Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

Machine Learning: analysis, optimizations and evaluation of modern frameworks on HPC clusters

30 June 2020

Autor:
Guillem Ramirez-Gargallo

Director:
Filippo Mantovani

Ponent:
Jesus Labarta

Co-Director:
Marta Garcia-Gasulla

Specialization:
Computer Engineering

Company:
Barcelona Supercomputing Center

Contents

1 Introduction

In recent years, Machine Learning (ML) becomes an essential technology in a variety of fields. ML techniques are automated methods to detect patterns [1]. These algorithms behind ML transform our world from shopping to health care. ML constitutes the technical background pushing forward research and business fields [2] such as autonomous driving [3] and personalized medicine [4]. ML can process a significant amount of data helping us to classify or predict events. ML also can help when do not have sufficient knowledge of a particular domain, but these algorithms can help us to *build* a good approximation of these particular domain process [5].

ML adoption has been pushed forward by large companies who developed open-source "frameworks" that facilitate to regular users the solution of a class of problems using ML. This frameworks implement generic ML routines and algorithms in a generic fashion but also with the cooperation of the users and machine vendors that helps to implement specific routines to use more efficiently in specific CPUs and accelerators like GPUs. Nowadays, the ML models has grow in terms of memory usage and many of this accelerators does not have the sufficient memory to contain the problem. For this reason we will think in CPU as a real alternative for training big ML models.

Due to this TFG is perform modality B, develop in a company, we present Barcelona Supercomputing Center(BSC) that is company where this project is develop. The BSC is the national supercomputing centre in Spain and is specialized in high performance computing (HPC) It manage MareNostrum 4, one of the most powerful supercomputers in the world. BSC provide HPC resources to the national and international scientific and industrial community and of industry. The centre manages the Red Española de Supercomputación (RES). Also is part of the Partnership for Advanced Computing in Europe (PRACE). BSC have a total staff of more than 650 R&D experts and professionals split in 5 departments: Computer Sciences, Life Sciences, Earth Sciences, Operations and Computer Applications in Science and Engineering. During by stay I involved on the Mobile and embedded-based HPC group.

Since most data centers does not want to specialize their hardware for specific tasks like ML. HPC data centers are traditionally populated with x86 architectures, but recently the trend is changing: the first ranked supercomputer of the Top500 is indeed Fugaku, a Fujitsu A64FX CPU with Arm architecture. And the second one is Summit, an IBM PowerPC architecture (boosted by NVIDIA GPUs). 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 TFG are: *i)* develop, build and evaluate a prof-of-concept of TensorFlow version with Arm Performance Libraries; *ii)* the study of performance evaluation of hybrid configurations of TensorFlow in three different platforms; *iii)* the study of scalability of TensorFlow on multiple nodes of three platforms with different architecture.

This document is organized as follows: In Section 2 we introduce the background concepts needed to understand the technical work of this TFG. Section 3 summarizes the motivation behind the research of this TFG. In Section 4 we report the planning of the TFG project including budget and time organization of the tasks performed during the 4 months of project development. Section 5 contains the bulk of the technical work performed for deploying an efficient ML framework on an Arm-based cluster. Section 6 reports the evaluation that we performed on 3 different state-of-the-art HPC clusters. In Section 7 we wrap up with some conclusion remarks.

2 Background

2.1 Parallel systems

In this section, we will describe the parallel systems. In a typically modern homogeneous cluster, we have a group of cores that are inside a socket. A node contains a single or multiple sockets, and typically the cores in these sockets shared the memory. Then the nodes connect with a fast network like InfiniBand. We can see a visual representation of typical modern cluster in figure 1.

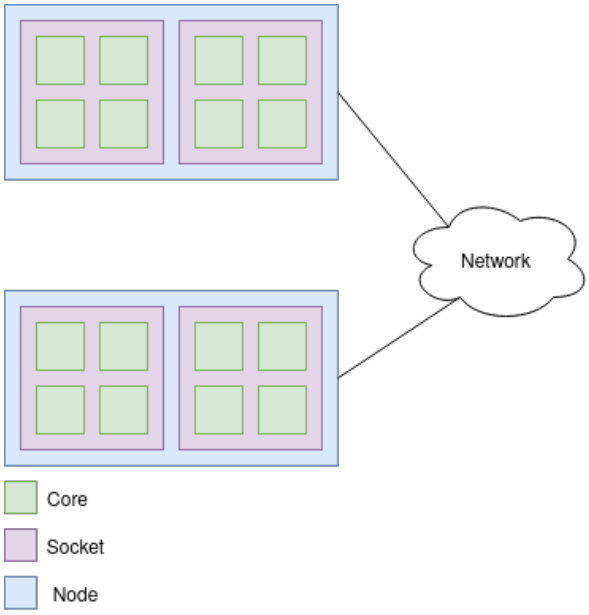


Figure 1: Diagram of typical homogeneous modern cluster

2.2 Parallel programming models

In this section, we present two of the most used programming models in HPC. A parallel programming model is a tool that helps the developer to program a parallel system. We can classify parallel programming models in 2 types, distributed memory and shared memory.

2.2.1 Distributed memory parallel programming model

Distributed memory parallel programming models are usually based on processes that do not share memory spaces. For this reason they usually need explicit communication. This characteristic allows us to execute the same program in multiple nodes. The standard de facto distributed parallel programming model in HPC is the Message Passing Interface(MPI).

MPI is a specification[1] of a distributed memory parallel programming model based on message passing. In this message passing model, the exchange of data is performed through exchanging messages that can be synchronous or asynchronous. Each process or rank has their own logical memory space addresses that normally is hidden to the other processes.
MPI is simply a specification, there are many implementations: ones are generic (Open Source or Proprietary) or Vendor Specific. Examples of implementations are OpenMPI, MVAPICH, MPICH and IMPI(Intel).
In Figure 2, we can see two MPI process communicating using synchronous calls. The first MPI Rank send data to the second process then the second process send to the fist one.

[1] https://www.mpi-forum.org/docs/

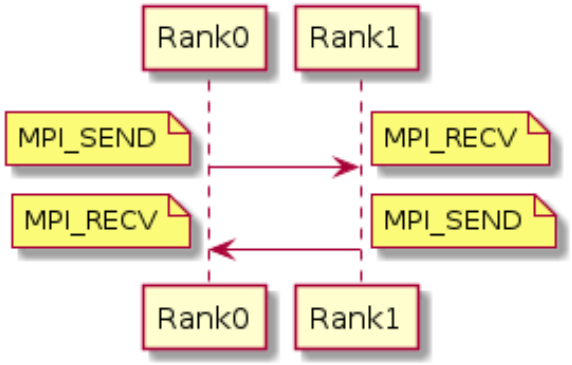


Figure 2: MPI example synchronous calls

2.2.2 Shared memory parallel programming model

A shared memory parallel programming model is based typically in threads that share the memory space, for this reason the communication is usually implicit. The standard de facto shared memory parallel programming model in HPC is Open Multi-Processing(OpenMP).

