Technical Report

Official TensorFlow benchmark on Finis Terrae II.

Abstract: One version of the official TensorFlow (TF) performance Benchmark was downloaded from TF webpage and prepared to run on Finis Terrae II (FT2) supercomputing infrastructure from CESGA. This report shows the achieved results on this CESGA supercomputing facilities.

Document Id.:	**2018-002**
Date:	**Nov. 11th 2018**
Responsible:	**Gonzalo Ferro Costas**
Status:	**Final**

Official TensorFlow benchmark on Finis Terrae II.

Galicia Supercomputing Centre

Index

List of Figures

List of Tables

1 Introduction

One version of the official TensorFlow (TF) performance Benchmark was downloaded from TF webpage and prepared to run on Finis Terrae II (FT2) supercomputing infrastructure from CESGA.

2 Resources description.

2.1 Infrastructure description

Finis Terrae II (FT2) is a computation system based on Intell Haswell processors interconnected by an InfiniBand FDR network with a peak performance of 328 TFlops and a held performance of 213 TFlops in the Linpack benchmark. FT2 is a live computing platform (Figure 1) that is evolving continuously, integrating more resources frequently. During the experiment, FT2 had a configuration with 4 types of computation nodes:

1. Thinnodes (306 nodes): with 2 Haswell 2680v3 processors (up to 24 cores and 2.5Ghz), 128 GB of RAM memory and a local hard disk of 1 TB.
2. GPUs Nodes (4): with 2 Haswell 2680v3 processors, with 2 GPUs NVIDIA Tesla K80 (driver version 375.26) and 128 GB of RAM memory and a local hard disk of 1 TB.
3. Xeon Phi Nodes (2): with 2 Haswell 2680v3 processors (up to 24 cores), 2 Intel Xeon Phi 7120P, 128 GB of RAM memory and a local hard disk of 1 TB.
4. FAT node (1): with 8 Intel Haswell 8867v3 processors (totally, 128 cores), 4096 GB of RAM memory, 24 local hard disk of 1.2 TB and 2 SAS hard disk of 300 GB.

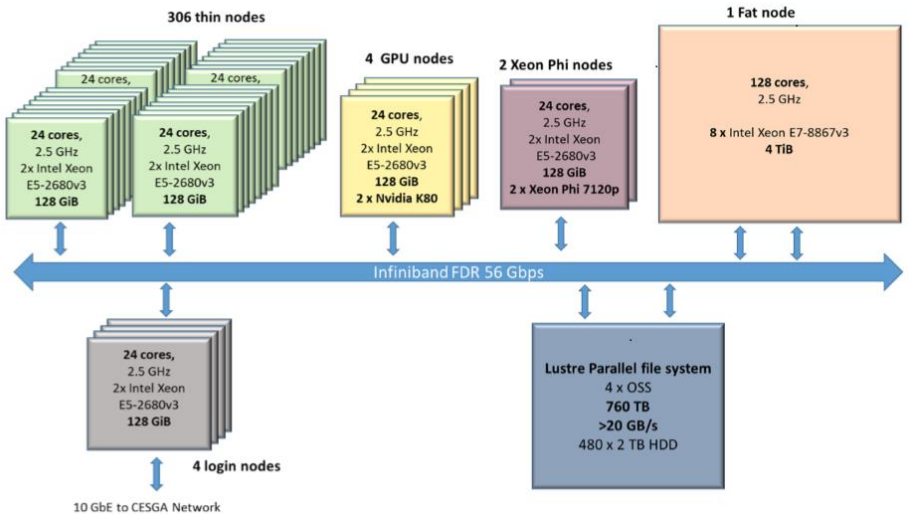

Figure 1. Basic description of FT II supercomputer.

In addition to the compute nodes, FT2 have 4 nodes for login to the system and to transfer files with 2 Haswell 2680v3 processors (up to 24 cores), 128 GB of RAM memory and 2 hard disk of 1 TB.

All nodes shared a high performance file system based on Lustre, which is connected through InfiniBand. This storage infrastructure has 760TB of storage (400 disks of 2TB each) with is server with 4 OSS to achieve a bandwidth higher than 20GB/s.

The execution of jobs in FT2 is managed by Slurm (version 14.11.10-Bull.1.0), which guarantees a

correct assignment of resources to each job. FT2 has the capability of Intel Turbo Boost of processors disabled, so the frequency of each assigned core is configured to the nominal one at the beginning of each job. The performance studies for this technical report were carried out on the Thinnodes. The Operative System of these nodes is Linux and the distribution is Red Hat Enterprise Linux Server release 6.7.

2.2 Machine Learning API.

In FT2 there are several versions of TF available:

- TensorFlow 1.0.0
- TensorFlow 1.2.1
- TensorFlow 1.3.1
- TensorFlow 1.7.0

Firs three versions were compiled with and without GPU support, using gcc version 4.9.1, python based on Anaconda2, cuda 7.5 and cuDNN 5.0. In the case of TensorFlow 1.7.0 there are three different compiled versions: 2 of them were compiled with Python 2.7.14 with GPU support (using cuda 9.1.85 and cuDNN 7.1.3) and without it. The other version was compiled with Python 3.6.5 without GPU support. These tree versions of TensorFlow 1.7.0 were compiled using gcc version 6.4.0.

To execute the official Benchmarks, TensorFlow 1.7.0 without GPU support and compiled with Python 2.7.14 was used.

