TensorFlow on state-of-the-art HPC clusters: a machine learning use case

Guillem Ramirez-Gargallo
Computer Science Department
Barcelona Supercomputing Center
Barcelona, Spain
guillem.ramirez@bsc.es

Marta Garcia-Gasulla
Computer Science Department
Barcelona Supercomputing Center
Barcelona, Spain
marta.garcia@bsc.es

Filippo Mantovani
Computer Science Department
Barcelona Supercomputing Center
Barcelona, Spain
filippo.mantovani@bsc.es

***Abstract*—The recent rapid growth of the data-flow programming paradigm enabled the development of specific architectures, e.g., for machine learning. The most known example is the Tensor Processing Unit (TPU) by Google. Standard data-centers, however, still can not foresee large partitions dedicated to machine learning specific architectures. Within data-centers, the High-Performance Computing (HPC) clusters are highly parallel machines targeting a broad class of compute-intensive workflows, as such they can be used for tackling machine learning challenges. On top of this, HPC architectures are rapidly changing, including accelerators and instruction sets other than the classical x86 CPUs. In this blurry scenario, identifying which are the best hardware/software configurations to efficiently support machine learning workloads on HPC clusters is not trivial. In this paper, we considered the workflow of TensorFlow for image recognition. We highlight the strong dependency of the performance in the training phase on the availability of arithmetic libraries optimized for the underlying architecture. Following the example of Intel leveraging the MKL libraries for improving the TensorFlow performance, we plugged the Arm Performance Libraries into TensorFlow and tested on an HPC cluster based on Marvell ThunderX2 CPUs. Also, we performed a scalability study on three state-of-the-art HPC clusters based on different CPU architectures, x86 Intel Skylake, Arm-v8 Marvell ThunderX2, and PowerPC IBM Power9.**

***Index Terms*—TensorFlow, High Performance Computing, Parallel Computing, Machine Learning, Image Recognition, Training, Arm, Power9, x86, Clusters**

I. Introduction

The use of Machine Learning (ML) in research and industry is growing, and with its growth, there is an increasing need for computational resources able to efficiently handle ML workloads. One approach to cope with this request is to develop domain-specific architectures (the most prominent example being the Tensor Processing Unit (TPU) by Google [1]). Another approach is to employ the latest generations of Graphics Processing Units (GPU) offering ML specialized cores (see e.g., [2]). Another transversal approach is to link the back-end of ML frameworks to optimized libraries so to improve the performance (and the efficiency) of standard homogeneous clusters based on high-end CPUs. This last approach can imply to leverage vendor-specific libraries targeting ML (see e.g., [3]) or develop kernels optimized for specific operations/kernels and architectures (see e.g., [4]).

Since most data centers cannot afford to specialized their hardware deployment for ML, it is essential to have ML tools making good use of computing resources, especially very high-end resources like the one in High-Performance Computing (HPC) data centers. HPC data centers are traditionally populated with x86 architectures, but recently the trend is changing: the first ranked supercomputer of the Top500 is indeed Summit, an IBM PowerPC architecture (boosted by NVIDIA GPUs). The Barcelona Supercomputing Center (BSC) hosts a 1.5 PFlops partition of compute nodes identical to the ones installed in Summit. Also, for the first time in the history of Top500, on November 2018 an Arm-based system, Astra, also entered the Top500 [5]. The BSC is engaged through the Mont-Blanc project in enabling the Arm architecture into HPC [6]. For these reasons, we present in this paper how to leverage the Arm Performance Libraries within TensorFlow and a complete evaluation of TensorFlow on modern HPC clusters, including x86, Power9 and Arm.

The main contributions of this paper are: *i)* the performance evaluation of hybrid configurations of TensorFlow in three state-of-the-art HPC based on different architectures; *ii)* the measurement of the performance benefits when plugging the Arm Performance Libraries to TensorFlow; *iii)* the study of the scalability of TensorFlow when running up to 1024 cores of state-of-the-art HPC clusters.

The remaining part of the paper is organized as follows: in Section II we define our test case and the method used for performing the following experiments. Section III summarizes the HPC hardware platforms on which we performed our tests; in Section IV we present the performance effects of using vendor-specific linear algebra libraries when running TensorFlow, while in Section V we present the scalability results measured on three HPC homogeneous clusters based on different architecture, x86, PowerPC and Arm. In Section VI we briefly compare our work with the most recent contributions related to TensorFlow in the HPC field. We summarize our final comments in VII.

This work is partially supported by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology project (TIN2015-65316-P), by the Generalitat de Catalunya (2017-SGR-1414), and by the European Community's Seventh Framework Programme [FP7/2007-2013] and Horizon 2020 under the Mont-Blanc projects, grant agreements n. (288777, 610402 and 671697).

II. Machine Learning Environment

TensorFlow is an open-source library for Machine Learning (ML) applications e.g., for image classification, recommendation of applications, and speech recognition [7]. It is designed and implemented by the Google Brain Team, written in C++ and Python and some of its parts use CUDA for acceleration on GPUs. It consists of ∼650,000 lines of code. Both the scientific community and industry (e.g., Intel and NVIDIA) contribute to its development[1]. It is employed as a benchmark of new ML architectures [8] and as ML engine when coupled with well-trained models [9].

Concerning parallelism, TensorFlow defines *intra-ops* as the number of threads used by a kernel and *inter-ops* the level of parallelism expressed at the node level of a graph, in other words, how many different kernels can be performed at the same time. *Horovod* [10] is a distributed training framework for TensorFlow and other machine learning frameworks. We use Horovod for allocating work among processes.

