low latency and coherent data offload in a fine-grained fashion, improving the use of the FPGA. It has been used in different areas, such as neural network processing, compression and collision detection [8]–[10], presenting good results.

Getting back to big data, Machine Learning is a hot research topic that is increasingly gaining attention from the scientific community. Algorithms of this domain, also referred as data mining methods, rely on a set of knowledge discovery methods for uncovering non trivial information from their input data. For instance, Deep Neural Networks (DNNs) are being used to recognize complex patterns; Q-Learning is being explored to enable automated learning; and the already mentioned

K-Means is being employed on cluster analysis [11]–[13]. Notwithstanding, with the ever growing amount of data to be processed, data mining algorithms are constantly requiring better performance from computing platforms.

Therefore, hybrid MCPs, such as the Intel® CPU+FPGA, can potentially accelerate these types of applications. In this context, the main goal of this paper is to propose and evaluate an energy efficient parallel K-Means algorithm for the Intel® Xeon® Broadwell + Arria 10 FPGA. In summary, this work delivers the following new contributions to the state-of-the-art:

- A parallel K-Means proposal using OpenCL focusing on a hybrid MCP platform with FPGA acceleration.
- A demonstration of an iterative development process in which we highlight some aspects that have to be taken into account when developing for the hybrid MCP platform.
- A comparative analysis against other platforms which unveils that the CPU+FPGA is a considerable energy efficient approach which also offers high performance.

The remainder of the paper is organized as follows. In Section II, we explain how K-Means works and present the target platform. In Section III, we present related work. In Section IV, we detail our K-Means implementation. In Section V, we present our methodology, and in Section VI we discuss the experimental results. Finally, in Section VII, we present our final considerations.

II. Background

In this section we present the K-Means algorithm and details of the target CPU+FPGA platform.

A. K-Means Clustering

The K-Means Clustering is an unsupervised machine learning algorithm that is widely employed in data mining to partition and group data according to their features [14], [15]. Algorithm 1 outlines how this method works. Given a set of n points in a real d-dimensional space, the algorithm partitions these n points into k clusters so as to minimize overall mean squared distance from each point to its nearest centroid.

Algorithm 1 K-Means Clustering

```
1: procedure KMEANS(k, points, d)
2:     centroids ← random_centroids(k, points, d)
3:     repeat
4:         map ← compute_distances(points,centroids, d)
5:         recalculate_centroids(points,centroids,map,d)
6:     until not check_if_should_stop()
7:     return map
8: end procedure
```

The algorithm starts by randomly assigning data points to each of the k clusters. Next, (1) the centroid of each one of the k clusters is computed; and then (2) data points are reassigned to the cluster whose centroids is the closest according to the Euclidean Distance [16]. Steps 1 and 2 are repeated until all centroids do not change, or until a preset threshold is reached. Figure 1 shows an example of a hypothetical K-Means clustering process, with four centroids and their associated points. As a baseline to perform our experiments, we considered a parallel version from the K-Means algorithm, which is implemented in the CAP Bench suite [17] using the OpenMP library.

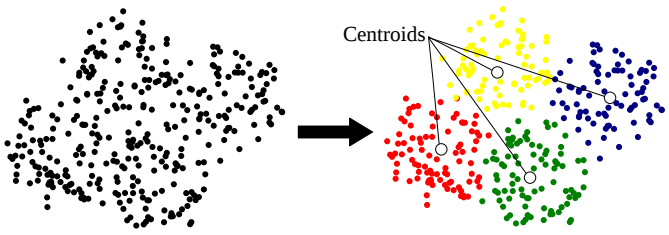

Fig. 1. K-Means clustering example.

B. The CPU-FPGA platform

Recently, Intel® launched a novel heterogeneous platform that enabled the design of new solutions for fine-grained communication-intensive applications [8]. This architecture features an Intel® Xeon® processor (host) with an FPGA (device) in the same package. The host and device are connector through a dedicated a high-speed interconnection.

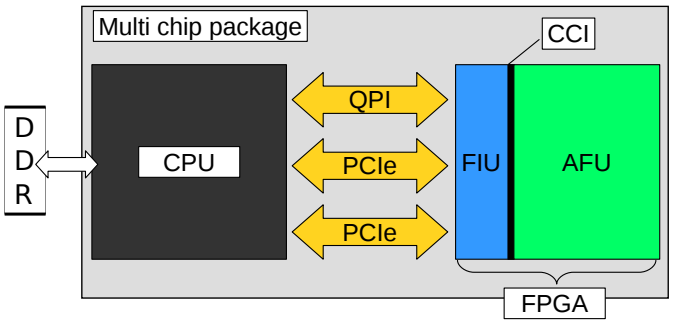

Fig. 2. Intel® Xeon® + FPGA platform overview.