2.3 Benchmark description.

TF official Benchmark scripts were downloaded from TF github. *cnn_tf_v1.5_compatible* [1] branch was downloaded. This benchmark is a complete suite of Python and TF code that allows user to test several different image classification models like:

1. ResNet-50
2. ResNet-152
3. InceptionV3
4. VGG16
5. AlexNet

Additionally, the benchmark suit allows user to test several training hyperparameters (like batch size or learning rate), different optimizers and great variety of high-performance trainings (different types of distributed training with GPUs or CPUs or very detailed controlled of the use of MKL libraries in order to take advantage of several CPUs parallelism…).

2.4 Benchmark modification.

When several tests were performed on the FT2 some issues concerning to the results of the benchmark when used in distributed mode (when no GPUs are used) were discovered:
TF Benchmark calculates the images per second (this is the performance metric of the benchmark) using equations **(1)** and **(2)**:

$$average_{wall\ time} = \frac{\text{Elapsed Time}}{\text{Number Of Steps}} \quad (1)$$

$$images\ per\ second = \frac{Number\ Of\ Workers \times Batch\ size}{average_{wall\ time}} \quad (2)$$

Where Elapsed Time is total training time measured by the benchmark, number of steps are the total number of batches done during training (measured again by the benchmark) and batch size is the number of images processed in each training step.
The problem with this calculation arises when distributed mode is using for training because the number of steps measured by the benchmark is the total number of steps performed by all the workers: so in distributed training over pure CPUs machine this calculation give to bad results (the images per second calculated were much higher than the real one).
Equations **(3)** and **(4)** are the proposed ones for calculating the image per second metric:

$$Number\ Of\ Total\ Images = Number\ Of\ Steps \times Batch\ Size \quad (3)$$

$$images\ per\ second = \frac{Number\ Of\ Total\ Images}{\text{Elapsed Time}} \quad (4)$$

Additionally, it should be noticed here than in posterior versions of official TF Benchmark this calculation approach is the used one.
This calculation was added to the correspondent script (benchmark_cnn.py) of the TF benchmark suite in order to use for getting properly results when distributed training is used.

2.5 Model Benchmark selection.

All the benchmark tests performed in this report were done with the general parameters showed in Table 1.

Parameter	Value
MODEL	ResNet-50
Batch Size	64
Training Data	Synthetic
Warmup Steps	10
Training Steps	100

Table 1. General parameters for model and training configuration.

2.6 Benchmark test procedure.

The benchmark test developed in this report the following protocol was used:

1. For each proposed test the benchmark model (defined in 2.5) was trained increasing the number of cores or tasks (this is explained in section 3 in each proposed test).
2. For each number of cores or tasks used 5 repetitions were executed.
3. For each repetition the images per second using Equations **(3)** and **(4)** was calculated.
4. For each number of used cores or tasks the average over the 5 repetition of the images per second is calculated.

5. For each number of used cores or tasks the difference between the maximum and minimum value of the images per second of the 5 repetitions will be used as measurement error.

Different tests were proposed and performed. These tests and the benchmark results using this testing protocol is presented in next Section.

3 Benchmark tests.

Main objective for the benchmark tests of this report is evaluate the Distributed TF capabilities over the Finis Terrae 2 infrastructure.

As a first approximation an extensive benchmark test using non-distributed training were performed in order to get some insight for preparing allocation resources for Distributed ones. However, the non-distributed test results are by self very interesting and can offer several interesting conclusions that users can use to do a proper resources allocation for their own jobs.

3.1 Non-Distributed Training Benchmark.

TF allows user to take advantage of multi-core machines by exploiting parallelization offered by the intel MKL libraries.

These capabilities are available in the official TF Benchmark and user can activate them. User can change the related MKL options by using following commands:

- --*num_inter_threads* : number of threads used for inter-op parallelism (recommendation is to set to the number of sockets of calculation machine)
- --*num_intra_threads* : number of threads used for intra-op parallelism (usually is recommended to set it to the number of cores you have available for calculation)
- --*mkl* : if true allow user to set MKL environment variables
- -- *kmp_blocktime* : time (ms) that a thread should wait after completing the execution of a parallel region (only with mkl=True, it is recommended set it to 0)
- --*kmp_affinity* : is a string that allow user to set several MKL parameters (only with mkl=True, by default: granularity=fine,verbose,compact,1,0).

Setting aforementioned parameters in a good way will have dramatic impact in time performance of the training as will be see.

In the case of non-distributed training the parallelization capabilities are based on the MKL libraries so in this case for all the benchmark tests under this section were performed using a complete FT 2 “thinnnode” and the number of cores dedicated to the training were increased.

We will split this set of tests in 2 types:

1. Setting *mkl* to False. In this case all the MKL environment variables will be set by the queue system by default.
2. Setting *mkl* to True and *kmp_blocktime*=0. In this case benchmark will overwrite MKL environment variables. Only MKL variable *kmp_blocktime* will be set to 0.

3.1.1 --mkl=False.

In this case the environment variables related with MKL libraries will be defined by the FT2 queue system.

Three kinds of test setting *num_inter_threads* parameter to 0,1 and 2 were done.

3.1.1.1 *--mkl*=False, *--num_inter_threads* =0.

When *num_inter_threads* is set to 0 then system will pick and appropriate number (this is the default behaviour of the benchmark suite). Three benchmarks tests were performed with *num_inter_threads*=1 and *num_intra_threads* equal to 0 (again system pick an appropriate number for this parameter), 1 (this is the suite default value for this parameter) and number of cores used in each test.