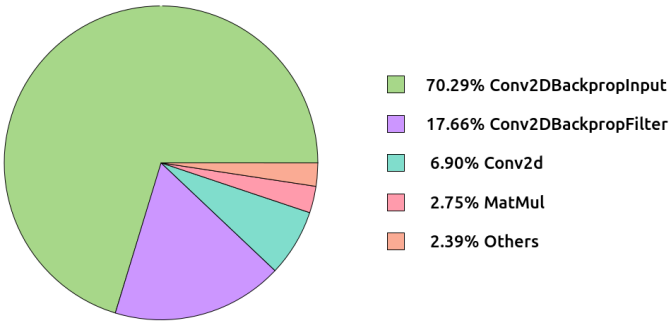

Fig. 1. Time distribution among operations for the `AlexNet` model on Marvell ThunderX2 CPU.

In Figure 1, we can see that ∼70% of the total execution time is spent in the `Conv2DBackpropInput` function, while ∼18% of the time is spent in `Conv2DBackpropFilter`. Of course, speeding up these two operations would improve the overall performance. We profiled (with `perf`) the inner calls and matched the most time-consuming functions and operations. As expected, all of them are part of the Eigen library. Eigen[2] is a C++ library for linear algebra. In the generic kernels (i.e., the generic implementation of an operation), Eigen is used for tensor operations. Since the Eigen library is a collection of linear algebra functions, we studied how to replace some of the Eigen calls with the corresponding optimized implementations that can be found in vendor specific linear algebra libraries. While for x86 architecture already existed a version of TensorFlow leveraging MKL, for Arm architectures we decided to test the Arm Performance Libraries (ArmPL). Due to the dominance in the execution time, we integrate calls to the Arm Performance Libraries first into the `Conv2DBackpropInput` and `Conv2DBackpropFilter` operations.

We perform our study using TensorFlow for training a network with images, using the benchmark tool from the TensorFlow repository[3]. In order to avoid I/O overhead, we perform all tests with a synthetic image set, however for the scalability study we also tested the ImageNet data set [11]. We apply two different models, ResNet-50 as an example of deep multi-layer network and AlexNet for legacy reasons. Since we are applying machine learning techniques for image recognition, we try to express our results homogeneously, considering always *images per second* (img/s) as the performance figure.

III. High Performance Computing Environment

For our experiments we used three state-of-the-art production HPC clusters: MareNostrum4, Power9 and Dibona, each one leveraging different architectures:

MareNostrum4 is a supercomputer based on Intel Xeon Platinum processors, Lenovo SD530 Compute Racks, a Linux Operating System and an Intel Omni-Path interconnection. Its general purpose partition has a peak performance of 11.15 Petaflops, 384.75 TB of main memory spread over 3456 nodes. Each node houses 2× Intel Xeon Platinum 8160 with 24 cores at 2.1 GHz, 216 nodes feature 12×32 GB DDR4-2667 DIMMS (8 GB/core), while 3240 nodes are equipped with 12×8 GB DDR4-2667 DIMMS (2 GB/core). MareNostrum4 is the PRACE Tier-0 supercomputer hosted at the Barcelona Supercomputing Center (BSC) in Spain and ranked 25 in the Top500 list of November 2018 [12].

Power9 is also hosted at Barcelona Supercomputing Center. This cluster is based on IBM Power9 8335-GTG processors with 20 cores each CPU operating at 3 GHz. Each compute node contains two CPUs, plus four GPUs NVIDIA V100 with 16 GB HBM2. Each compute node is equipped with 512 GB of main memory distributed in 16 DIMMS of 32 GB operating at 2666 MHz. Nodes are interconnected with an Infiniband Mellanox EDR network and the operating system is Red Hat Enterprise Linux Server 7.4. The Power9 cluster has been included in our study because its computational elements are architecturally identical to the ones of the Summit supercomputer, ranked first in the Top500 of November 2018 [12]. It must be clarified that we do not consider in our evaluation the accelerator part composed by the GP-GPUs.

Dibona is an Arm-based HPC cluster, designed and deployed by ATOS/Bull within the Mont-Blanc 3 project. Its first evaluation and benchmark is presented in [13]. Each compute node is powered by two Marvell's ThunderX2 CPUs with 32 cores each operating at 2.0 GHz. The main memory on each node is 256 GB of DDR4 running at 2667 MHz. Nodes are interconnected with Infiniband Mellanox EDR network. The Dibona cluster has been considered for our study because it features the same CPU technology that composes the Astra supercomputer, the first Arm-based system ranked 204 in the Top500 list of November 2018 [5].

In Table I we show the software environment used in each cluster and for each version. We tried to keep the

[1] https://github.com/tensorflow/tensorflow/tree/master/tensorflow
[2] http://eigen.tuxfamily.org/
[3] https://github.com/tensorflow/benchmarks

TABLE I
ENVIRONMENT FOR EACH CLUSTER

Cluster	TF version	Compiler	MPI	Back-End / Perf. Libs.	Front-End Flags
MN4	r1.11	GCC 7.2.0	OpenMPI 3.1.1	-	-mtune=skylake-avx512 -march=skylake-avx512 -O3
MN4	r1.11	GCC 7.2.0	OpenMPI 3.1.1	MKL DNN v0.16	-mtune=skylake-avx512 -march=skylake-avx512 -O3
Dibona	r1.11	GCC 8.2.0	OpenMPI 2.0.2.14	-	-march=native -mtune=thunderx2t99 -O3
Dibona	r1.11*	GCC 8.2.0	OpenMPI 2.0.2.14	ArmPL 19.0	-march=native -mtune=thunderx2t99 -O3
Power9	r1.11	GCC 8.2.0	OpenMPI 3.1.1	-	-mtune=power9 -mcpu=power9 -O3

environment across clusters as homogeneous as possible based on available software. We used the TensorFlow code in the version 1.11, it has been modified only in the version using the Arm Performance Libraries (ArmPL), marked in the table as TensorFlow version `r1.11*`. Also, we did not have access to vendor specific libraries for Power9, so we evaluated only the vanilla version of TensorFlow on it.