Figure 2 shows an overview of the new Intel's architecture. The platform we used in this work consists of a 14-core Intel® Xeon®, at 2.4GHz (Broadwell), connected to an Intel® FPGA Arria 10, model GX1150, by three interfaces: two PCIe interfaces which offer a theoretical maximum bandwidth of 16 GB/s together, and one Intel® QuickPath Interconnect (QPI) which can provide up to 12.8 GB/s throughput. The FPGA is divided into two sections. The first one is the FPGA Interface Unit (FIU), provided by Intel®, which contains the bit stream responsible for make the interface between the mentioned interconnections and the user-provided bit stream. The second part is the user-provided bit stream, called Accelerated Function Unit (AFU). Between those two sections, there is an interface responsible for expose the channels of communication to the user (CCI). If the QPI is used, the communication is done in a coherent fashion (between the host and the FPGA memories).

In this platform, the CPU and FPGA shares the DDR memory, which improves the acceleration of algorithms, and simplifies the application development. This is an advantage of MCP strategy, since off-chip memory accesses are much faster than on traditional CPU and FPGA systems. With respect to the development for the platform, it is possible to use High Level Synthesis (HLS) or hardware description languages, (e.g., Verilog and VHDL). The Intel® SDK for OpenCL is a development kit available to implement kernel functions in OpenCL, compile and execute them on the FPGA.

A specific abstraction layer for the hybrid MCP platform is also provided, in order to initiate the device, compile the kernel code, among other features. The data exchange between the host and the device is specified by parameters in the kernel. Thus, the programmer creates the OpenCL kernel, specifying inputs and outputs (parameters), and the programming logic itself. The compiler produces bit streams which are loaded into the FPGA AFU, to be called by the host part of the code. At the end of the application, other calls and functions from the host unloads internal modules of the multi-chip package system.

III. Related work

In this section, we briefly discuss current research efforts that are related to our work. Machine Learning has recently become a hot research area due to its wide applicability, and thus several works currently focus on accelerating computational kernels on this domain. For instance, concerning FPGA-based solutions, some authors studied existing challenges to process DNNs in FPGAs [18], and they found out that these were related to memory restrictions, such as, bandwidth and reduced memory addressing space. Based on this observation, they proposed the use of stream buffers to better exploit locality to minimize data exchange between the FPGA and the host. Likewise, a framework based on OpenCL for large-scale DNNs was proposed [19]. In this work, the authors proposed to use a pipeline of kernels to reduce issues related to memory bandwidth, increasing on-chip data reuse. To improve external memory access, the authors of [20] proposed a strategy to reduce floating point precision on FPGA operations, despite a small loss of accuracy. These papers mentioned at the outset demonstrates how important is to be worried with FPGA implementations when implementing big data applications. Also, they reinforces one of the proposals of the hybrid MCP platform, which is to have high communication capabilities between the host and the device.

Specifically about the K-Means clustering, prior works were also carried out, not only with FPGA platforms. For instance, a Raspberry Pi Cluster was employed to run K-Means [21]. An Intel® Xeon Phi™ was used as baseline for comparison, and results uncovered that an eight-node quad-core Raspberry Pi cluster presented 85.17% lower energy consumption than the former platform. Alternative works also investigate the use of standalone FPGAs to address the problem.

Also, a hardware of K-Means was designed and implemented [4]. The hardware had some level of parallelism when processing data features. When comparing it against a Intel® Xeon® E5-2620 processor, it was possible to achieve gains between 91% and 98% in the number of processing cycles, which led to 95% less energy consumption. Both works presented constraints regarding the communication bandwidth. In the former, a local area network restricted the bandwidth to 10 GBits/s, and in the latter, the hardware design itself.

A parallel and parameterized hardware design of K-Means was proposed [22], in which the number of parallel units to calculate the Manhattan distance and the number of accelerating cores can be modified. The architecture relies on 32-bit data and the authors compared their results against a software implementation. When varying the number of cores, number of points, dimensions and centroids, the implementation in the ZYNQ 7045 FPGA reached about 105 Gflops at best, achieving a performance efficiency of 99%, thus, presenting a speedup of $10\times$ in relation to the software version.

There is also a work in which a parallel K-Means on CPU was implemented [23]. The authors developed it using OpenMP and Basic Linear Algebra Subroutines (BLAS), as well as in GPU, using CUDA. The authors employed what they called 'kernelization' in the data, transforming them into a new vector representation, solving some problems related to non-linearly separable data. Their evaluation used different datasets, varying the number of clusters, features and patterns, demonstrating that OpenMP and CUDA implementations were, respectively, $2\times$ and $6\times$ better than the sequential approach.

Other authors analyzed a K-Means implementation on heterogeneous platforms with a Symmetric Multiprocessor (SMP) and a GPU or an FPGA [24]. The goal was to use the OmpSs programming model, which allowed to simplify communication between the host processor and the accelerators, in a transparent manner for the programmer. The results showed that the system formed by SMP and FPGA consumed less energy than the one with SMP and GPU, but the latter stood out in terms of speedup.