OpenMP is a parallel programming model for shared memory systems. It uses pragmas to to annotate the code to indicate how to create the threads and split the work. Pragmas are recognized by compilers, this allows that the same source can be compiled and is correct with and without OpenMP support. The pragmas start with `#pragma omp`. It also offers an API, for example, to get the number of threads in a parallel region using `omp_get_num_threads()`.
OpenMP uses a fork-join model of thread management. This fork-join model is based on creating a group of threads when the master thread reacts to a user-defined parallel region. This threads will be a copy of the master thread and will have the same variables that the master thread until the parallel region. To indicate a parallel region, we use directive `#pragma omp parallel`. In figure 3 we can see a representation

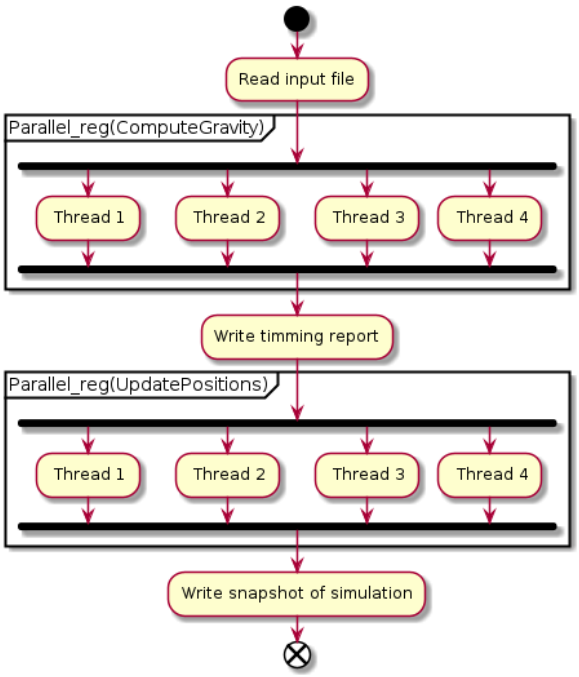


Figure 3: OpenMP example parallel region

In OpenMP we can defined variables that are shared above the treads and others that are private for each thread. This characteristic allow us to structure the code to operate some data at the same time and also make one variable to have local instance for each thread. Asynchronous concurrent access can lead to race conditions, and mechanisms such as locks, semaphores and monitors can be used to avoid these.

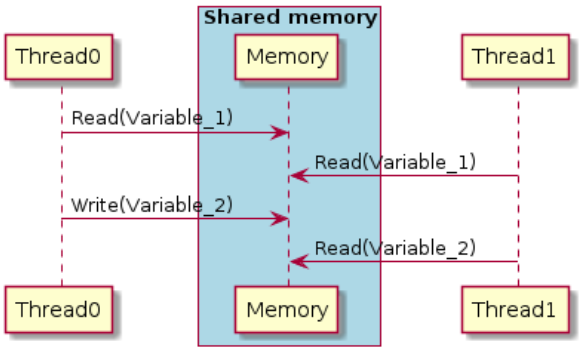


Figure 4: OpenMP example access same memory

2.3 Linear algebra libraries

In HPC and AI world is common to use the same routines to process the data. For this reason, the community create libraries to create a generic, tested and efficient routines for the most common operations. Mathematical linear algebra libraries are ones or the most important libraries in HPC and have generic and vendor-specific implementations.

2.3.1 Eigen

Eigen[2] is a C++ lineal algebra library. Eigen implements his algorithms in a high-level language in standard C++ language, making the library portable and compiler compatible. The library is made with templates, with his cost model of floating-point operations, performing explicit loop unrolling and vectorization, these techniques help to the compiler to perform compile-time optimizations.

2.3.2 Arm Performace Libraries

Arm Performance Libraries(ArmPL) are vendor-specific math libraries for scientific computing and HPC workloads. This library is hardware-specific code for Arm64 architectures with generic and aware microarchitecture implementations.
The library implements BLAS and LAPACK interfaces for linear algebra and FFTW interface for FFT. Since these interfaces are De facto in scientific computing and HPC, we can use these implementations without change the code or with little effort.

2.3.3 Intel MKL-DNN

Intel MKL-DNN is an open-source library for accelerating deep learning frameworks on Deep Learning. The Intel-MKLDNN have a built-in Just-in-Time compiler to runtime optimizations in code for certain operations. This library is used in most the most popular Machine Learning frameworks like TensorFlow, PyTorch and Apache MXNet.

2.4 Profiling tools

The profiling tools help to the programmer and user to find most waste time routines in an execution.

[2] http://eigen.tuxfamily.org/index.php?title=Main_Page#Overview

2.4.1 Perf

Perf is a profiler tool for Linux. Perf uses software events that are predefined by the kernel or user-defined; also can get (Performance Monitoring Unit)PMU hardware events provided by the processor. For example, the number of involuntary context-switches is a software event, and the number of executed instruction is an example of a hardware event.

2.4.2 gprof

GNU profiler most know as gprof is a performance analysis tool that profiles the execution of an application. Gprof works instrumenting the code during the compilation phase. During the execution of the application, the application will collect data of the time spent in the routines and write a report file. After the execution, we can get the profile in differents view like annotating the code, a heatmap of the call-graph or flat profiling.

2.5 Machine Learning

Typically, Machine Learning methods are composed of two phases:

Training is the phase where a model "learns" from a reference dataset (e.g., a set of reference images picturing an object to be recognized). This phase is the most computationally intensive part of the Machine Learning techniques. Due to its heavy computational requirements, this phase is usually executed on powerful computational platforms such as the cluster of accelerators or supercomputers.

Inference is the phase where the model assigns a given input (e.g., an image) to a category with a certain accuracy. This part is less computationally intensive and can be deployed on edge devices such as cars, smartphones and cameras.

We can classify the machine learning in 4 different groups [6]:

Supervised learning are the type of algorithms that need input dataset provided with the correct answer for each example. Some examples are linear regression, decision trees and Gaussian naive Bayes.

Unsupervised learning are the types of algorithms does not need an input dataset with the correct answer. The algorithm is categorizing the examples in common. Some examples are k-Means, hidden Markov models and hierarchical clustering.

Reinforcement learning like in unsupervised learning, the algorithms do not need a correct answer, but when the answer is wrong, it gets notified. The algorithm explores multiple possibilities until it finds the correct answer. Some examples are Q-learning, DQN and SARSA.

Evolutionary learning like in reinforcement learning and unsupervised learning does need a dataset with the correct answer. This types of algorithms explore multiple possibilities and get and score for each path. After this exploration, it keeps going exploring more possibilities with the previous best scores models with some changes in the weights.

2.5.1 Machine Learning frameworks: TensorFlow

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[3]. It is employed as a benchmark of new ML architectures [8] and as ML engine when coupled with well-trained models [9].

[3] https://github.com/tensorflow/tensorflow/tree/master/tensorflow

We perform our study using TensorFlow for training a network with images, using the benchmark tool from the TensorFlow repository[4]. 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 [10]. 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 *images per second* (img/s) as the performance figure.

2.5.2 TensorFlow parallelism techniques

TensorFlow threadpool The TensorFlow threadpool is based on the Eigen threadpool. In this pool, we can add a task by a lambda function that will be executed by a thread. This threadpool has two main parameters to tunning, *intra-ops* and *inter-ops*. TensorFlow defines *intra-ops* as the number of threads used to executing ready tasks. TensorFlow defines *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. We can see visual representation the TensorFlow operation graph on figure 5. We can see in figure 6 a visual representation of the execution of TensorFlow operation graph of the figure 5 on TensorFlow threadpool with *intra-ops*=3 and *inter-ops*=2. An analogy for the function of the threadpool in TensorFlow is OpenMP.