IV. INTRANODE EVALUATION

In this section, we evaluate the performance of TensorFlow within a computational node. We divide the study into two parts; first, we study the effect on the performance (img/s) of changing the number of threads varying the batch size. Second, we evaluate the best configurations of MPI processes and threads when using all the computational resources of one node.

We perform both studies on the vanilla version of TensorFlow and the version leveraging vendor-specific linear algebra libraries (MKL for x86 and Arm Performance Libraries for Arm). The goal is to be able to compare the improvement in performance that can be achieved when using optimized linear algebra libraries as back-end.

For this evaluation, we use the three HPC clusters described in Section III: MareNostrum4, Dibona, and Power9.

We train our network for several epochs until it reaches a steady regime of img/s and checking the increasing trend of the accuracy to ensure that the network is being effectively trained. We measure sustained performance as reported by TensorFlow. To provide a reasonable statistical accuracy, we average four executions for each test. Since we notice variability below 10% in all our measurement campaign, we neglect error bars.

A. Thread Scaling

To evaluate the performance of thread scaling within a node, we use the AlexNet model, and we increase the number of threads used by TensorFlow from one to the number of cores available for each architecture.

In Figure 2 we can see the performance, expressed in images per second when training the AlexNet model using the vanilla version of TensorFlow in MareNostrum4. The x-axis represents the number of threads and in the y-axis is shown the performance obtained, the different series correspond to the different batch sizes used for the training.

We can observe that bigger batch sizes have a better performance than smaller ones. A batch size of 32 loses performance when using more than 4 threads while a batch size of 64 can scale up to 16 threads. For bigger batch sizes

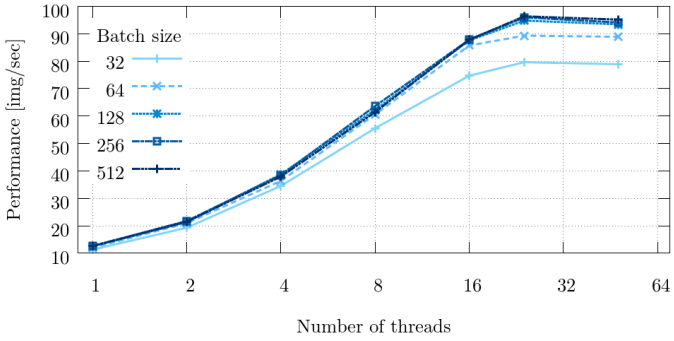

Fig. 2. Thread Scaling of AlexNet with Vanilla version on one node of MareNostrum4

the scalability is reduced when reaching 24 threads, in the case of MareNostrum4 this is the number of cores per socket. Therefore, it is not optimal to run a single process using the two sockets of one node. We can also observe that for batch sizes of 128, 256 and 512 there is almost no difference in performance.

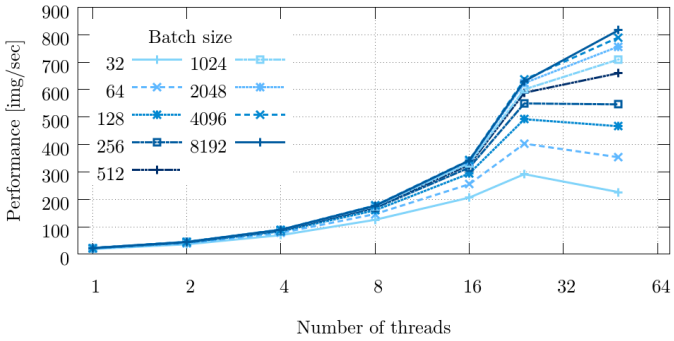

Fig. 3. Thread Scaling of AlexNet with MKL version on one node of MareNostrum4

In Figure 3 is shown the performance of AlexNet in MareNostrum4 when running the TensorFlow version using MKL libraries. Comparing Figure 2 and 3 we can observe that the performance is increased almost by 7× by using the optimized linear algebra libraries provided by the vendor (MKL libraries). In this case, we evaluated batch sizes up to 8192, because for a high number of threads the performance can still be improved by increasing the batch size.

It is also interesting to see that when using de MKL libraries the size of the batch size have a higher impact on the performance. Also, the effect of using two sockets (48

threads) is worst when using the MKL libraries, resulting in worse performance than with 24 threads when using batch sizes of 512 or less.

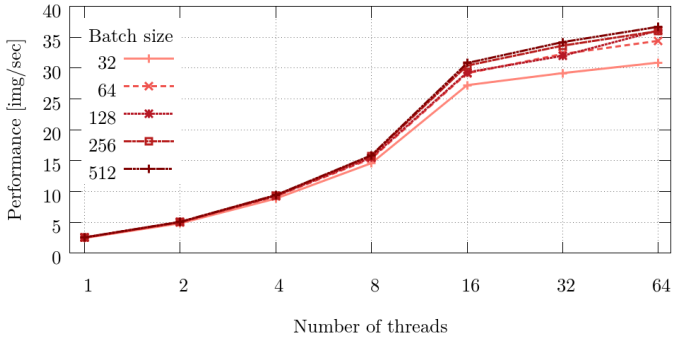

Fig. 4. Thread Scaling of AlexNet with Vanilla version on one node of Dibona

In Figure 4 we can see the performance of training the AlexNet model in Dibona with the vanilla version of TensorFlow. On the x-axis is represented the number of threads used while on the y-axis the performance obtained in images processed per second.