An approach for data science programmers was carried out using the Intel® FPGA SDK for OpenCL [3]. The proposal was to implement the K-Means algorithm, running it on a Stratix V A7 FPGA, comparing its performance against a six-core Intel® Xeon® W3670, and its power consumption against the GPU GTX280. Experimental results unveiled that it is possible to achieve from $3\times$ to $21\times$ better performance with the FPGA, while being $29\times$ more energy efficient than the GPU approach.

A hardware and software architecture called MUCH-SWIFT was proposed [25], based on the parallelization of K-Means, using a ZYNQ Ultrascale + FPGA board, with several hard-core processors and a ZU9EG FPGA. The authors performed several runs, varying the number of clusters and iterations. They obtained a speedup of $330\times$ in relation to a K-Means version in software.

Our work differs from the aforementioned ones since it is

a parallel K-Means proposal for an integrated heterogeneous MCP platform, with a parallel CPU and a powerful FPGA. We evaluated the performance of K-Means on an Intel® Xeon® CPU and Arria 10 FPGA platform, which offers better peak bandwidth than traditional FPGA devices, thereby improving performance. Furthermore, our K-Means implementation offers better Flops per Watt than other state-of-the-art platforms, as we discuss in Section VI-D.

IV. Proposed Parallel K-Means for the Hybrid CPU+FPGA Platform

In this Section, we present our strategy to implement and accelerate K-Means using the Intel® Hhybrid MCP platform. We implemented our K-Means code based on an OpenMP version from CAP Bench [17], which is a benchmark suite for low-power manycore processors. Furthermore, we consider this code to compare the MCP platform against others. Table I presents the input sizes from K-Means that we considered in our work. Furthermore, we set in 200 iterations the threshold stopping condition for the algorithm. However, this threshold was not reached with the input sizes we considered.

TABLE I
K-Means input sizes

Workload	Nº of data points	Nº of centroids	Nº of features
Tiny	4096	256	16
Small	8192	512	16
Standard	16384	1024	16
Large	32768	1024	16
Huge	65536	1024	16

To accelerate algorithms on the MCP, we can use OpenCL language to implement the application kernel, and C/C++ to implement the host code. The latter is responsible for instantiate and initialize the FPGA device, for allocate host memory, for define the necessary objects to allocate memory at the device, and for synchronization tasks.

OpenCL kernels can be loaded on the FPGA using work items, which are run sections that contains instances from the developed kernel. These work items are executed in parallel, sharing different memory resources in the FPGA, accessing different data elements. This approach is very suitable for FPGAs, due to the fact that the customized hardware can be developed thinking of parallel compute units, even more if the application code snippets has no data dependency (thus, data can be accessed without mandatory synchronization). Other approaches to have parallel computing is to use pipelines and vectorization, which are both possible with the hybrid MCP platform and its OpenCL development kit.

Figure 3 shows a flowchart of how our final parallel implementation works. In the rest of this section, we explain the development decisions we made to reach this implementation. In a prototype implementation, we first chose to compute distances and to recalculate centroids on the FPGA, each step as a separated kernel. Thus, in each iteration, it has been defined OpenCL memory buffers to exchange data between the host and kernels, followed by the procedure to

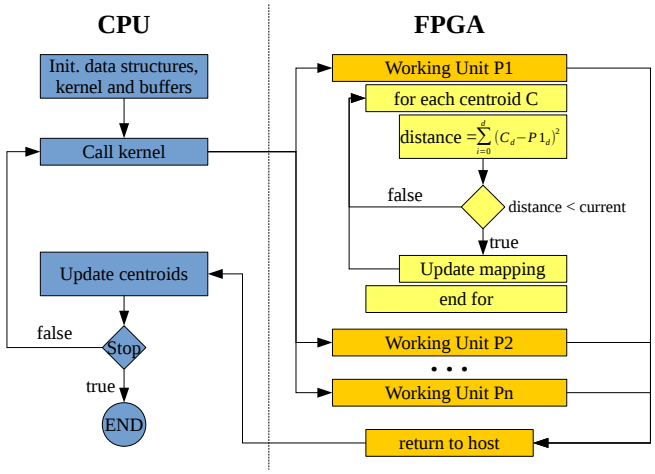

Fig. 3. Proposed parallel K-Means implementation flowchart.

call the kernels. It is worth noting that data which does not change during neither the computation need reallocation.

Those kernels were defined using the *Single Task* approach, which means that a single compute unit was performed at the FPGA. When this strategy is used, the compiler explores kernel loops to infer pipelining, when there is no data dependency. However, we checked that using single tasks was not a good strategy, due to small performance to control the loops, and lower resource usage.

A more interesting strategy was to use the *NDRange Kernel*, which uses the FPGA as multiple compute units, with high parallelism. In this strategy, we need to define how many working units should be used. We defined the number of data points as the number of working units, in which distances will be computed. These values are flexible, since they can be changed at the host. After performing some tests, we saw fit to use the number of data points from input sizes as the number of working units, which are lower than the maximum value accepted by Arria 10.