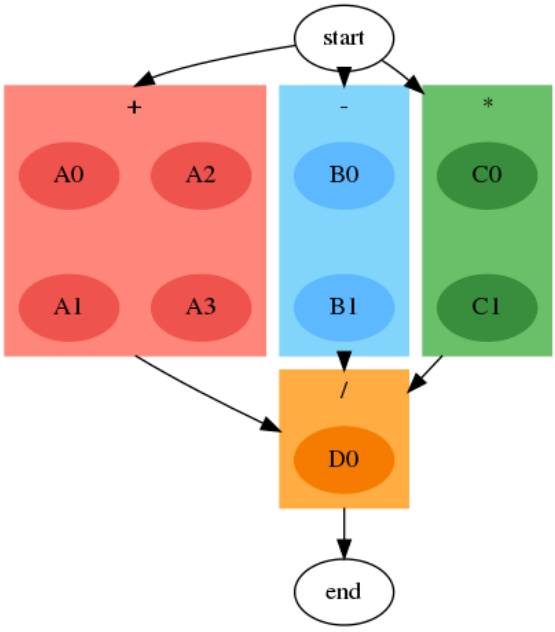


Figure 5: TensorFlow operation graph

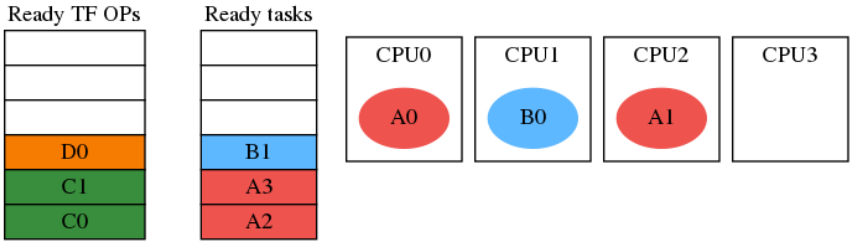


Figure 6: TensorFlow threadpool represenatation

[4] https://github.com/tensorflow/benchmarks

Horovod *Horovod* [11] is a distributed training framework for TensorFlow and other machine learning frameworks. We use Horovod for allocating work among processes. Horovod can use an MPI implementation in order to communicate process inside or in different nodes also can use Nvidia libraries for GPU-to-GPU direct communications. Horovod helps to scale-out machine learning models. An analogy for the function of the Horovod in TensorFlow is MPI.

2.6 HPC hardware 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 Marvells 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 [14].

3 Motivation

Until recently, machine learning problems were relatively small, and researchers could use a single powerful workstation to train their models. Nowadays, with the explosion of AI usage, the computational requirements of ML problems has grown. Researchers often create large and complex models [15] requiring gigabytes of a computer cluster to leverage this training phase. In this context, the use of cluster using CPU with large memory capacity, compared with a typical GPU accelerator that offers a limited amount of memory, is a must [16]. The complexity of those models does not fit in a single computer and needs to be distributed. With the appearance of larger computational platforms such as supercomputers, a benchmarking infrastructure is needed to provide a sense of their performance under Machine Learning frameworks.

3.1 Why we need benchmarking in ML workload?

Like in other areas of scientific computing, the benchmarking is a basic step in the design, deployment and operating of a computer or cluster. We can get different metrics to benchmark the system; generics like floating point operations(Flops) per second, elapsed time, Flops per Watt(Flops/Watt) and energy to solution; or domain/application specific like images per second(img/s) and steps per second(steps/s).

ML workloads start to take into account the importance of benchmarking of the platform. Nowadays, ML is becoming one of the most intensive computation tasks. We need to measure the performance of a machine in order to optimize the cost, energy, better CPU or accelerator design; to improve the use of our resources.

The ML community is discussing which is the best way and which best represents the performance of the platforms for ML workflows. But there are many challenges ahead in ML training benchmarks[5], first of all which metric to pick, throughput or time-to-train. On the one hand, we can increase the throughput of the benchmark and the elapsed time of the execution would increase, but with throughput metric, we can make smaller benchmarks. In the other hand, if we measure time-to-train, we need actual powerful machine or cluster to execute in a reasonable time and without using architecture simulators in order to design new computing elements.

3.1.1 Survey in state-of-the-Art ML benchmarks

The evaluation of platforms for the workflow of Machine Learning is a hot topic with numerous solutions proposed from vendors, framework developers and researchers. For this study, I considered a set of verified benchmarks/applications well known by the Machine Learning community:

- **MLPerf** [17]: Is becoming the reference benchmark in the Machine Learning world and it is highly supported by companies i.e., Google, Facebook, Arm and Intel.
- **TensorFlow Benchmarks**[6]: Are the built-in benchmarks that the developers of TensorFlow offer to their user to evaluate the performance of platforms, TensorFlow build and improvements in code.
- **HPE Deep Learning Benchmarking Suite(HPE DLBS)**[7]: Is the automated benchmarking tools developed by Hewlett Packard Enterprise (HPE) vendor.
- **CosmoFlow** [18]: This application aims to use deep learning techniques for cosmology simulations.
- **BERT** [19]: State-of-the-art machine learning for Natural Language Processing (NLP) tasks.

[5] https://researcher.watson.ibm.com/researcher/files/us-ealtman/MLPerf%20Design%20Choices.pdf
[6] https://github.com/tensorflow/benchmarks
[7] https://developer.hpe.com/platform/hpe-deep-learning-cookbook/home

Benchamrk/App	Image classification	Speech recognition	DPL	Physics Simulations
MLPerf	✓	✓	✓	×
TensorFlow Benchmarks	✓	×	×	×
HPE DLBS	✓	✓	×	×
CosmoFlow	×	×	×	✓
BERT	×	×	✓	×

Table 1: Machine Learning Benchamarks/Applications and his problems that solves

Table 1 can summarize the benchmarks and their different field of knowledge. Moreover, they are from many different providers like researchers, vendors, Machine Learning frameworks implementations. The literature shows more and more examples of mixed simulations that use Machine Learning techniques with state-of-the-arts classical simulation mathematical models e.g., molecular dynamics [20], computational fluid dynamics [21] or drug simulation [22]. The diversity of the proposes benchmark set attempts to cover this important matter involving transversal scientific fields.

3.2 Arm specific implementation

In recent years, the HPC and AI community have more interest in Arm architecture, but the X86 continues having the most of the market share in this segment. The TensorFlow framework for training phase is mainly designed to use efficiently on x86 CPUs and Nvidia GPUs. This fact makes the developers focus mainly on developing the applications and frameworks aware of x86 architecture.

On November 2018, Astra [14] an Arm based system enter to the Top500 list. The BSC is engaged through the Mont-Blanc project in enabling the Arm architecture into HPC [23]. Since I participate in the Mont-Blanc project and the clear irruption of Arm architecture in HPC, we decided to use Arm specific linear algebra libraries in order to improve the performance TensorFlow for Arm architectures. After the changes we will evaluate in the Mont-Blanc 3 prototype described in Section 2.6.

4 Project planning

4.1 Scope

This section will describe the scope of this project, including mandatory and optional objectives. Since the research path is defined but not yet implemented, a separate section will be dedicated to the risks and obstacles that can appear while implementing the project.

4.1.1 Objectives

In this project the researcher will need to:

- Search and study pros and cons of established benchmarks for Machine Learning
- Define a methodology in order to evaluate new platforms

4.1.2 Requirements

Platform evaluation requirements

- Select widely used benchmarks
- Inspect in deep all the selection to know his features
- Test different platforms with a selected features from the benchmarks
- Get the conclusion
- (Optional) Perform a specific platform optimization
- (Optional) Test again the affected platform/s
- (Optional) Get the conclusion of the vanilla vs optimized version

4.1.3 Risks

- Builds problems:
  Machine Learning frameworks have multiple dependencies with thousand of lines of code. Also they need to interact with the HPC software stack. Emerging technologies tools can be unmature and this can generate errors.
- Choose meaningful parameters:
  The parameters of the neural networks and the topology of the cluster can be combinatory problems. Their selection can waste a lot of time as well as the test of new parameters and different software stack configurations.

4.2 Work organization methodology

For the nature of this project the work organization methodology will be the iterative and incremental build model [24]. In this methodology the project is developed in a iterative fashion. This iterations are based on create or improved a feature in the project. Every time you finish an iteration the project will remain in a stable or coherent state depends of the task.