We can observe that the performance when increasing the number of threads drops at 16 threads, that is, before reaching the number of threads corresponding to the cores in the socket in Dibona (32 cores per socket). In this case, we also observe that the performance increases with the batch size, being the difference more important for a large number of threads.

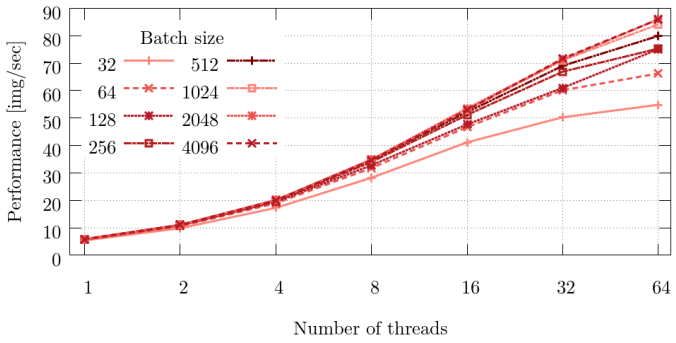

Fig. 5. Thread Scaling of AlexNet with ArmPL version on one node of Dibona

Figure 5 shows the performance obtained in Dibona using the modified version of TensorFlow leveraging the Arm Performance Libraries. We can observe that the performance is more than 2× better than using the vanilla version of TensorFlow. It is also important to notice that the Arm Performance Libraries version delivers a good thread scaling up to 64 threads.

Also in this case, we evaluated higher batch sizes as the performance for a high number of threads increases for batch sizes above 512.

In Figure 6 we plot the performance obtained in Power9 when using the vanilla version of TensorFlow. In Power9 we can observe a significant performance degradation when going from 20 to 40 threads. This effect can be explained,

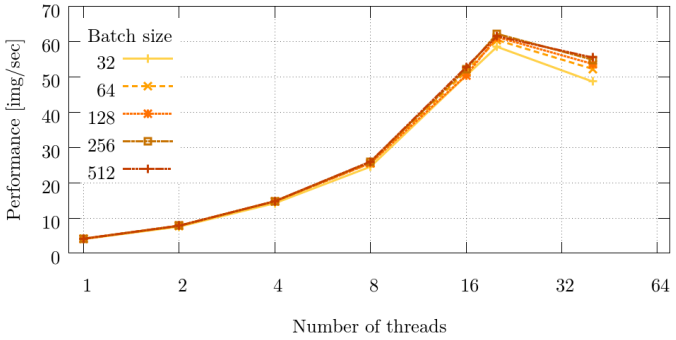

Fig. 6. Thread Scaling of AlexNet with Vanilla version on one node of Power9

as for MareNostrum4, by the fact that a computational node is composed of two sockets, and it is not optimal to spawn threads of the same process across different sockets.

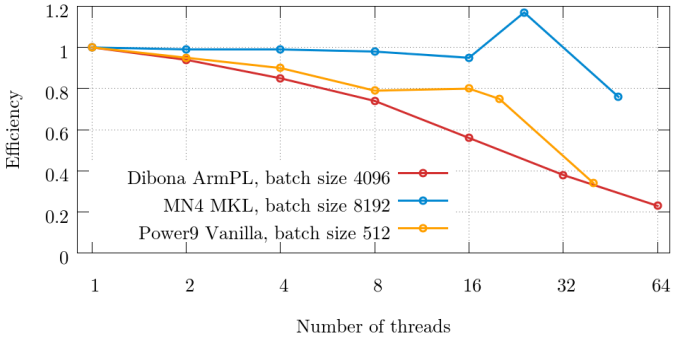

Fig. 7. Efficiency of AlexNet thread scalability in the different platforms

Figure 7 shows the efficiency e of the three clusters when running with the batch size delivering the best performance. The efficiency e is computed as $e = \frac{p_t}{(t \cdot p_1)}$ where p_t is the performance expressed in img/s when running with t threads and p_1 is the performance when running with one thread.

We can observe that in MareNostrum4 the efficiency is almost perfect up to 24 threads, but it drops for 48 threads. In the case of Power9, the efficiency is quite good up to 20 threads, although it does not reach the efficiency obtained in MareNostrum4. The efficiency of Power9 drops at 40 cores, like in MareNostrum4, when using both sockets of the node. Finally, in Dibona, the efficiency decreases constantly when increasing the number of threads.

B. Hybrid Configurations Evaluation

We have seen that TensorFlow in different machines and with a batch size high enough can scale up to the number of threads of a socket (24 in MareNostrum4, 32 in Dibona and 20 in Power9). In this subsection, we analyze which is the optimum configuration between the number of MPI processes and the threads inside a computational node.

For this evaluation, we use two different models: AlexNet and ResNet-50, and we will vary the distribution of resources among processes and threads always using all the computational resources available.

In the previous subsection, we have seen that bigger batch sizes provide better performance. We have also observed that higher batch sizes imply a higher memory consumption. For this reason, we cannot run the same batch size for all different configurations, because in almost all the cases we run out of memory.

In Table II we can see the different batch sizes used for each configuration on each HPC platform.