Nevertheless, the performance was not the desirable yet. To verify this, checked the possible bottlenecks, by measuring how much time each kernel takes, and the host-to-device communication time. The communication time was irrelevant when compared to the computation, due to the high bandwidth that the integrated platform delivers. However, we noted that centroid recalculation was the major bottleneck. There is not much complexity on the centroids compute step, since it computes the average point of a cluster. Furthermore, some conditional branches are used, which are not adequate for the FPGA. Thus, we opted to recalculate centroids also at the host. To achieve better performance, we carried out this computation in the host, using OpenMP. In Section VI we discuss how good this alternative was.

From now on, we decided that this adapted implementation should used and optimized. The first optimization concerns to the distances calculation. In K-Means, to get the assignment from points to centroids, there is no need to use the square

root from the Euclidean distance. This step adds extra complexity for calculate distances, and further decreases the FPGA performance [26]. Since we only need to find the minor distance between features from points and centroids, we remove this step from the calculation. Thus, our implementation can be synthesized in Equation 1.

$$dist(p,c)^2 = \sum_{i=1}^{d} |p_i - c_i|^2 \quad (1)$$

Next, we identified that the loop used to go through the features could be parallelized, since there is no dependency in this loop. Thus, we added the directive *#pragma loop unroll*. The SDK compiler identify this directive, and parallelize the hardware. This way, each feature is compared in parallel, and the result is summarized at the end.

V. Experimental Methodology

In this section, we detail the tools, resources and strategies we used to carry out our experiments. We run our tests 20×, presenting the average results. The standard error was omitted from the plots presented in this work, due to their low values (less than 8.8% on CPU, and almost 0% on FPGA). To build the kernel, we used the Intel® SDK for OpenCL compiler (AOCL), which has Quartus along with it, version 16.0.2. The kernel must be compiled off-line, before its execution, since it it takes to hours to build and synthesize the hardware into bitstreams. The Arria 10 FPGA resource usage was obtained from the compiler. The Broadwell host code was written in C/C++ and compiled with GNU C Compiler (GCC), version 4.9.2. The Intel® MCP platform we have access to is hosted at the Paderborn Center for Parallel Computing (PC2) in the context of the Intel® Hardware Accelerator Research Program (HARP2) [27].

We compared results from the OpenCL K-Means kernel against an OpenMP version from CAP Bench. The OpenMP version was compiled also using GCC 4.9.2, and OpenMP library 4.0. Its execution was done in a system using two Intel® Xeon® E5-2620 processor (Sandy Bridge), with six cores and 12 threads each, totaling 24 threads available. The cores have a private L2 cache with 256 kB each. Those six cores shares an L3 cache of 15 MB.

We measured the times perceived for each computing step separately: distance computing, centroids recalculation, and data off-load. We did not plot off-load time on the chart so as not to disturb the visibility of other time information. For energy efficiency, we compute the total floating point operations of each input size, which is the product from number of points, centroids, dimensions and iterations. The number of iterations observed when the algorithm finished were, for each input size respectively, 13, 15, 14, 25 and 48. Thus, we measured the floating point operations per second (Flops) per Watt consumed.

To get the energy consumption from the FPGA, we get the power consumption, in Watts, from Quartus Power Play. It is important to note that the measured energy concerns only to the distances computing step. This was done to enable a specific understanding of the energy consumption on the FPGA. On the CPU-only version, we measured the energy consumption using the PAPI tool [28].

To further check whether the Intel® MCP platform is more efficient than others, we performed a comparative analysis between the obtained results against other platforms. The first compared platform is the Intel® Xeon Phi™ Knights Corner co-processor [29], a manycore with 61 cores and support for 244 threads (each core has implemented a 4-way simultaneous multithreading). This architecture has a memory system formed by private 32 kB L1 instruction cache, 32 kB L1 data cache and 512 kB L2 cache per core. They are interconnected by a ring bus, which also connects them to the main memory. We also verified an 8-node Raspberry Pi Cluster [30], interconnected by a Fast Ethernet switch. Each node is a Raspberry Pi 2 model B with a quad-core ARMv7 processor, with 64 kB L1 instruction cache, 64 kB L1 data cache, and 512 kB of L2 cache, this one shared among the 4 cores. The last platform was the Kalray MPPA-256 [31], a low-power manycore processor with 16 compute clusters interconnected by two folded-torus networks-on-chip. Each compute cluster has 16 cores, with instruction and data L1 caches of 8 kB, sharing a 2 MB SRAM.

VI. Experimental Results

In this section, we present the FPGA resource usage, time to solution, and energy efficiency results. At the end, we present a comparison between our results and those from other works.

A. FPGA Resource Usage

Table II presents the estimated resource usage obtained after compiling K-Means kernel for the integrated platform.