To verify every software feature or improvement comparison with well know and verified third-party implementations. Every step forward must pass the verifications performed in the past iterations in order to make sure of multiple code dependencies.

4.3 Description of tasks

This section describes the tasks needed to achieve the goals of the project.

4.3.1 Descrition at phase level

This project is parted into phases. Every phase is a bunch of tasks very related between them. Phases are weakly related between, but at the final, they have the same objective.

- **Project management (GEP) phase**
  This phase of the project will be a description of the technical, the timings and the budget/resources to develop it.
- **Platform evaluation (PE) phase**
  This phase will evaluate different platforms to Machine Learning workloads and get conclusions that will be useful for de Machine Learning community.

4.4 Descrition at task level

Project management (GEP) phase

- **[GEP-1] Meeting GEP introduction**
  In this meeting, the researcher will meet with the directors in order to present the GEP course. In this meeting will be a discussion about the scope of the project. The directors will help to organize and address a regular meeting to follow the work.
- **[GEP-2] Context and scope of the project**
  In this task, the research will need to prepare a description of the context and scope of the project
- **[GEP-3] Time planning**
  In this task, the researcher will divide the project into tasks and organize the time of the project.
- **[GEP-4] Budget and sustainability**
  In this task, the researcher will describe the budget and the sustainability of the project.
- **[GEP-5] Final document of project management**
  In this task, the researcher will address all the feedback from his GEP's tutor and prepare an associated document.
- **[GEP-6] Meeting GEP conclusions**
  In this meeting, the researcher, directors and teacher of the FIB(Ponent) will meet in order to summarize and get conclusions oof the GEP phase.

Platform evaluation (PE) phase

- **[PE-1] First contact meeting**
  In this meeting, the researcher will meet with his directors in order to discuss the platform evaluation topic. In this meeting, the directors will decide with platforms that are available to the researcher will be able to use in his evaluation. Also, the directors will be a debate about the technical methodology.
- **[PE-2] Meeting with Machine Learning framework real users**
  In this meeting, the researcher meets with first level AI/ML researchers to discuss to perform real workflows. Also, the experts will advise in the choice of execution parameters.
- **[PE-3] Select the platforms to evaluate**
  In this task, the researcher will choose with the advice of the directors and the experts in the field to choose the platforms where the evaluation will perform.
- **[PE-4] Select the benchmarks to use in the evaluation**
  In this task, the researcher will choose with the advice of the directors and the experts in the field to choose the benchmarks that will be used to perform the evaluation.

- **[PE-5] Build the frameworks to the selected platforms**
  Compile the dependencies and the framework it-self using the most aggressive optimization flags in each platform. Also, the researcher must check for each dependency and framework that the binary passes all the unit test.

- **[PE-6] Write the scripts to run the benchmarks**
  Write bash scripts that will be able to send jobs to the clusters, execute the benchmarks with the desired parameters and parse/classify the results.

- **[PE-7] Execute the benchmarks and address the errors**
  In this task, the researcher will ensure that the cluster jobs are running without errors and with the desired behaviour. If there are any error or unwanted behaviour, the researcher will address those errors.

- **[PE-8] Conclusions of the benchmarks vanilla code**
  The researcher will think in the conclusions of the executions and the pros and cons of the selected benchmarks and platforms. The researcher also will describe with the previous task the portability of the benchmarks solution used previously.

- **[PE-9] Meeting platform evaluation conclusions**
  In this meeting, the researcher, the directors and the FIB teacher(Ponent) will meet in order to discuss the conclusions.

- **[PE-10] Write documentation**
  The researcher must document all the process of the platform evaluation phase. From why he choose the platform and benchmarks, technical methodology and conclusions.

- **[PE-11] Weekly meeting with directors**
  In this weekly meeting, the directors will follow the progress of the project in the platform evaluation field. They also will discuss and advise on the decision taken by the researcher.

4.4.1 Summary

ID	Name	Time(h)	Dependencies
GEP	Project management (GEP) phase	57	-
GEP-1	Meeting GEP introduction	1	-
GEP-2	Context and scope of the project	20	GEP-1
GEP-3	Time planning	15	GEP-1
GEP-4	Budget and sustainability	15	GEP-1
GEP-5	Final document of project management	5	GEP-[2,3,4]
GEP-6	Meeting GEP conclusions	1	GEP-5
ID	Name	Time(h)	Dependencies
PE	Platform evaluation (PE) phase	163	-
PE-1	Meeting first contact	2	-
PE-2	Meeting with Machine Learning framework real users	2	PE-1
PE-3	Select the platforms to evaluate	20	PE-2
PE-4	Select the benchmarks to use in the evaluation	35	PE-2
PE-5	Build the frameworks to the selected platforms	40	PE-[3,4]
PE-6	Write the scritps to run the benchamrks	6	PE-[3,4]
PE-7	Execute the benchmarks and address the errors	20	PE-[5,6]
PE-8	Conclusions of the benchmaks vanilla code	20	PE-7
PE-9	Meeting platform evaluation conclusions	2	PE-8
PE-10	Write documentation	40	-
PE-11	Weekly meeting with directors	8	-
ID	Name	Time(h)	Dependencies
Total	Project	220	-

Table 2: Summary of the tasks

4.4.2 Resources

Human resources During the GEP phase will be involved in the project, the principal researcher, the directors and the FIB teacher(Ponent).
During the platform evaluation phase will be involved in the project, the principal researcher, Machine Learning experts to be consulted, a DevOps team, cluster user support team and reproducibility expert.
During the performance, in the analysis phase will be involved in the project, the leading researcher, Machine Learning experts to be consulted and a testing team.

Hardware resources

- Laptop Used in all project
- Clusters: Used in platform evaluation phase
  - MareNostrum 4
  - CTE-POWER
  - Dibona

Software resources Used in during the project.

- Git
- LaTex
- PlantUML

- AWK
- LibreOffice

4.5 Tasks timing planification

This section will show the task in a timeline in the form of a Gantt diagram.

4.5.1 Gantt diagramm

You can see in the Figure 7 a Gantt diagram with the task described above. This Gantt diagram have visually separate in colors and with a horitzontal lines the 3 principal phases of the project. Every rectangle is a task and the arrows are de dependencies with other tasks. Based on the description of the Table 2. The phase platform evaluation and the performance analysis phase can be performed at the same time. Some of the platform evaluation task given its nature can be very asynchronous e.g., submit the jobs and wait for the results. But for simplicity and organization, the tasks for those phases will be executed in a sequential fashion.

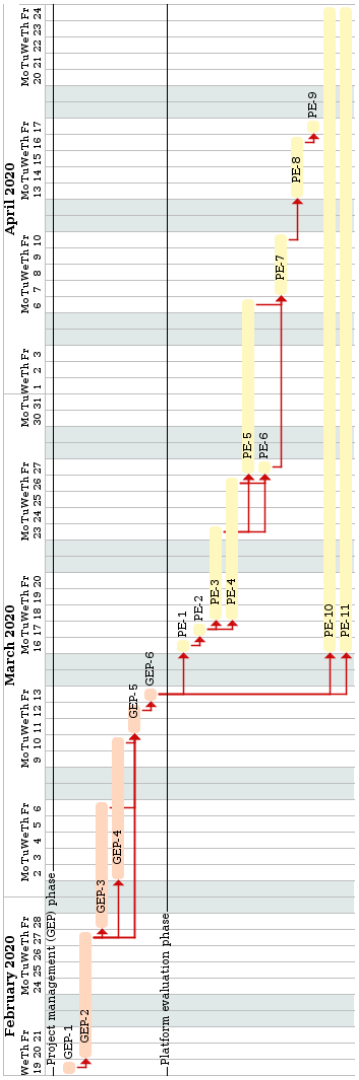


Figure 7: Gantt diagram

4.6 Risks management

This section will describe the most probable risks and whom to manage it.