TABLE II
BATCH SIZE ASSIGNED TO EACH PROCESS, FOR EACH CONFIGURATION OF PROCESS × THREAD

MareNostrum4	1×48	2×24	4×12	8×6	16×3	24×2	48×1
AlexNet	8192	4096	2048	1024	512	341	170
ResNet-50	512	256	128	64	32	21	10
Dibona	1×64	2×32	4×16	8×8	16×4	32×2	64×1
AlexNet	512	256	256	256	256	256	256
ResNet-50	64	64	64	64	64	64	64
Power9	40×1	20×2	10×4	8×5	20×2	40×1	
AlexNet	512	256	256	256	256	256	
ResNet-50	64	64	64	64	64	64	

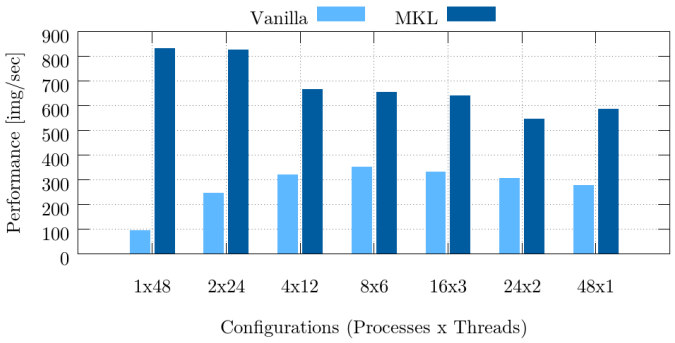

Fig. 8. Performance of hybrid configuration of AlexNet model in MareNostrum4

In Figure 8 we can see the performance obtained with different configurations when using AlexNet in MareNostrum4. On the x-axis are represented the different configurations as *MPI processes* × *threads*.

We can observe that when using the vanilla version the best configuration is 8×6, and in general the configurations that are not extreme (i.e., high number of threads or high number of processes). But when using TensorFlow with MKL, the best configurations are with the highest number of threads per node and only one or two MPI processes.

Figure 9 contains the performance of ResNet-50 model in MareNostrum4. For this model, the best configurations of MPI processes and threads when using the vanilla version is very similar to the one obtained with AlexNet. The version that uses MKL seems to be slightly different, being the worst configurations the ones in the middle. But, still, the best option is to use a high number of threads and one MPI process.

In Figure 10 we can see the performance in Dibona when training an AlexNet model with images. We can observe a similar trend in the best configuration when using the vanilla version and the Arm PL one. In this case, the best

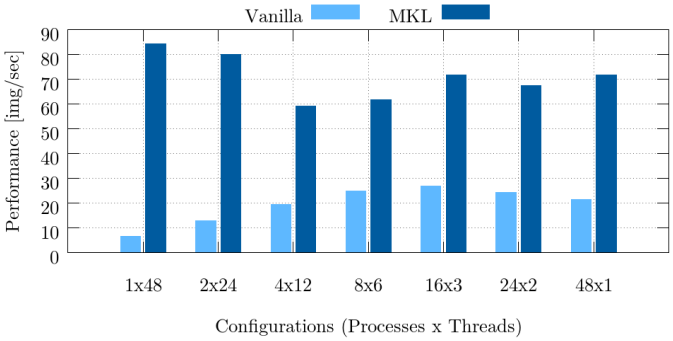

Fig. 9. Performance of hybrid configuration of ResNet-50 model in MareNostrum4

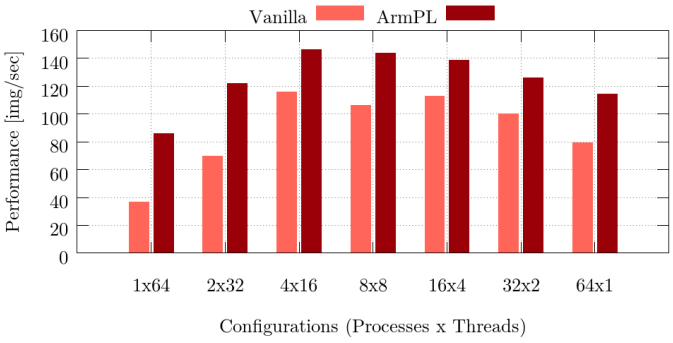

Fig. 10. Performance of hybrid configuration of AlexNet model in Dibona

configuration in both cases is using 4 MPI processes and 16 threads per MPI process.

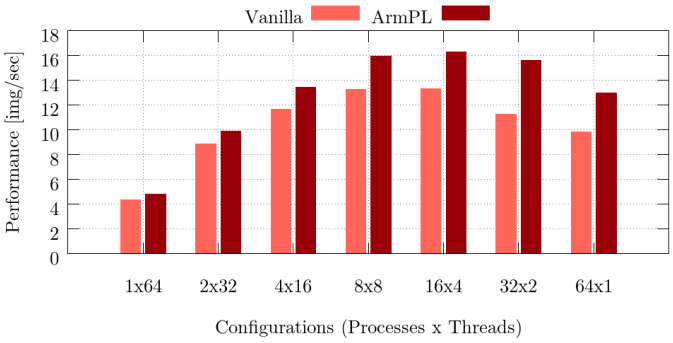

Fig. 11. Performance of hybrid configuration of ResNet-50 model in Dibona

Figure 11 shows the performance obtained by the different configurations in Dibona when using the ResNet-50 model. For the ResNet-50 model, the best configuration is to spawn 16 MPI processes and 4 threads each one. The trend is very similar to the one observed when using the AlexNet model.