TABLE II
Estimated resource usage

Resource	Available	FIU + CCI	AFU
Logic utilization	1150	49% (563)	10% (115)
ALUTs	427200	23% (98256)	4% (17088)
Registers	1708800	27% (461376)	6% (102528)
Memory blocks	67244	23% (15466)	12% (8069)
DSP blocks	1518	12% (182)	5% (76)

A considerable part of FPGA resources is used by FIU and CCI interfaces. The amount of each type of resources used demonstrates that Arria 10 supports well our K-Means kernel. Certainly, if more complex computational units were implemented, such as the square root operation, these values would increase. Since all FPGA logic element resources were not used, there is still space left for deploying other kernels.

B. Time to Solution

Figure 4 shows the time observed when K-Means was run using the CPU-FPGA MCP plataform (Broadwell + Arria 10), and the Xeon® CPU. Note that centroids calculation does not represent much in time to solution. Indeed, this step walks through centroids and their assigned points to performs an arithmetic mean, which is not much complex, and we chose to execute it at the host.

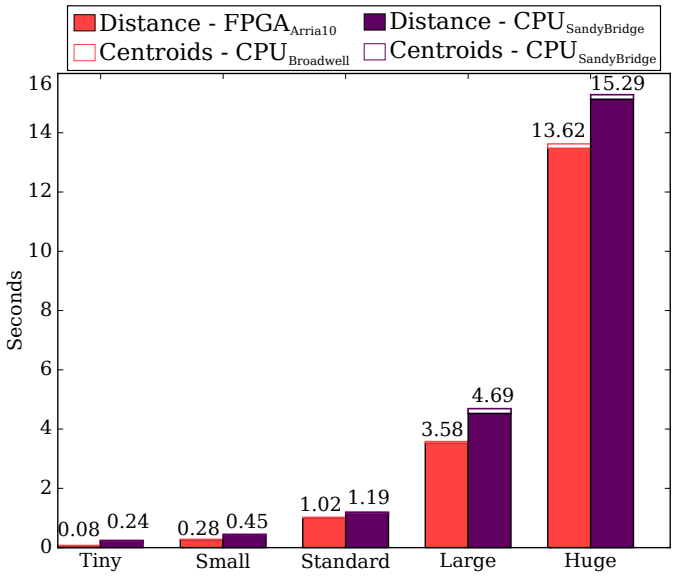

Fig. 4. Time results.

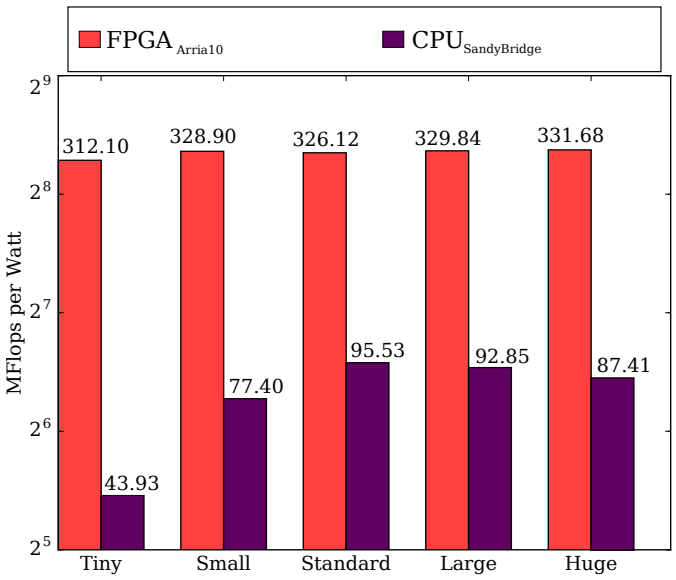

Fig. 5. Energy efficiency results.

We measured the time taken to off-load data from host to kernel on the integrated platform. The read/write time was, at most, 1.22% from the overall time, with Tiny input. This decreases even more when input size is increased, reaching 0.06% when Huge was used. That demonstrates the advantages of the platform, using better approaches to fine grain communication than traditional FPGAs systems.

With respect to distances computing, we verified that our kernel hardware scaled well when the input size was increased. The increase in time is pronounced from Tiny to Huge due to the number of iterations, which is a behavior that cannot be changed without reduce clustering precision. However, we evaluated each iteration in an isolated manner, founding that increasing the input size did not make the FPGA have its performance impaired. In other words, the presented time to solution increase was directly proportional to the input size.

Regarding the CPU-only approach, we note that it presented a better scalability. The advantages from CPU are processing power (higher frequency) and use of caches. Despite that, with even higher input sizes, the FPGA will keep its proportional scalability, while the CPU may slow down, due to potential synchronization operations overhead and cache misses. Nevertheless, a good point is that K-Means kernel runs faster at the integrated FPGA platform, when compared against a Xeon® CPU. The difference between these architectures were, from Tiny to Huge input sizes, 68.21%, 36.57%, 14.03%, 23.59% and 10.82%.