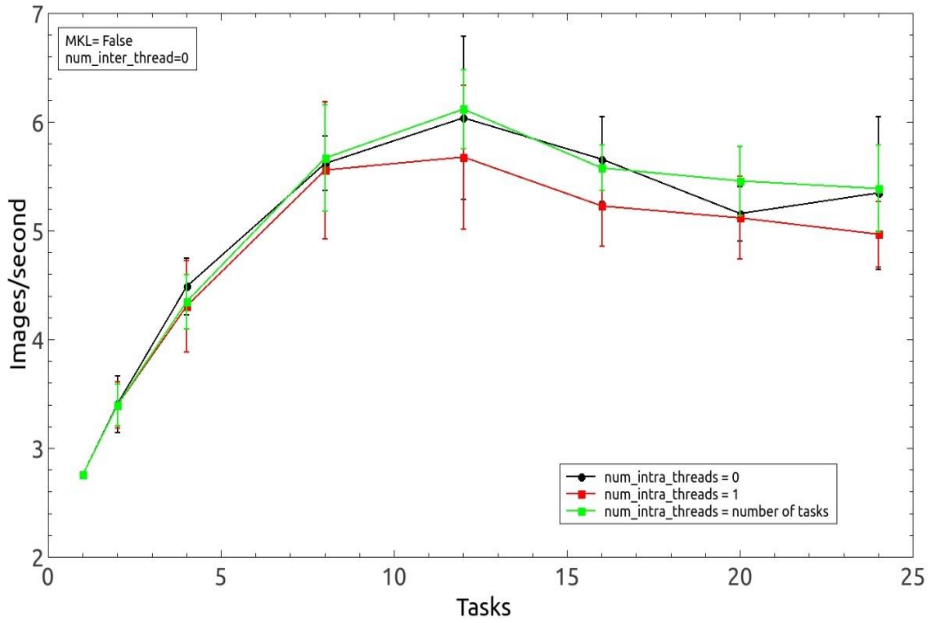

Figure 2. Image per second vs number of cores used. MKL=False. num_inter_threads=0. num_intra_threads= 0 (Black symbols), 1 (red symbols) and number of cores (green symbols).

Results of the benchmark test are showed in Figure 2. As can be seen setting *num_intra_threads* to 1 has the lowest time performance when number of cores is higher than 8. There is not difference between to set *num_intra_threads* to 0 or setting to the correspondent number of cores. In this case highest Image per second metric is near 6 and it is obtained when number of cores is 12.

3.1.1.2 *--mkl*=False, *--num_inter_threads* =1.

num_inter_threads equal to 1 is the default value in the benchmark suite. Three benchmarks tests were performed with *num_inter_threads* s=1 and *num_intra_threads* equal to 0 (again system pick an appropriate number for this parameter), 1 (this is the suite default value for this parameter) and number of cores used in each test.

Results of the benchmark test are showed in Figure 3. In this case there is not difference between *num_intra_threads* set to 0 or to the number of cores. Again, setting *num_intra_threads* to 1 is the worst time performance option.

This case is very interesting because best performance results are obtained when maximum number of cores (24) are involved in the training. So when *num_inter_threads* equal to 1 full time performance from a complete "thinnode" can be obtained (in the case of *num_inter_threads*=0 best performance was obtained at half a node). Additionally, the highest image per second rate is 28.9 (this value is more than 4 times higher than the highest value when *num_inter_threads*=0).

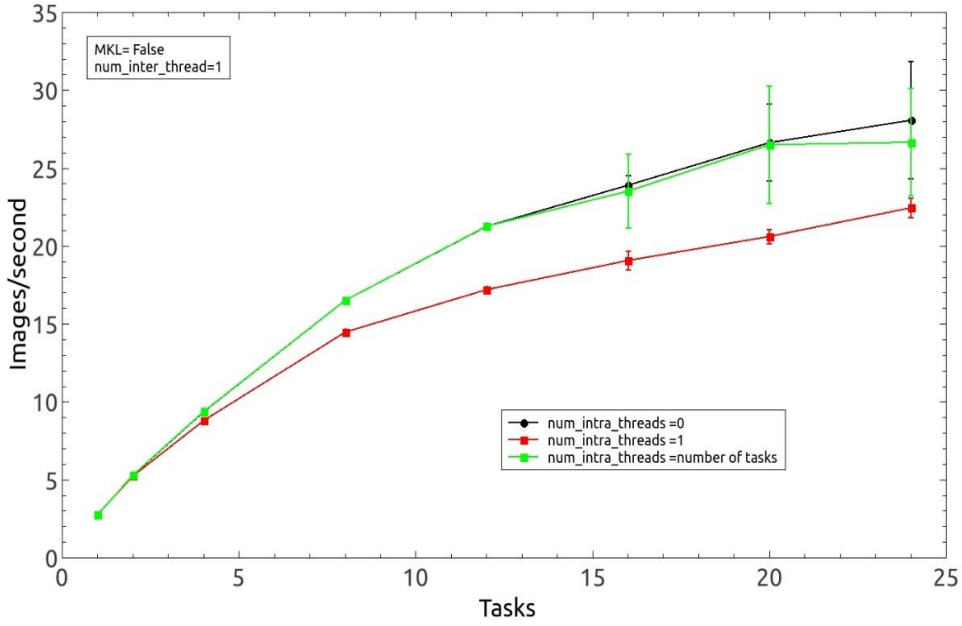

Figure 3. Image per second vs number of cores used. MKL=False. num_inter_threads=1. num_intra_threads= 0 (Black symbols), 1 (red symbols) and number of cores (green symbols).

3.1.1.3 *--mkl*=False, *--num_inter_threads* =2.