Figures 12 and 13 show the performance obtained in Power9 using different configurations for AlexNet and ResNet-50 models. In Power9 the best trend seems to be to use more processes and fewer threads, being the best configuration for both models to use 20 MPI processes and 2 threads each one.

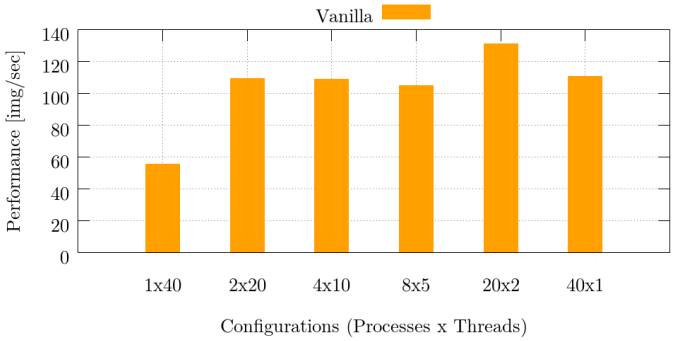

Fig. 12. Performance of hybrid configuration of AlexNet model in Power9

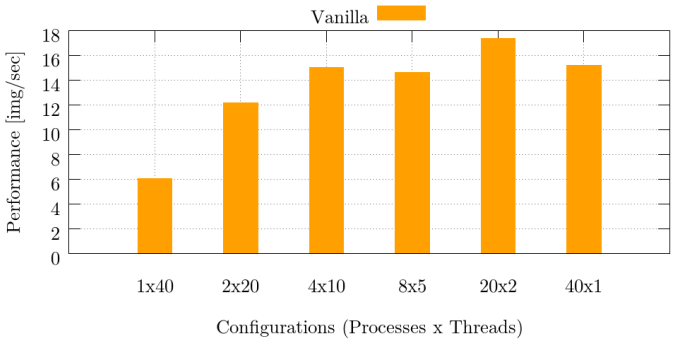

Fig. 13. Performance of hybrid configuration of ResNet-50 model in Power9

V. Scalability

Finally, we evaluate the scalability of AlexNet and ResNet-50 in the three HPC clusters. For this evaluation, we use the best configurations learned from the previous section. We use synthetic images and real ones from the ImageNet dataset.

In Table III we report a summary of the different parameters used to train the two models in the tree platforms.

TABLE III
Configurations used for each cluster and model.

Cluster	Model	Configuration	Inter-ops	Batch Size
MareNostrum4	AlexNet	1×48	1	8192
MareNostrum4	ResNet-50	1×48	1	512
Dibona	AlexNet	4×16	2	2048
Dibona	ResNet-50	8×8	2	64
Power9	AlexNet	2×20	20	2048
Power9	ResNet-50	20×2	2	128

In Figure 14 we can see the scalability up to 16 nodes of the AlexNet model. On the x-axis are represented the number of nodes and on the y-axis the performance in images processed per second. It is interesting to see that the scalability is similar in all clusters, but also that in MareNostrum4 the difference in performance between using synthetic images or real ones is much higher than in the other clusters. This could be explained because real images need to communicate through the network more than when using synthetic ones and the three clusters use different network technologies.

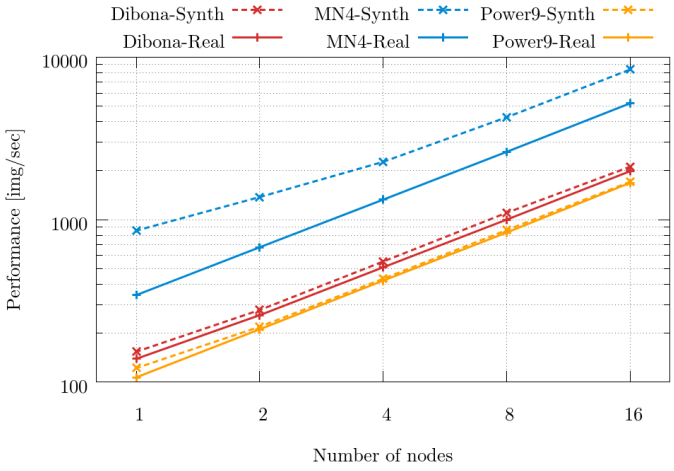

Fig. 14. Scalability of Alexnet model on the different platforms

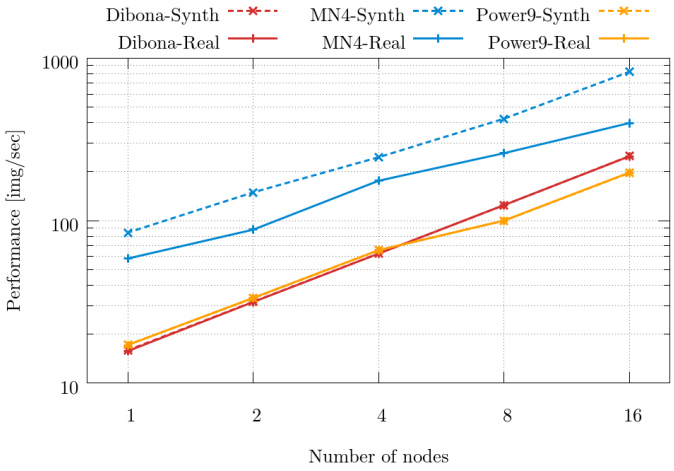

Fig. 15. Scalability of ResNet-50 model on the different platforms

Figure 15 shows the scalability of the ResNet-50 model. We can see that the scalability of this model is worse and more irregular than when using AlexNet in MareNostrum4 and Power9. This can be explained by the fact that this model is more complex due to its the deep multi-layer network. We see again that the performance with synthetic images has a much better performance than the one achieved with real images in MareNostrum4 compared to the other systems. The performance of the training phase of TensorFlow strongly depends on collective communications. Also our three HPC clusters have different interconnection tecnology, Dibona and Power9 use Infiniband while MareNostrum4 uses Intel Omi-Path. Collective operations have different performance on this two network technologies (see [13] for details), this could explain the results we obtain.