C. Energy consumption

In order to understand the energy efficiency from our implementation using the novel platform, we compared energy consumption from this device and from CPU. Figure 5 shows measured floating point operations per second (Flops) per Watt, that is, the energy efficiency. The information on the chart pertains only to the distances computing step, enabling a specific understanding of the energy consumption on the FPGA. Although it may seem unfair, it is not, because the synchronization with the host and centroids computing step are not much relevant to this measure, that is, the distance computing is the most consuming part of the algorithm.

When the input size was increased from Tiny to Standard, the CPU presented 2.17× more energy efficiency, while the FPGA did not change much. However, if we increase the input sizes even more, the CPU starts to reduce its efficiency. On the other hand, the efficiency of the FPGA increased a bit. Although it represents only 6.27% of increase, the scalability of energy consumption on Arria 10 is much better than on the CPU.

Recall that time spent to perform K-Means in Arria 10 is better than in CPU-only too, which led to less energy being consumed. However, the main factor for such difference is power consumption. The CPU has much complex computational units and higher frequency than the FPGA. Besides that, a more robust memory system and multicore and multithread support raises this power consumption. We evaluate our K-Means hardware model, which presented a total power consumption of 23.05 Watts, while thermal design power (TDP) from Intel® Xeon® E5-2620 is 95 Watts.

The power consumption on the FPGA is almost constant, which makes energy consumption to follow this mentioned behavior. From Standard to Huge input sizes, we found an increase factor in energy consumption of less than 2×. It is worth noting that workload size also increased in such factor, which indicates that even bigger workloads will have an acceptable proportional increase. Generally, we found that using a hybrid CPU+FPGA platform was more energy efficient than the CPU-only approach from 70.71% (Standard) to 85.92% (Tiny).

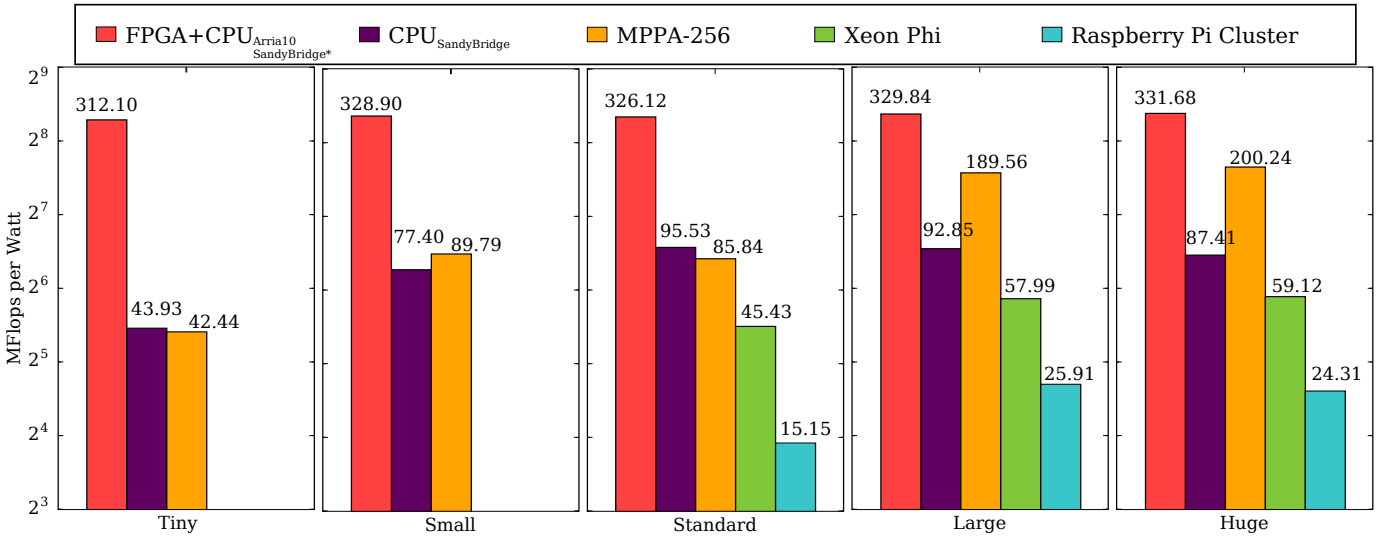

Fig. 6. Comparison against other platforms.

D. Other platforms

In this Section, we present a comparison between the results obtained in this work against those from others, using three different platforms. The first one is the Xeon Phi™, a manycore co-processor with 61 cores and support for 244 threads. The second one is the 8-node Raspberry Pi Cluster, with each board composed by a quad-core ARM processor, totaling 32 threads. The last one is the Kalray MPPA-256, a low-power manycore processor that has 16 compute clusters with 16 cores each running at low frequency (400MHz).