4.6.1 Most risky task and alternative plan

PE-5 task can be risky on the cluster with emerging technology. The build tools in this type of machine can be unmatured. In order to manage the build problems, we can decide to low the optimizations flags and remove some dependencies.
PE-7 task can extremely risk there are many runtime problems when using state-of-the-art machines and software. If one combination of machine/framework has much error in runtime is possible to discard from the evaluation until a new update.

4.7 Budget

This section will describe the economic aspect of this project. First of all, the section will identify the costs to consider in the budget of this project. On the other hand, it will be a quantitative description by category and an overall summary. Finally, it will describe the mechanisms to control high possible scenarios with budget deviations.

4.7.1 Identification of costs

This subsection will describe the cost and map with tasks descripted in the last section. For simplicity, this budget report will consider three different categories, human resources costs, HW/SW costs and indirect costs.

Human resources cost This section will describe the human resources used and its cost. The project will be developed mainly by one employer. This role will be assigned to the junior research engineer. Also, there will be other roles in order to take decisions and managing resources. These roles will be assigned to the project manager and seniors researchers. Finally, in some parts, the project will need a specific knowledge field researchers. This role will be assigned to seniors research engineer. The taxes are estimated as 30% of the net salary.

The Table 3 summary the price per hour and a summary of the total cost. The Table 4 associate the roles with tasks.

Role	Price(€/hour) before tax	Price(€/hour) after tax	N. hours	Total cost(€)
Junior research engineer	15.00	19.50	509	9925.00
Senior research engineer	25.00	33.75	40	1350.00
Project manager	29.00	39.15	11	430.65
Senior reseacher	32.00	44.80	114	5107.20

Table 3: Human resources costs

Role	Related tasks
Junior research engineer	all
Senior research engineer	PE-2
Project manager	GEP-1, PE-1
Senior reseacher	GEP-[1,6], PE-[1,3,4,8-11]

Table 4: Human resources associated with tasks

HW/SW costs This section will describe the costs of the software and hardware needed to develop the project. The hardware cost will be the personal supplies for the main developer. Besides, the computation power needed will be charge in hours of CPU used. Finally, the software will be as much as possible Free Software[8] and also in this case free of cost. The amortization is based in 6 moth usage of the material. In these case the price is divided by 8 because all the hardware have 4 years of useful life.
The Table 5 describe the costs of the hardware and software of this project.

Product	Price(€) after tax	Units	Usefull life	Amortization	Related tasks
Laptop	1700.00	1	4	212.50	all
Monitor	200.00	1	4	25.00	all
Keyboard	20.00	1	4	2.50	all
Mouse	10.00	1	4	1.25	all
MN4 CPU hour(Non-official)[9]	∼0.01	1000000	N/A	9100.00	PE-7
Xubuntu 18.04	0	1	N/A	N/A	all
Vim	0	1	N/A	N/A	all
GCC	0	1	N/A	N/A	all
Git	0	1	N/A	N/A	all
SGoQ	0	1	N/A	N/A	all

Table 5: Hardware and Software costs

Indirect costs This section will describe the cost of the physical place where the project will develop. The main developer will need a space where work in which work in the project and supplies. Also the where need to rent a place to meet with the other roles. The transport is other of the points to consider. The project has a daily worker that can get a monthly public transportation pass. For the other members, the cost will be 1.135€per travell. The Table 6 describe the indirect costs of the project in order to well development.

Product	Price(€) after tax	Units	Total cost(€)
Office rent(monthly)	2500	6	15000
Office furniture	2000	1	2000
Office supplies(monthly)	250	6	1500
Transportation monthly pass(T-Usual)	40	6	240
Transportation daily pass(T-Casual)	11.35	7	79.45

Table 6: Indirect costs

Summary costs The Table 7 summarizes the cost of the project.

Origin of cost	Total(€)
Human resources cost	16812.85
HW/SW cost	9341.25
Indirect cost	18819.45
Contingency	6746.03
Project cost	51719.58

Table 7: Summary costs

[8]https://www.fsf.org/about/what-is-free-software
[9]Get from https://www.bsc.es/news/bsc-news/marenostrum-4-has-provided-85-million-euros-computing-hours-researchers-outside-the-bsc-during-2019

4.7.2 Management control

This section will describe the primary possible budget deviations. The three more possible deviations will be:

- Human resource deviation
  It is plausible to need more hours from human resources. Because of bad decisions, bugs in the code and non-coherent results. It is possible at least 30% more hours from the junior research engineer, 15% more hours from the seniors' researchers and 5% more hours of senior research engineer.
- Computation time deviation
  Is very common to get a runtime error and execute again the simulation changing the parameters depending on the other platforms. The deviation will be around 30% more hours.
- Amortization deviation
  If the main developer needs more hours will be less amortization of the equipment. This deviation will be proportional to the extra hours taken by the junior research engineer. 30%

Origin of cost	Total(€)
Human resources cost	16812.85
HW/SW cost	9341.25
Indirect cost	18819.45
Contingency	6746.03
Human resources deviantion	3811.08
Computation time deviation	2730.00
Amotization deviation	72.37
Project cost + deviations	58333.03

Table 8: Summary costs + deviations

4.8 Sustainability report

This report will help me to make questions about the sustainability of an IT project. Like how this project will impact in the society. Sustainability is a complex topic, and I also touch the surface of every possible dimension.
In the environmental dimension, I have a low perspective on the possible impact of this project. In one hand, I conscious of wasted thousands or millions of CPU hours computing non-direct knowledge. However, on the other hand, it is possible to this possible new knowledge make to other researchers waste less energy in his experiments and better use of the machines. This can help in less raw computation power for the same amount of project and a directly propositional of computer wasted.
In the economic dimension, a moderate I know how to analyse and quantify the resources and computation power this project will need. Also, this project, I am not able to calculate the economic viability in a business field. But will could create wealth in the Machine Learning research world.
In a social dimension, I have a moderate level of scope. Personally, this project will help me to improve my formal writing and help to meet with different people. This project can help in, and indirect way improved the quality of life of the people.

4.8.1 Envionmental dimension

- **Regarding PPP: Have you estimated the environmental impact of undertaking the project?**
  The main environmental negative impact it will be waste of electric energy due to platform evaluation phase. The laptop, monitor, keyboard and mouse used to this project will brand new.

These materials will have a high environmental impact in creation and use during the time of life.

- **Regarding PPP: Have you considered I how to minimize the impact, for example by reusing resources?**
  The project will use large clusters that are already deployed. This hardware will be a reused by many researchers, and his use is over 80% utilization.

- **Regarding Useful Life: How is the problem that you wish to address resolved currently (State-of-the-Art)?In what ways will your solution environmentally improve existing solutions?**
  This project, on the one hand, will help the researcher to choose the right cluster to run his Machine Learning executions. This will make that the executions will take less time. Normally less time imply less energy and more usage of the machine for other researchers. On the other hand, this project will develop tools that get an insight into machine utilization to the Machine Learning frameworks developers. This help better utilization of the resources.

4.8.2 Economics dimension

- **Regarding PPP: Have you estimated the cost of undertaking the project (human and material resources)?**
  This project has a detail estimated cost description in Section 4.7.

- **Regarding Useful Life: How is the problem that you wish to address resolved currently (State-of-the-Art)? In what ways will your solution economically improve existing solutions?**
  This solution will help to reduce the hours taken to run a training phase in a Machine Learning problem. This helps to reduce the bills to pay every month to the researchers or get more room to get new research.