For a deeper understanding of the behaviour at scale, Figures 14 and 15 can be complemented with the efficiency heat map shown in Table IV. In the table we plot the relative efficiency E_i on each machine, computed as $E_i = \frac{P_i}{(P_1 \cdot i)}$, where P_i is the performance in images per seconds obtained

TABLE IV
PARALLEL EFFICIENCY ALEXNET AND RESNET-50 MODELS ON THE DIFFERENT PLATFORMS

	# of Nodes	MN4 (Synth)	MN4 (Real)	Power9 (Synth)	Power9 (Real)	Dibona (Synth)	Dibona (Real)
AlexNet	1	1.00	1.00	1.00	1.00	1.00	1.00
	2	0.80	0.98	0.90	0.99	0.90	0.93
	4	0.66	0.97	0.88	0.98	0.90	0.91
	8	0.62	0.95	0.88	0.97	0.89	0.90
	16	0.61	0.94	0.87	0.98	0.86	0.89
ResNet-50	1	1.00	1.00	1.00	1.00	1.00	1.00
	2	0.88	0.75	0.97	0.97	0.99	1.00
	4	0.73	0.75	0.96	0.96	0.98	0.99
	8	0.63	0.56	0.73	0.73	0.98	0.99
	16	0.61	0.43	0.71	0.72	0.98	0.99

when running on i compute nodes and P_1 is the performance in images per seconds when running with a single node. Looking at Table IV we see that the overall efficiency reaches 43% in the worst case, when running with 16 nodes using a real dataset on MareNostrum4. The most remarkable difference in the efficiency is visible on MareNostrum4 when running with 16 nodes with real image set: we can see that AlexNet achieves 94% efficiency, while with ResNet-50 the efficiency drops to 43%. We can also notice that Dibona and Power9 have better scalability, ranging between 70% and 99% when using 16 compute nodes with ResNet-50.

VI. RELATED AND FUTURE WORK

Due to the recent increase of interest for machine learning techniques the literature provides several contributions that are sometimes overwhelming. In this section, we try to highlight the papers that guided our study and, in our opinion, present a similarity with our work.

Shams et al. in [14] already performed a study of HPC clusters with machine learning framework. However, they include accelerators (KNL and GPU), and they do not include considerations about the effects of optimized linear algebra routines.

Our paper follows the idea of Cunha et al. in [15]. Improving the performance using optimized algebra library, in fact, is an implicit way for improving the strong scalability. Also, Sarbu et al. in [16] reinforce our approach since they focus on improving the performance of a sparse matrix kernel, yet another way of improving the performance of the underlying arithmetic libraries.

We took as inspirational work the paper of Sakiyama et al. [17]: we both compare different architectures of HPC clusters, including a scalability study. We complement the work in [17] including Armv8 and IBM Power9 architectures and extending the study to two models, AlexNet and ResNet-50. Even if we do not focus on auto-tuning, the work of Hasabnis [18] helped us in the process of plugging the Arm Performance Libraries into the back-end of TensorFlow.

Finally, for the work on Dibona, our Arm-based platform, we acknowledge the previous work of the Mont-Blanc project [6] as well es the evaluation of more recent Arm-based architectures [19], [20].

On the front of linear algebra library, we are in contact with the developer of the Performance Library in Arm, and we plan to continue our fruitful collaboration for improving the performance of TensorFlow on Arm platforms. As a contribution to the data centers, following our previous experience described in [21], we plan to evaluate TensorFlow in combination with containers technologies for an easier deployment for the users.

VII. CONCLUSIONS

One of the main goals of this paper was to evaluate TensorFlow coupled with an algebra library optimized for Arm. As proof of concept, we plugged the Arm Performance Libraries into some of the most compute-intensive kernels of TensorFlow. We evaluated our optimized version with a synthetic workload and two models, AlexNet and ResNet-50, on a state-of-the-art Arm-based cluster powered by ThunderX2, Marvell's latest Arm CPU, and we measured a speed-up between $1.5\times$ and $2.3\times$.

The second main contribution of this paper was to provide an evaluation of TensorFlow on state-of-the-art HPC clusters with the goal of understanding which are the most effective hardware/software configurations that maximize the efficiency across different architectures.

The first and most relevant observation is that the use of vendor optimized libraries improves the strong scalability on both clusters, x86, and Arm. With MKL on x86, we measure a performance improvement between $1.7\times$ and $6.9\times$.

Concerning parallelization strategies with processes and threads, we noticed that with the Eigen back-end of TensorFlow the best configuration for all architectures and both models is to balance in the mid-range the number of processes and the number of threads (configurations with all threads or all processes are the ones delivering less performance). When using optimized libraries in the back-end, the configurations using more threads and fewer processes delivers overall better performance. This is more evident when using MKL on x86 probably because our modifications for leveraging Arm Performance Libraries was not exhaustive for all the back-end calls.