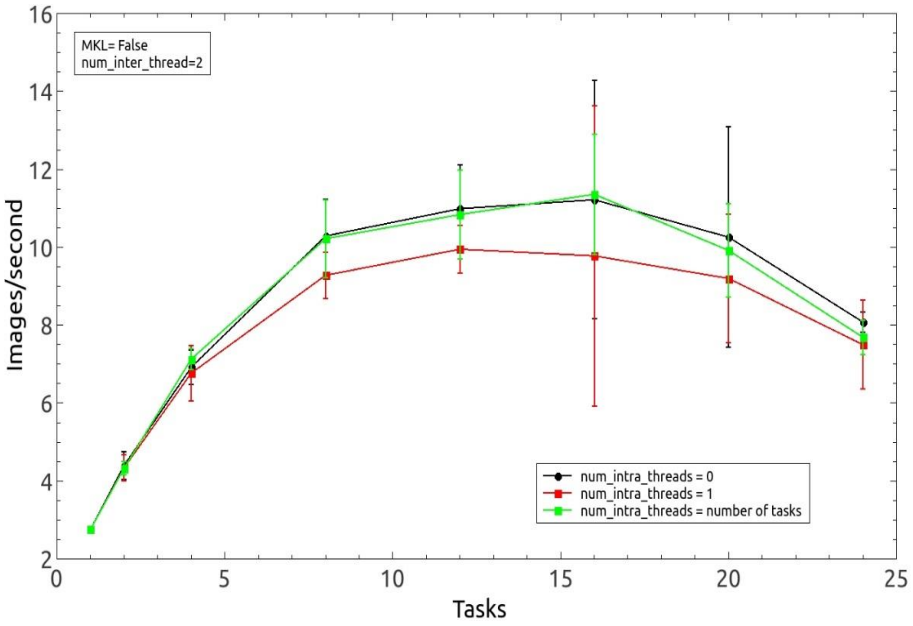

Figure 4. Image per second vs number of cores used. MKL=False. num_inter_threads=2. num_intra_threads= 0 (Black symbols), 1 (red symbols) and number of cores (green symbols).

num_inter_threads equal to 2, in theory, is the recommended value for "thinnodes" because usually it is recommended setting it to the number of the sockets of the machine. Three benchmarks tests were performed with *num_inter_threads*=1 and *num_intra_threads* equal to 0 (again system pick an appropriate number for this parameter), 1 (this is the suite default value for this parameter) and number of cores used in each test.

Results of the benchmark test are showed in Figure 4. Again, setting *num_intra_threads* to 1 is the worst option and there is no difference setting it to 0 or to number of cores. In this case best value, 11.4 images per second, arise when 16 cores are used.

3.1.1.4 *--mkl*=False, *--num_inter_threads*=number of tasks.

One last test was done setting the *num_inter_threads* equal to the number of tasks and setting *num_intra_threads* to number of tasks.

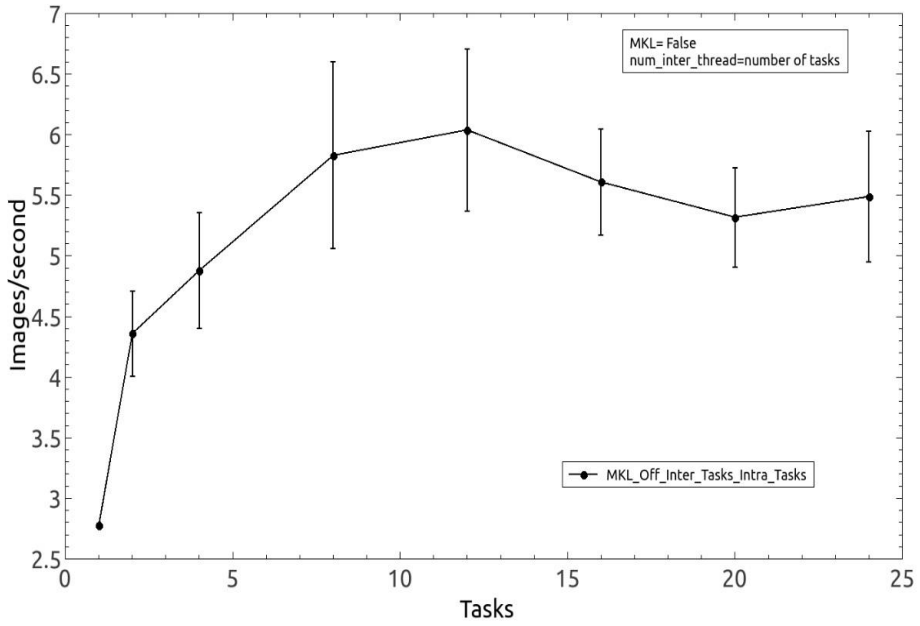

Figure 5. Image per second vs number of cores used. MKL=False. num_inter_threads=number of tasks and num_intra_threads=number of tasks.

As can be seen in Figure 5 this configuration best images per second value is 6.0 and is obtained when 12 cores are used.

3.1.1.5 *--mkl*=False, *--num_inter_threads*=comparative.

Table 2 presents highest images per second and number of cores needed to achieve it for the 4 tested configurations of the *num_inter_threads* parameter.

num_inter_threads	*num_intra_threads*	Highest Images per second	cores
0	Number of cores	6.1	12
1	0	28.1	24
2	Number of cores	11.4	16
Number of cores	Number of cores	6.0	12

Table 2. Bests configuration performance for the different *num_inter_threads* s tested.

As can be seen best performance is obtained when using *num_inter_threads* equal to 1 and with *num_intra_threads* equal to 0.