4.8.3 Social dimension

- **Regarding PPP: What do you think undertaking the project has contributed to you personally?**
  This project helps me to structure my ideas in a formal text and get in contact with other researchers in other areas of interest.

- **Regarding Useful Life: How is the problem that you wish to address resolved currently (State-of-the-Art)?In what ways will your solution socially improve (quality of life) existing?**
  Directly will be not improved in quality of life of anybody.

- **Regarding Useful Life: Is there a real need for the project?**
  Yes, there are a group of Machine Learning researcher who needs and speed-up to his Machine Learning training phase in a CPU. Also, there is a need to know how Machine Learning frameworks behave in big clusters to best resources utilization.

5 Implementation

5.1 Deployment on different cluster

In this section we describe how to deploy the TensorFlow binary in the different cluster.

5.1.1 Vanilla

In this section, we describe how to deploy a binary without a specific library for the architecture. In this vanilla version, we use the Eigen library as the main library for lineal algebra back-end. Fist of all, we will create a virtual-environment to maintain our environment with a controlled version of python packages. After we create our virtual-environment in each cluster will download Bazel in order to build TensorFlow source code. All the clusters used in this evaluation does not have internet access. We download the source code from the TensorFlow official repository. To build the TensorFlow source code, we need to have access to the internet to download third-party dependencies because the files describing the build process only have the possibility to get the source from the internet. To solve this problem, we write a code in python that download the dependencies in a machine that have internet and change the Bazel files to get the sources from local files. We can see part of the code in the Appendix A.1. After the last step, have the dependencies in our laptop we copy the dependencies to our cluster, to compile we load the modules described in Table 10. We compile the TensorFlow using next command `-march=native -mtune=native -O3` where native will be.

5.1.2 Using Intel MKL-DNN library

In order to use Intel MKL-DNN, we need to add the –config=mkl-dnn to Bazel in order to activate Intel MKL-DNN in our x86 CPU.

5.2 Find critical kernels

In this section, we present the profiling of a TensorFlow training session. Our base code for TensorFlow is the version 1.10.0[10] and we use the benchmark tool from the TensorFlow repository[11]. The model is `AlexNet` and the input set is based on synthetic data, so we do not read real datasets (e.g., ImageNet) to avoid intensive IO operations. The batch-size of our tests includes 1024 images. For profiling a simple case, we configure the number of inter-ops to 1, so we process one operation of the graph at a time, and number of intra-ops to 64, so to take advantage of all the computational resources within a Dibona node (64 threads).

To profile a training step with `AlexNet` we use a TensorFlow built-in trace generator. Traces generated by this tool can be visualized with Chromium or Chrome. In Figure 8, we show one trace of the Vanilla version of TensorFlow. The trace visualization tool gives us three main pieces of information: the type of operation executed (red box), tensor lifespan (blue), and memory usage (green).

The trace visualizer can also provide the aggregated execution time of every operation. With this information, we are able to understand which are the most time consuming functions. In Figure 9, 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. In table 9, we show that most of the execution time is spent in routines from a library called `Eigen`.

[10] https://github.com/tensorflow/tensorflow/releases
[11] https://github.com/tensorflow/benchmarks

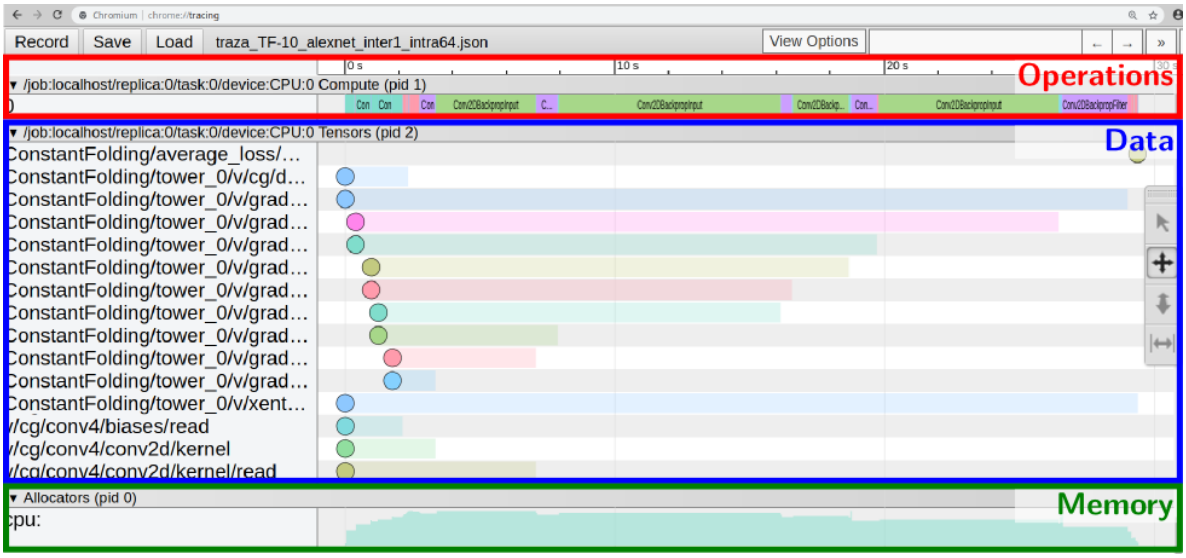


Figure 8: TensorFlow visualizer of an execution using inter-ops=1 and intra-ops=64

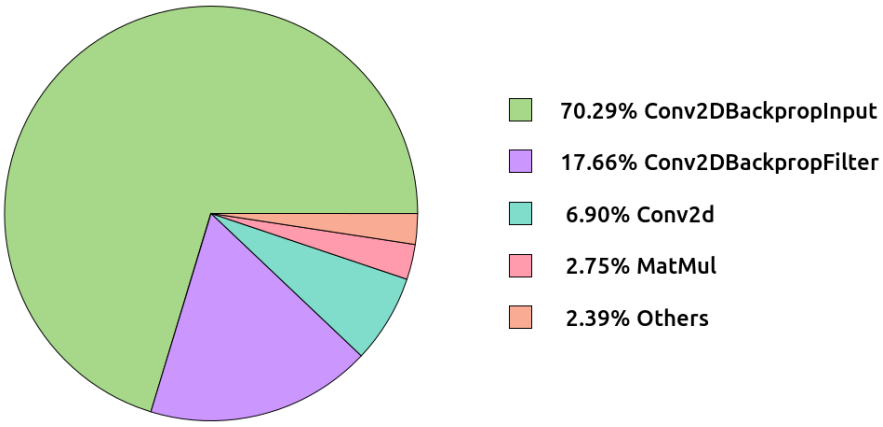


Figure 9: Time distribution among operations for the `AlexNet` model

Overhead	Shared object	Symbol
76.58%	_pywrap_tensorflow_internal.so	Eigen::internal::gebp_kernel<float, float, long,...
4.44%	_pywrap_tensorflow_internal.so	Eigen::internal::gemm_pack_rhs<float, long,...
4.06%	_pywrap_tensorflow_internal.so	Eigen::internal::gemm_pack_rhs<float, long,...
2.02%	_pywrap_tensorflow_internal.so	std::_Function_handler<void (long, long),...
1.50%	_pywrap_tensorflow_internal.so	Eigen::internal::gemm_pack_lhs<float, long,...
1.42%	_pywrap_tensorflow_internal.so	(anonymous namespace)::Col2im<float>
1.32%	libc-2.17.so	memcpy

Table 9: Perf report output of TensorFlow/1.10, only selecting functions with overhead >1%

Eigen[12] 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. The `Eigen::internal::gebp_kernel` function spends ∼76% of the time in the subroutine of Eigen *general block-times-panel multiply* for matrix multiplication. Once the significance of Eigen calls to the total execution time was identified, we analyzed `Conv2DBackpropInput` and `Conv2DBackpropFilter` for isolating Eigen calls. We performed a `perf record` saving the call-graph but the Eigen calls wasn't executing inside the of the scope of the `Conv2DBackpropInput` and `Conv2DBackpropFilter` operations. The calls were executed inside the scope of a thread executed manage by the threadpool. With this scenario we need to perform a static analysis of the operations in order to isolate the Eigen calls. We went to the source code of TensorFlow and find the routines `Conv2DBackpropInput` and `Conv2DBackpropFilter`, in this routines we saw that Eigen calls. They were inside a for loop that partitions the matrices with a cpp lambda function that captures the portions of the matrices and delegates the execution of the partial matrix multiply to the threadpool.