The results we obtained with those processors using K-Means are described in other papers [17], [21]. We run K-Means on them using the same input sizes we used in this work. Figure 6 shows the comparison of the platforms, in MFlops per Watt. Tiny and Small input sizes were not tested for the Xeon Phi™ and the Raspberry Pi Cluster, thus we do not present these results in the figure. The time and energy consumption were measured considering both distance and centroids compute steps, thus, we need to perform the same measuring. Unfortunately, it was not possible to read energy from the Broadwell CPU (Figure 6, FPGA+CPU*) at the environment we have access to, thus, to make the analysis fairer, we considered the values from the centroids compute step from SandyBridge, along with the Arria 10 values.

From Figure 6, we can note that our K-Means using a hybrid CPU+FPGA platform is considerably more efficient than others. Indeed, the use of a hardware designed for a specific problem leads to high performance. Furthermore, even if we consider low-power architectures (Raspberry and MPPA-256), FPGAs end up having simpler circuits and mechanisms of control, which improves energy efficiency.

The Raspberry Pi Cluster presented poor efficiency due to its low computing performance. However, it has the advantage of scalability at low cost, since Raspberry Pi boards have lower prices than the other mentioned platforms. The Arria 10 at the integrated platform is up to 21.5× more efficient than the Raspberries with the Standard input size. The Xeon Phi™ has high processing power capabilities with its multiple cores, but it suffers in relation to power consumption, due to its complex components and interconnection system. The FPGA approach is up to 7.2× more efficient than the Xeon Phi™, when running K-Means with Standard input size. Last, but not least, MPPA-256 was the closest one to the integrated platform. Using Huge input size, the FPGA presented results 1.7× better. If we look at the Standard size, we verified results 3.8× better for the FPGA.

VII. Conclusion

In this work, we present a performance evaluation of the K-Means Clustering algorithm on the novel hybrid CPU+FPGA platform from Intel®, a multicore Xeon® CPU and an Arria 10 FPGA in the same die. We measured the performance and energy consumption of this platform when running our implementation. We considered different architectural decisions and optimizations, and we contrasted the performance with a Xeon® CPU-only platform and others.

As a first statement, we noted that the high level synthesis approach, using the OpenCL language, brought some advantages, such as faster development and more assertive optimizations. Also, it could attract software developers to the use of FPGAs. In a nutshell, the K-Means for the integrated platform proved to be better than the CPU-only version by up to 85.92% with Tiny input size, in means of energy efficiency. An improvement of up to 68.21% in performance was also noted, with Tiny input size too. It is worth noting that during the distance computing, the host multicore stayed idle, an thus it could be used to further improve the parallel computation, using a specific heterogeneous-aware load balance strategy, or even be used to process other algorithm or computational task.

Furthermore, the MCP platform achieved better results than other platforms, which were used before to run K-Means with

the same input sizes we used in this work, also with regard to energy efficiency. The hybrid CPU+FPGA approach proved to be more energy efficient than a low power 8-node Raspberry Pi Cluster up to 21.5×, better than an Intel® Xeon Phi™ processor up to 7.2×, and up to 3.8× better than the Kalray MPPA-256 with the Standard input size, in terms of Flops per Watt.

The Intel® Hybrid CPU+FPGA MCP can be used to accelerate K-Means and further Machine Learning algorithms using the OpenCL language to High Level Synthesis, with high performance and energy efficiency. However, it is important to be aware of strategies for reusing data on the FPGA, avoiding off-chip communications. Also, multiple compute units and loop unrolling are good strategies when dealing with Machine Learning non-dependent data objects.

As future works, we intend to explore Shared Virtual Memory features from the platform; to study load balance algorithms to exploit the heterogeneity of the system; and to evaluate further Machine Learning algorithms, such as DNNs, Support Vector Machines, among others.

ACKNOWLEDGMENT

The presented results were partially obtained on resources hosted at the Paderborn Center for Parallel Computing (PC^2) in the Intel® Hardware Accelerator Research Program (HARP2) [27]. We thank CAPES, FAPEMIG, CNPq for the partial support in this work.

REFERENCES

[1] D.-J. Lim, T. R. Anderson, and T. Shott, "Technological forecasting of supercomputer development: The march to exascale computing," *Omega*, vol. 51, pp. 128 – 135, 2015.

[2] H. Simon, "Barriers to exascale computing," in *High Performance Computing for Computational Science (VECPAR)*. Kope, Japan: Springer, 2012, pp. 1–3.

[3] Q. Y. Tang and M. A. S. Khalid, "Acceleration of k-Means Algorithm Using Altera SDK for OpenCL," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 1, pp. 6:1–6:19, 9 2016.

[4] L. A. Maciel, M. A. Souza, and H. C. de Freitas, "Projeto e Avaliação de uma Arquitetura do Algoritmo de Clusterização K-means em VHDL e FPGA," *Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*, 2017.