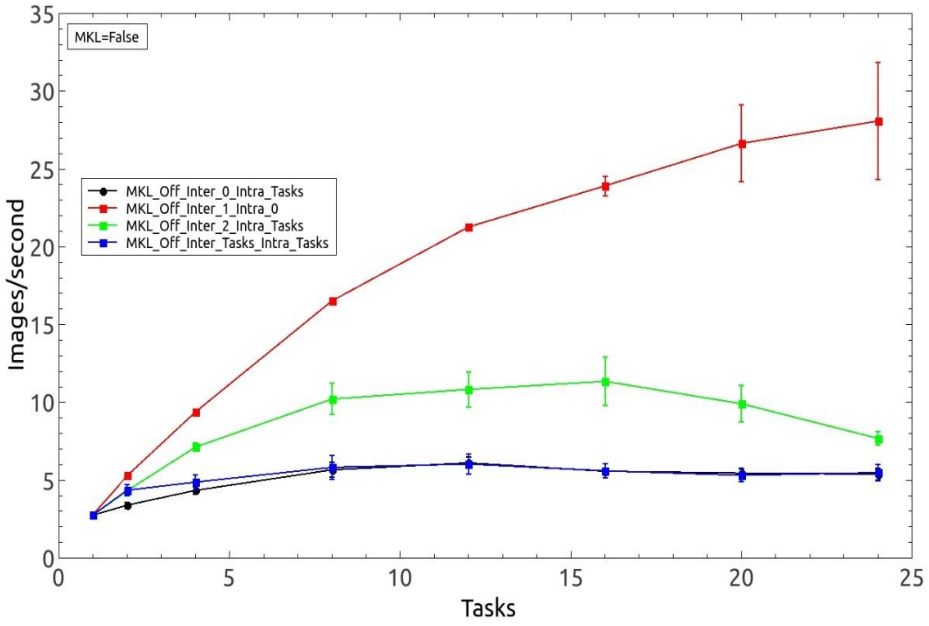

Figure 6. Image per second vs number of cores used. MKL=False. Comparative of benchmark results of the best results for each num_inter_threads parameter tested: 0 (Black symbols), 1 (red symbols), 2 (green symbols) and number of cores (blue symbols).

Figure 6 presents the Images per second vs the number of cores for the configurations presented in Table 2. As can be seen setting *inter_num_threads* to 1 result in best performances for all the number of cores tested: all the computational power of the whole node (24 cores) is exploited and highest images per second rate are obtained (until 4 times higher rates can be achieved!!).

3.1.2 --mkl=True.

In this case the environment variables related with MKL libraries can be defined by the user. For do it that user must set the *–mkl* parameter of the benchmark to True, and then can give values to *kmp_blocktime* and *kmp_affinity*.

For the tests with *mkl* fixed to True *kmp_blocktime* was set to 0 and *kmp_affinity* was fixed to the benchmark suite value. The *intra_num_threads* was fixed to the number of correspondent cores and different values of *inter_num_threads* were tested: 0,1 and 2.

Figure 7 presents the Images per second vs the number of cores for the three different *inter_num_threads* tested: 0 (black symbols), 1 (red symbols) and 2 (green symbols). As can be seen best results are obtained when *inter_num_threads* is 1 or 2 (in a strict sense best performances are obtained with 2 but the error make two values optimal options in term of time performance). With this configuration full node is totally exploited and the number of images per second can be raised to 33.5.

So controlling the MKL environment variables can help to increase even more the performance metric in the TF benchmarks (best image per second with MKL=False was 28.1).

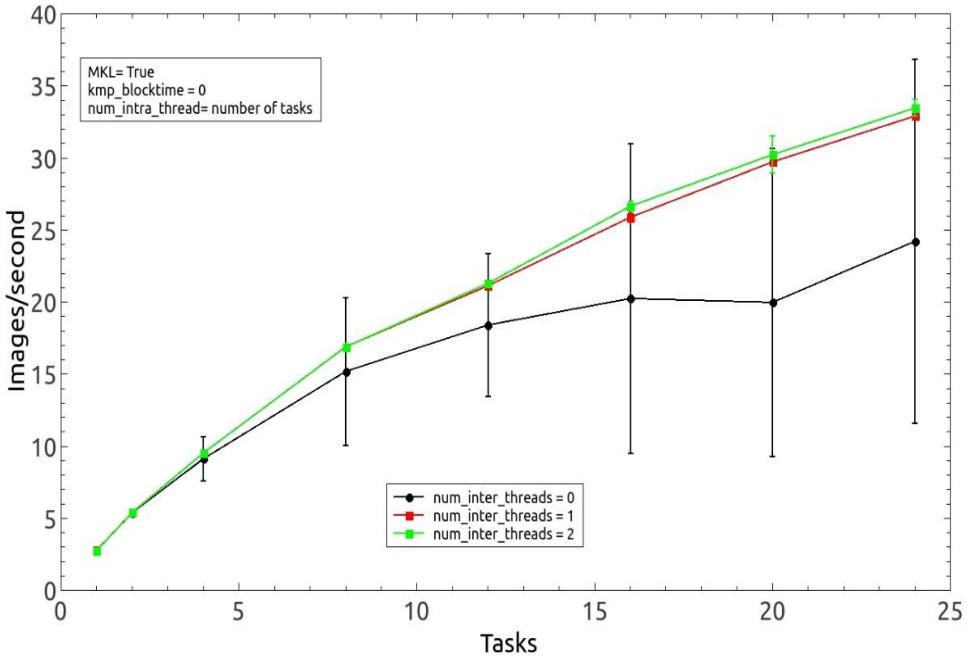

Figure 7. Image per second vs number of cores used. MKL=True, kmp_blocktime=0, kmp_affinity by default and intra_num_threads= number of cores. num_inter_threads= 0 (Black symbols), 1 (red symbols) and 2 (green symbols).