5.3 Using the ArmPL

We identified the Eigen calls `Eigen::internal::gebp_kernel` located within `Conv2DBackpropFilter` and `Conv2DBackpropFilter`. The `Eigen::internal::gebp_kernel` is a function to make a matrix matrix product "by parts". This technique is used by TensorFlow to generate different task that are added to a threadpool to create parallelism. Our goal is to replace this call by the optimized version for Arm uarchitecture provided by the Arm Performance Library.

The implementation consists of 4 steps:

1. Set up the environment
2. Prepare auxiliary code for the debug mode
3. Write routines with ArmPL
4. Test the new implementation

5.3.1 Set up the environment

The first step is to prepare the TensorFlow environment to perform our changes. TensorFlow use Bazel tool in order to build the code. We create a new folder named ArmPL in the folder third_party at TensorFlow source root. This third_party folder contains the code and instructions to build and link external libraries. Our new folder ArmPL contains three new files where is described how to find and link the Arm Performance Libraries by Bazel build system. The files contents are describe in appendix A.2. Also we prepare the build for three possible scenarios the vanilla version, with the Arm Performance Libraries and debug mode. The debug mode is used for verification and is explained in detail in the following section.

[12] http://eigen.tuxfamily.org/

```
bazel build  ... #Build vanilla version
bazel build  --config=use_armpl ... #Build ArmPL version
bazel build  --config=use_armpl --config=use_armpl_debug ... #Build ArmPL with debug mode
```

Listing 1: Build user interface

5.3.2 Prepare auxiliary code for the debug mode

Due the complexity of the code and the operations is difficult to verify if the changes made are computing the same operation. This second second step prepares the tools to the proper verification between the original code and optimized version. In order to verify the correctness of our implementation the debug runs several steps. The different steps can be seen in the diagram in Figure 10. As we can see in the debug mode the input is computed trough the original Eigen function and the new implemented ArmPL function. We implemented a new routine to compare the resulting matrix from both functions with a certain tolerance (the code of the diff routine can be found in Appendix A.3.

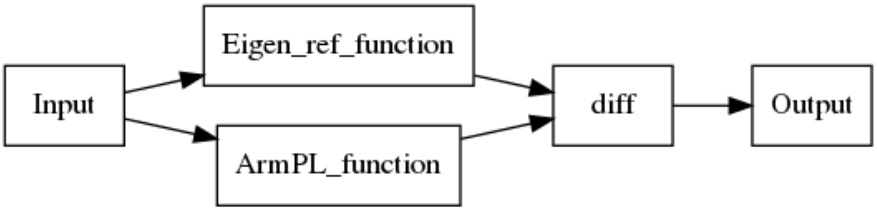


Figure 10: Debug mode

```
/*...*/
#ifdef ARMPL
        //Add alternative code with ArmPL
    #ifdef ARMPL_DEBUG
        //Execute original code and compare results
        C_debug.device(context->eigen_cpu_device()) = A.contract(B, contract_dims);
        bool areEqual = areEqualMatrix<T>(out_debug, col_buffer_data,
                                          output_image_size*filter_total_size);
        if(areEqual) std::cout << "Conv2DBackpropInput DEBUG: " << "SUCCESS"<< std::endl;
        else         std::cout << "Conv2DBackpropInput DEBUG: " << "FAIL"   << std::endl;
    #endif //ARMPL_DEBUG
#else
        //Original code
        C.device(context->eigen_cpu_device()) = A.contract(B, contract_dims);
#endif //ARMPL
/*...*/
```

Listing 2: Preparing Conv2DBackpropInput Operation to plug new code

5.3.3 Write routines with ArmPL

The code implemented to plug the ArmPL libraries in TensorFlow is shown in Listing 4. The auxiliar code necessary to translate data types from TensorFlow to cblas is shown in Listing 3. In the third step we replace the Eigen implementation with the Arm Performance Library gemm routine. The code implemented in the operations TensorFlow code is orientated to the Eigen interface. To use the generic BLAS interface we had created a interface that is able to change from Eigen interface to the classical BLAS gemm interface. The function `contract_to_cblas` is the responsible to prepare and execute the gemm operation within the TensorFlow context. TensorFlow use different data types

in their operations, e.g., double, half, complex, but ArmPl and other BLAS implementations only implements double and single precision. For this reason in the code of `contract_to_cblas` Listing in 3, we specialize the float and double to execute with the Arm Performance Library and the other types with the Eigen implementation.

```
#define USE_EIGEN_TENSOR
#define EIGEN_USE_THREADS

#include "tensorflow/core/framework/op_kernel.h"
#include "third_party/eigen3/unsupported/Eigen/CXX11/Tensor"

#include "armpl.h"
#include <iostream>

template<typename T>
void contract_to_cblas(tensorflow::OpKernelContext* context, bool TransA, bool TransB,
                       const T* a, const T* b, T* c, int m, int n, int k, bool accumulate){
    typedef Eigen::TensorMap<Eigen::Tensor<      T, 2, Eigen::RowMajor>, Eigen::Unaligned>
            TensorMap;
    typedef Eigen::TensorMap<Eigen::Tensor<const T, 2, Eigen::RowMajor>, Eigen::Unaligned>
            ConstTensorMap;

    ConstTensorMap A( a, k, m);
    ConstTensorMap B( b, k, n);
    TensorMap      C( c, m, n);

    Eigen::array<Eigen::IndexPair<Eigen::DenseIndex>, 1> contract_dims;
    contract_dims[0].first = 1;
    contract_dims[0].second = 1;
    if(accumulate)C.device(context->eigen_cpu_device()) += A.contract(B, contract_dims);
    else C.device(context->eigen_cpu_device()) = A.contract(B, contract_dims);
}

template<>
void contract_to_cblas<float>(tensorflow::OpKernelContext* context, bool TransA, bool TransB,
                              const float* a, const float* b, float* c, int m, int n, int k,
                              bool accumulate){
        cblas_sgemm( CblasRowMajor, TransA?CblasTrans:CblasNoTrans, TransB?CblasTrans:CblasNoTrans,
                     m, n, k, 1.0f, a, TransA?m:k, b, TransB?k:n, accumulate?1.f:0.f, c, n);
}

template<>
void contract_to_cblas<double>(tensorflow::OpKernelContext* context, bool TransA, bool TransB,
                               const double* a, const double* b, double* c, int m, int n, int k,
                               bool accumulate){
        cblas_dgemm( CblasRowMajor, TransA?CblasTrans:CblasNoTrans, TransB?CblasTrans:CblasNoTrans,
                     m, n, k, 1.0f, a, TransA?m:k, b, TransB?k:n, accumulate?1.f:0.f, c, n);
}
```