If we look at the ML tool configuration, we can conclude that, when using optimized libraries with a high number of threads, users must increase the batch size to keep the best performance. So, further optimization is needed for use cases requiring small batch sizes. Also, we are aware that reducing the precision can imply a performance improvement. However we did not explore this corner in our current work: since none of the CPU architectures offered specific reduced precision features, we left this for the future.

Concerning a pure architectural comparison, conclusions are less sharp since our goal was not to find a winner ML CPU architecture. We can, however, observe that the difference in performance between x86 and Arm could come from the size of the SIMD registers: in x86 the AVX512 extension offers registers $4\times$ bigger than the Arm NEON SIMD units. We

expect that the nature of tensor operations can take better advantage of larger SIMD units. Also, MareNostrum4 shows a thread scalability close to ideal up to 24 threads, while Dibona loses efficiency when increasing the number of threads. As a general conclusion for all three HPC clusters, spawning threads across sockets harms the performance. If we analyze the scalability, the overall result is that scalability is better with AlexNet than with ResNet-50, but it is generally good disregarding the cluster architecture/configuration. A way for refining the scalability study could be to focus on the performance of the collective operations within MPI or the I/O performance when using real datasets. However, since we did not want to fine-tune for a given HPC cluster, we preferred to focus in this paper on the optimized back-ends leaving those low-level topics for a future study.

REFERENCES

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[2] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 522–531.

[3] E. Ould-Ahmed-Vall, M. Abuzaina, M. F. Amin, J. Bobba, R. S Dubtsov, E. M. Fomenko, M. Gangadhar, N. Hasabnis, J. Huang, D. Karkada, Y. Jin Kim, S. Makineni, D. Mishura, K. Raman, A. Ramesh, V. V. Rane, M. Riera, D. Sergeev, V. Sripathi, B. Subramanian, L. Tokas, and A. C. Valles, "Accelerating tensorflow on modern intel architectures," 2017. [Online]. Available: http://aim2017.cse.psu.edu/

[4] 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.

[5] Sandia National Laboratory, November 2018. [Online]. Available: https://share-ng.sandia.gov/news/resources/news_releases/top_500/

[6] N. Rajovic, A. Rico *et al.*, "The Mont-Blanc prototype: an alternative approach for HPC systems," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 444–455.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[8] N. A. Gawande, J. A. Daily, C. Siegel, N. R. Tallent, and A. Vishnu, "Scaling deep learning workloads: Nvidia dgx-1/pascal and intel knights landing," *Future Generation Computer Systems*, 2018.

[9] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "Rstensorflow: Gpu enabled tensorflow for deep learning on commodity android devices," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*. ACM, 2017, pp. 7–12.

[10] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[12] Top500, November 2018. [Online]. Available: https://www.top500.org/list/2018/11/

[13] F. Banchelli, M. Garcia, M. Josep, F. Mantovani, J. Morillo, K. Peiro, G. Ramirez, X. Teruel, G. Valenzano, J. W. Weloli, J. Gracia, A. Lumi, D. Ganellari, and P. Schiffmann, "MB3 D6.9 – Performance analysis of applications and mini-applications and benchmarking on the project test platforms," Tech. Rep., 2019, version 1.0. [Online]. Available: https://www.montblanc-project.eu/wp-content/uploads/2019/02/MB3_D6.9_Performance-analysis-of-applications-and-benchmarking-on-the-project-test-platforms.v1.0.pdf

[14] S. Shams, R. Platania, K. Lee, and S.-J. Park, "Evaluation of deep learning frameworks over different hpc architectures," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1389–1396.

[15] R. L. d. F. Cunha, E. R. Rodrigues, M. P. Viana, and D. A. B. Oliveira, "An argument in favor of strong scaling for deep neural networks with small datasets," in *Proceedings of High Performance Machine Learning Workshop, Held in conjunction with IEEE SBAC-PAD 2018 (HPML18)*, 2018.

[16] P.-C. Sarbu and H.-J. Bungartz, "Optimization of a sparse grid-based data mining kernel for architectures using avx-512," in *Proceedings of High Performance Machine Learning Workshop, Held in conjunction with IEEE SBAC-PAD 2018 (HPML18)*, 2018.

[17] K. Sakiyama, S. Kato, Y. Ishikawa, A. Hori, and A. Monrroy, "Deep learning on large-scale multicore clusters," in *Proceedings of High Performance Machine Learning Workshop, Held in conjunction with IEEE SBAC-PAD 2018 (HPML18)*, 2018.

[18] N. Hasabnis, "Auto-tuning tensorflow threading model for cpu backend," *arXiv preprint arXiv:1812.01665*, 2018.

[19] F. Mantovani, E. Calore, F. Mantovani, and E. Calore, "Performance and Power Analysis of HPC Workloads on Heterogeneous Multi-Node Clusters," *Journal of Low Power Electronics and Applications*, vol. 8, no. 2, p. 13, May 2018. [Online]. Available: http://www.mdpi.com/2079-9268/8/2/13

[20] M. Garcia-Gasulla, M. Josep-Fabrego, B. Eguzkitza, and F. Mantovani, "Computational Fluid and Particle Dynamics Simulations for Respiratory System: Runtime Optimization on an Arm Cluster," in *Proceedings of the 47th International Conference on Parallel Processing Companion*, ser. ICPP '18. ACM, 2018, pp. 11:1–11:8.

[21] O. Rudyy, M. Garcia-Gasulla, F. Mantovani, A. Santiago, R. Sirvent, and M. Vázquez, "Containers in hpc: A scalability and portability study in production biological simulations," in *2019 IEEE 33rd International Parallel and Distributed Processing Symposium, in press*. IEEE, 2019.