Additionally, using *mkl*=True and fixing *inter_num_threads* to 1 or 2 give more stable times. As can be seen from Figures 2 to 6 the error bars (that are the difference between the maximum and the minimum images per second of the 5 repetitions) in several measurements are really big. But in the case of Figure 7 the error bars are very small (for *inter_num_threads* equal to 1 or 2 of course) so the metric performance measurements are more similar.

3.1.3 Relevance of non-distributed tests.

The benchmark test presented in before sections are very important and provide a very useful insight about the TensorFlow scalability.

As have been demonstrated time performance of the training can be very different when different configurations are used and properly set of a configuration can increase until a factor 5 the training time performance in TensorFlow when deployed in FT2 “thinnodes”.

Especially important are the variables *num_inter_threads* and *num-intra_threads* because TF users can add them to their code in an easy way by using *tf.ConfigProto* and setting properly ***inter_op_parallelism_threads*** and ***intra_op_parallelism_threads*** variables. This *tf.ConfigProto* can be attached to the *tf.train.MonitoredTrainingSession* function where the training is done.

3.1.4 Real World.

The one node tests from before subsections were done using the benchmark developed by TF developers. So there are implemented a lot of speed up tricks that TF developers know very well but standard TF programmers maybe do not know how implement in their code. So question arises: in an standard TF program this configurations of the *tf.ConfigProto* can be relevant or not.

In order to test this an example of typical code for MNIST classification from TF webpage[1] was downloaded and tested. Before doing tests several modifications on the code were done in order to use the *tf.train.MonitoredTrainingSession* for do the training and can included the *tf.ConfigProto* configuration in an easy way.

In this case not so extensive tests could be done so qualitative results will be provided.

3.1.4.1 *--mkl*=False, *--num_inter_threads*=comparative.

In this case the environment variables related with MKL libraries will be defined by the FT2 queue system. Three kinds of test setting *num_inter_threads* parameter to 0,1 and 2 were done. The number of *num_intra_threads* was set to 0 (so system pick an appropriate value).

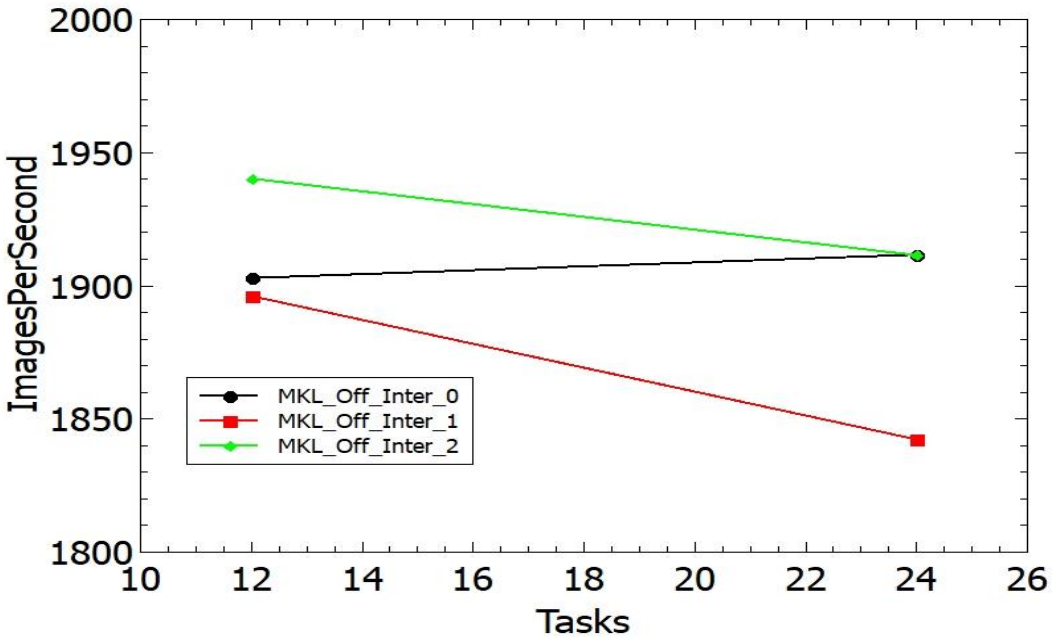

Figure 8. Image per second vs number of cores used for real world MNIST classification. MKL=False. Comparative of benchmark results of the best results for each *num_inter_threads* parameter tested: 0 (Black symbols), 1 (red symbols), 2 (green symbols).

Figure 8 shows images per second in function of number of cores (only for 12 and 24 cores) for the three values of *num_inter_threads:* 0 (black symbols),1 (red symbols) and 2 (green symbols). As can be seen with our real-world problem there is no such difference between selecting *num_inter_threads* and there

[1] https://github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/examples/tutorials/mnist/mnist_deep.py

is not improvement between using 12 or 24 cores (in general is worse to use 24 cores instead of 12 cores). This Figure should be compared with Figure 6. Reader should be aware that comparisons are not fair because in the benchmark case the Convolutional Network is much more complex and bigger than in our real-word case. However, the comparison is good for understanding that TF programs can be very complex, and an initial performance test of the code is always recommended in order to obtain better performance from FT2 infrastructure.

3.1.4.2 *--mkl*=True, *--num_inter_threads*=comparative.

In this case the environment variables related with MKL libraries were defined by the user: *kmp_blocktime* was set to 0 and *kmp_affinity* was fixed to "granularity=fine,verbose,compact,1,0" (this is equivalent the benchmark suite default value). The *intra_num_threads* was fixed to 0 and different values of *inter_num_threads* were tested: 0,1 and 2.