[5] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of FPGAs and GPUs," in *International Symposium on Field-Programmable Gate Arrays*, New York, USA, 2018, pp. 288–288.

[6] K. O'Brien, L. D. Tucci, G. Durelli, and M. Blott, "Towards exascale computing with heterogeneous architectures," in *Design, Automation Test in Europe Conference Exhibition*, March 2017, pp. 398–403.

[7] R. Tummala, N. Nedumthakady, S. Ravichandran, B. DeProspo, and V. Sundaram, "Heterogeneous and homogeneous package integration technologies at device and system levels," in *Pan Pacific Microelectronics Symposium*, Feb 2018, pp. 1–5.

[8] P. Colangelo, E. Luebbers, R. Huang, M. Margala, and K. Nealis, "Application of convolutional neural networks on Intel® Xeon® processor with integrated FPGA," in *IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2017, pp. 1–7.

[9] W. Qiao, J. Du, Z. Fang, L. Wang, M. Lo, M.-C. F. Chang, and J. Cong, "High-Throughput Lossless Compression on Tightly Coupled CPU-FPGA Platforms," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: ACM, 2018, pp. 291–291.

[10] F. A. M. Alves, P. Jamieson, L. B. da Silva, R. S. Ferreira, and J. A. M. Nacif, "Designing a collision detection accelerator on a heterogeneous CPU-FPGA platform," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2017, pp. 1–6.

[11] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11 – 26, 2017.

[12] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov 2017.

[13] M. Allahyari, S. A. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "A brief survey of text mining: Classification, clustering and extraction techniques," *CoRR*, vol. abs/1707.02919, 2017. [Online]. Available: http://arxiv.org/abs/1707.02919

[14] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

[15] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[16] M. Estlick *et al.*, "Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware," in *International Symposium on FPGA*. ACM, 2001, pp. 103–110.

[17] M. A. Souza, P. H. Penna, M. M. Queiroz, A. D. Pereira, L. F. W. Góes, H. C. Freitas, M. Castro, P. O. Navaux, and J. Méhaut, "CAP Bench: a benchmark suite for performance and energy evaluation of low-power many-core processors," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 4, p. e3892, 2017.

[18] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ deep learning accelerator on Arria 10," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1701.03534

[19] D. Wang, J. An, and K. Xu, "PipeCNN: An OpenCL-based FPGA accelerator for large-scale convolution neuron networks," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1611.02450

[20] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *International Symposium on Field-Programmable Gate Arrays*. New York, USA: ACM, 2017, pp. 25–34.

[21] J. Saffran, G. Garcia, M. A. Souza, P. H. Penna, M. Castro, L. F. W. Góes, and H. C. Freitas, "A Low-Cost Energy-Efficient Raspberry Pi Cluster for Data Mining Algorithms," in *Euro-Par: Parallel Processing Workshops*. Cham: Springer, 2017, pp. 788–799.

[22] J. Canilho, M. Véstias, and H. Neto, "Multi-core for K-means clustering on FPGA," in *International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[23] M. Baydoun, H. Ghaziri, and M. Al-Husseini, "CPU and GPU parallelized kernel K-means," *Journal of Supercomputing*, pp. 1–24, 2018.

[24] M. Vidal, B. Arejita, J. Diaz, C. Alvarez, D. Jiménez-González, X. Martorell, F. Mantovani *et al.*, "Implementation of the K-means algorithm on heterogeneous devices: a use case based on an industrial dataset," in *Parallel Computing is Everywhere (serie: Advances in Parallel Computing)*, vol. 32. IOS Press, 2018, pp. 642–651.

[25] H. Mardani Kamali and A. Sasan, "MUCH-SWIFT: A High-Throughput Multi-Core HW/SW Co-design K-means Clustering Architecture," in *Proceedings of the Great Lakes Symposium on VLSI*, ser. GLSVLSI '18. New York, NY, USA: ACM, 2018, pp. 459–462. [Online]. Available: http://doi.acm.org/10.1145/3194554.3194648

[26] Z. Lin, C. Lo, and P. Chow, "K-means implementation on FPGA for high-dimensional data using triangle inequality," in *International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2012, pp. 437–442.

[27] I. Corporation, "Hardware accelerator research program." [Online]. Available: https://software.intel.com/en-us/hardware-accelerator-research-program

[28] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting Performance Data with PAPI-C," in *Tools for High Performance Computing*. Berlin, Heidelberg: Springer, 2010, pp. 157–173.

[29] G. Chrysos, "Intel ® Xeon Phi coprocessor (codename Knights Corner)," in *IEEE Hot Chips Symposium (HCS)*, Aug 2012, pp. 1–31.

[30] E. Upton and G. Halfacree, *Raspberry Pi user guide*. Wiley, 2014.

[31] B. D. de Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss *et al.*, "A clustered manycore processor architecture for embedded and accelerated applications," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*. IEEE, 2013, pp. 1–6.