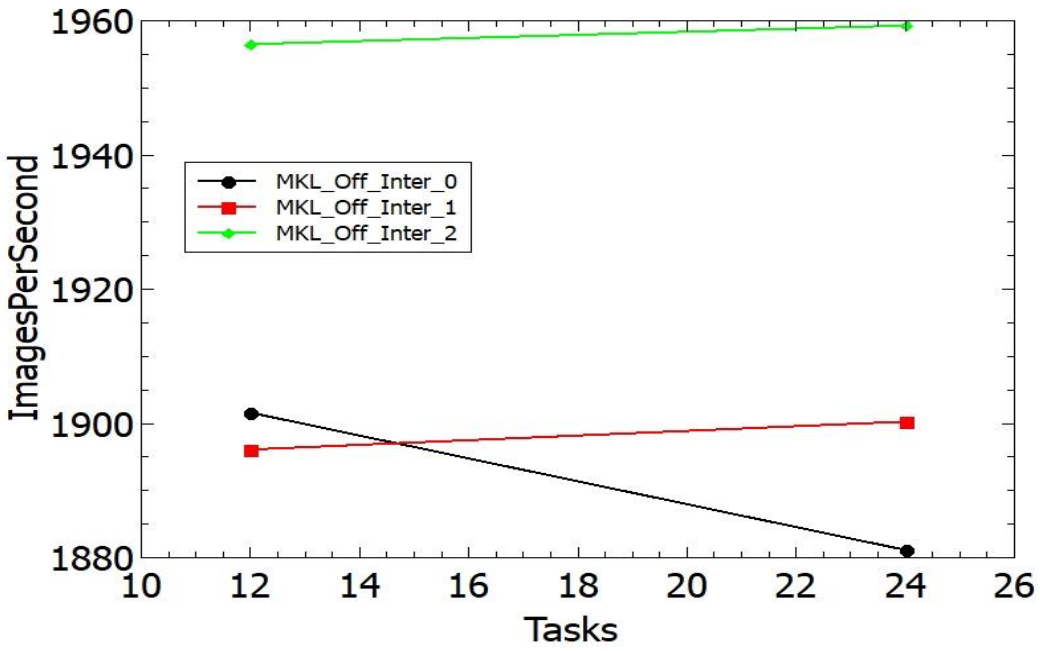

Figure 9. Image per second vs number of cores used for real world MNIST classification. MKL=True. Comparative of benchmark results of the best results for each num_inter_threads parameter tested: 0 (Black symbols), 1 (red symbols), 2 (green symbols).

Figure 9 shows images per second in function of number of cores (only for 12 and 24 cores) for the three values of *num_inter_threads:* 0 (black symbols),1 (red symbols) and 2 (green symbols). As can be seen in the figure tuning the num_inter_threads could be increase performance of the training, but results are less dramatic than in the benchmark suite. Figure 9 should be compared with Figure 7: in both cases setting num_inter_threads=2 gives best time performance results. Again, there is not a benefit between use 12 or 24 cores for training.

Reader should be aware that comparisons are unfair due to the convolutional network model to train and that in the benchmark several tricks for time performance are implemented. As in the before subsection recommendation is always make an initial and simple time performance test by selecting num_inter_threads or fixing the corresponding environment variables.

3.2 Distributed Training Benchmark.

The official Benchmark was tested using distributed training capabilities of TF on the FT2. For this the python package tf4slurm was used.

Using the study and conclusion of Section 3.1 following options for distributed training were used:

- --*mkl*: true (user set MKL environment variables)
- -- *kmp_blocktime*: 0
- --*kmp_affinity*: default value (granularity=fine,verbose,compact,1,0)
- --*num_inter_threads*: 1
- --*num_intra_threads*: number of cores (in the test this value was set to 24)

For the distributed test the number of distributed tasks were increased from 2 tasks to 10 and the measurement protocol (see 2.6) was used. For all the trainings the number of Parameter Servers was always 1. And 24 cores (on complete FT2 thinnode) for each task (Parameter Server and Workers types) was used. Table 3 shows the results for Distributed TF training for the benchmark suite. Reader should be aware that number of tasks and nodes in this test. Only worker tasks do real computation so for 2 tasks (2 nodes) results are similar to the non-distributed trainings (because Distributed TF always need at least one Parameter Server so in fact with 2 tasks we are only executing a complicated version of non-distributed training). For this reason the analysis of the scalability is done in function of the number of workers and not in function of the number of tasks.

Tasks/Nodes	Workers	Mean	Error	Speed Up
2	1	31.0	0.6	1.00
3	2	61.1	0.9	1.97
4	3	90.3	3.0	2.91
5	4	118.8	1.9	3.83
6	5	146.0	3.3	4.71
7	6	171.3	5.2	5.52
8	7	198.4	1.6	6.40
9	8	224.1	3.5	7.23
10	9	247.3	4.9	7.97

Table 3. Benchmark results for Distributed TF Training.

Speed Up was calculated following:

$$Speed\ Up = \frac{ImagesPerSecond(N_{Workers})}{ImagesPerSecond(1)} \quad (5)$$

Were:

- ImagesPerSecond($N_{Workers}$) is the images per seconds for N workers
- ImagesPerSecond(1) is the images per second when 1 worker is used.

Figure 8 shows the mean number (with errors bars) of the number of images per second (left axis and black symbols) vs the number of workers of the training. Additionally, the correspondent Speed Up (right

axis and red symbols) is plotted too. As can be seen the scalability is nearly linear and almost perfect (compare with the dashed blue line that correspond with ideal speed up).

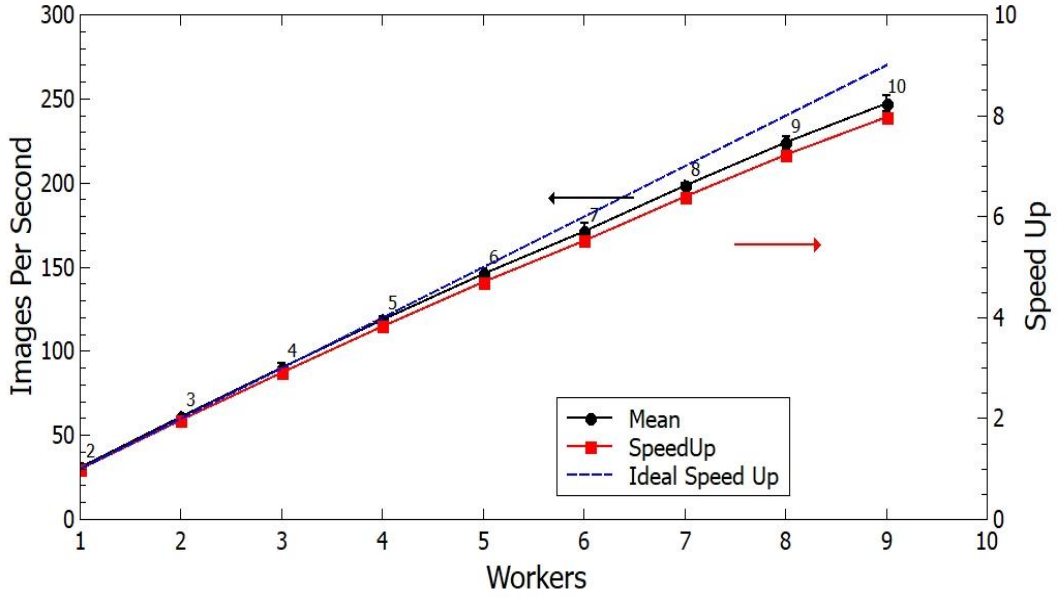

Figure 10. Images per second (left axis and black symbols) and correspondent Speed Up (right axis and red symbols) vs number of workers for Distributed TF training. Dashed blue line is ideal speed up and labels are the number of tinnodes used.

Obtained benchmark results allows us to compare FT2 with official benchmark results presented in TF webpage[2]. This comparison is presented in Figure 9.

[2] https://www.tensorflow.org/guide/performance/benchmarks

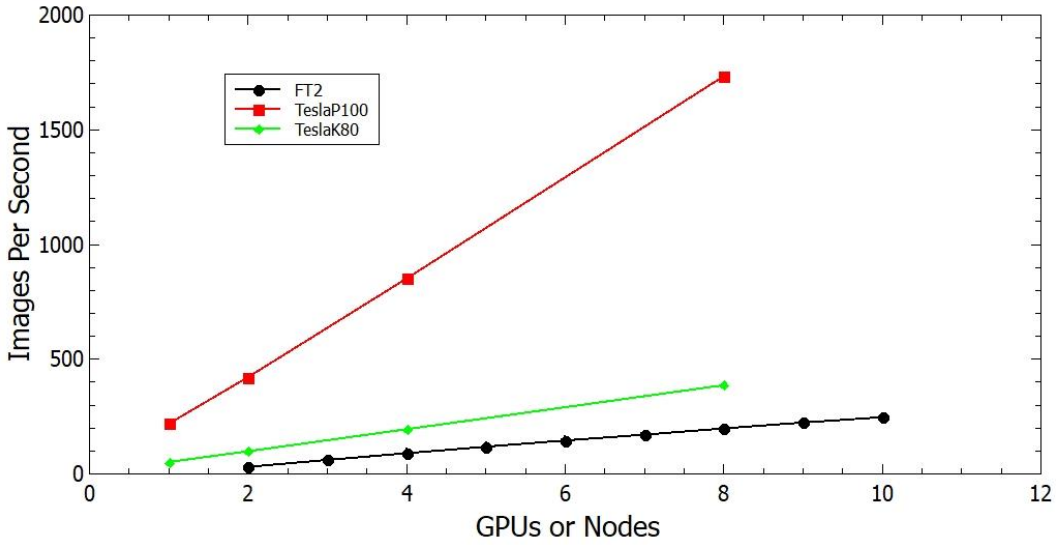

Figure 11. Comparative benchmark results among CESGA FT2 (black symbols), TeslaP100 (red symbols) and TeslaK80 (green symbols).

4 Conclusions.

In this report several tests with an original TensorFlow benchmark were done on the Finis Terrae II CESGA infrastructure. Two kinds of tests were done:

1. Non-Distributed Training: in this case several configurations were tested in order to have a better insight of how different tuning options affects to the scalability.
2. Distributed Training: using the tf4slur package a complete test of distributed training capabilities was conducted. The results obtained here can be compared with the benchmark results presented in the TF webpage

Acknowledgements

The work presented in this report has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680481 (Fortissimo 2 project). These results reflect only the author's view and that the Commission is not responsible for any use that may be made of the information it contains. Authors want to thank also to AIMEN for the provision of the data and CESGA for the access to Finis Terrae II HPC infrastructure.

5 References

[1] TensorFlow Benchmarks GitHub, «TensorFlow Benchmarks GitHub cnn_tf_v1.5_compatible,» [En línea]. Available: https://github.com/tensorflow/benchmarks/tree/cnn_tf_v1.5_compatible/scripts/tf_cnn_bench

marks.