Listing 3: Code responsable of convert from Eigen contract to C BLAS interface

```
/*...*/
#ifdef ARMPL
        auto out_data = out_backprop_data + output_offset * image_id;
        contract_to_cblas<T>( context, false, true, out_data, filter_data, col_buffer_data,
                              output_image_size, filter_total_size, dims.out_depth, false);
    #ifdef ARMPL_DEBUG
        C_debug.device(context->eigen_cpu_device()) = A.contract(B, contract_dims);
        bool areEqual = areEqualMatrix<T>(out_debug, col_buffer_data,
                                          output_image_size*filter_total_size);
        if(areEqual) std::cout << "Conv2DBackpropInput DEBUG: " << "SUCCES"<< std::endl;
        else         std::cout << "Conv2DBackpropInput DEBUG: " << "FAIL"<< std::endl;
    #endif //ARMPL_DEBUG
#else
        C.device(context->eigen_cpu_device()) = A.contract(B, contract_dims);
#endif //ARMPL
/*...*/
```

Listing 4: Conv2DBackpropInput Operation with ArmPL version implemented

5.3.4 Test the new implementation

Finally, we run the AlexNet and ResNet-50 model to verify the changes. We run some of the experiments described in the Section 6. To perform the verification, we run with the debug mode described in the previous steps. This execution perform three epochs the ResNet-50 and AlexNet models with a batch size of 2, 4 and 8 images. The OpenMP was configured to run with 8, 16, 32 and 64 threads for each model and each batch size.
During the execution with the debug mode first was execute the call to the ArmPL and after the reference calls from the Eigen library. After the execution of the reference code and the optimized architecture call to lineal algebra library, we compare the results to verify if the two routines are performing the same mathematical operation to the matrix. After executing the tests described above without differences in the two output matrix, we assume that the new code is verified.

6 Evaluation on image recognition ML models

In this section we see the evaluation of two State-of-the-Art machine learning models in the area of the image recognition. This evaluation is perform in modern HPC machines with vendor lineal algebra optimized libraries.

6.1 Methodology

In this evaluation, we use the benchmark tool that is available in the TensorFlow repository[13]. This tool helps us to select a predefined and well know ML model, parallelization configurations, batch sizes and other runtime parameters. The benchmark tool computes every N batches, specified by the user, the number of images processed per second; also be reported at the end of the execution.
We are aware that varying the batch size affects the convergence of the training process. However, we left the study of this correlation out of our analysis, focusing only on the throughput.
In all the tests a synthetic dataset is used provided by the benchmark tool to avoid the overhead produced by the parallel filesystem. In the test performed for multinode evaluation we use ImageNet [25] dataset be explicitly said to be used.
The images per second(img/s) are our Figure of Merit. The img/s fluctuate during the time, in the firsts measurements this fluctuation trends to be bigger due cold caches. For this reason we run each experiment at least during ten minutes and minimum of three batches to reach a steady regime of img/s. To provide reasonable statistical accuracy, we average five executions for each test. Since we notice variability below 10% in all our measurement campaign, we neglect error bars.

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

Table 10: Environment for each cluster

In Table 10 we show the software environment used in each cluster and for each version. We tried to keep the environment across clusters as homogeneous as possible based on available software. We used the TensorFlow code version 1.11, except in Dibona where we evaluated the modified version leveraging the ArmPL. This modified version is reported in Table10 as 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.

6.2 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. In both tests, we map threads on physical cores contiguously. 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 performance improvement achieved achieved when using optimized linear algebra libraries as back-end. This lineal algebra libraries can be ML specific like Intel MKL-DNN or generic like we saw in Section 5. Moreover, we present performance figures for a wide range of batch sizes, making the study informative for both researchers: those tied to a fixed batch size can foresee the performance of their models, and those with flexibility in the batch size choice can select the most convenient configuration.

For this evaluation, we use the three HPC clusters described in Subsection 2.6: MareNostrum4, Dibona, and Power9.

[13]https://github.com/tensorflow/tensorflow/tree/master/tensorflow

6.2.1 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.

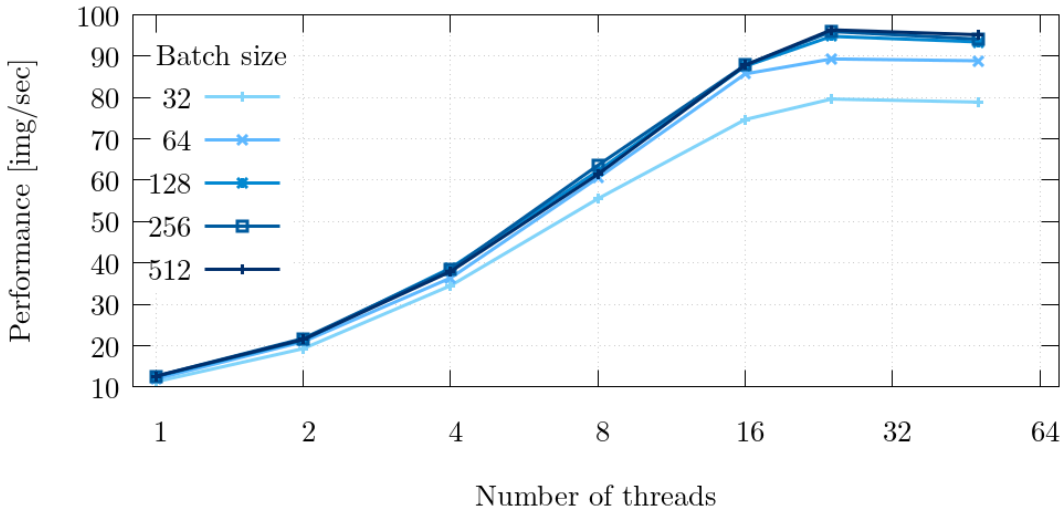


Figure 11: Thread Scaling of AlexNet with Vanilla version on one node of MareNostrum4

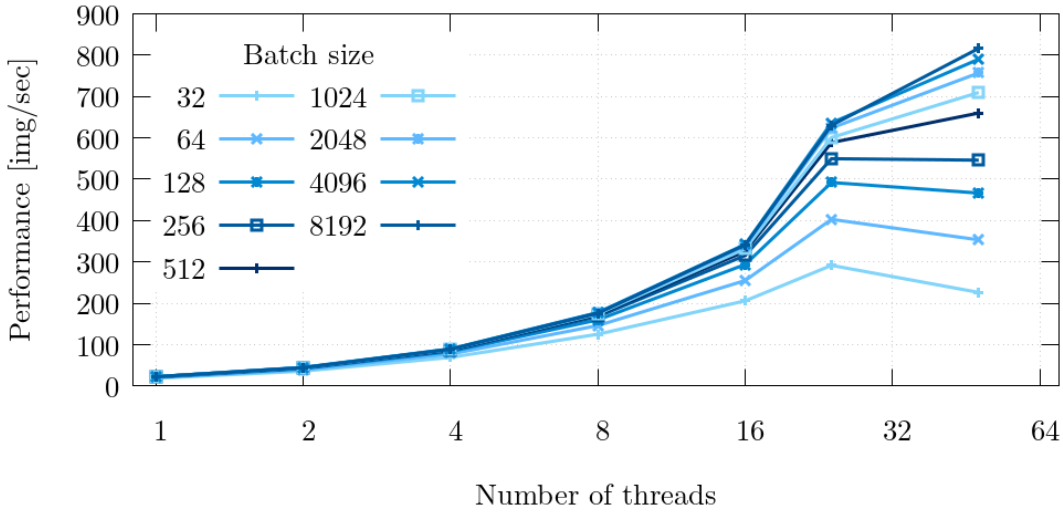


Figure 12: Thread Scaling of AlexNet with MKL version on one node of MareNostrum4

In Figure 11 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 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.

In Figure 12 is shown the performance of AlexNet in MareNostrum4 when running the TensorFlow version using MKL libraries. Comparing Figure 11 and 12 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.

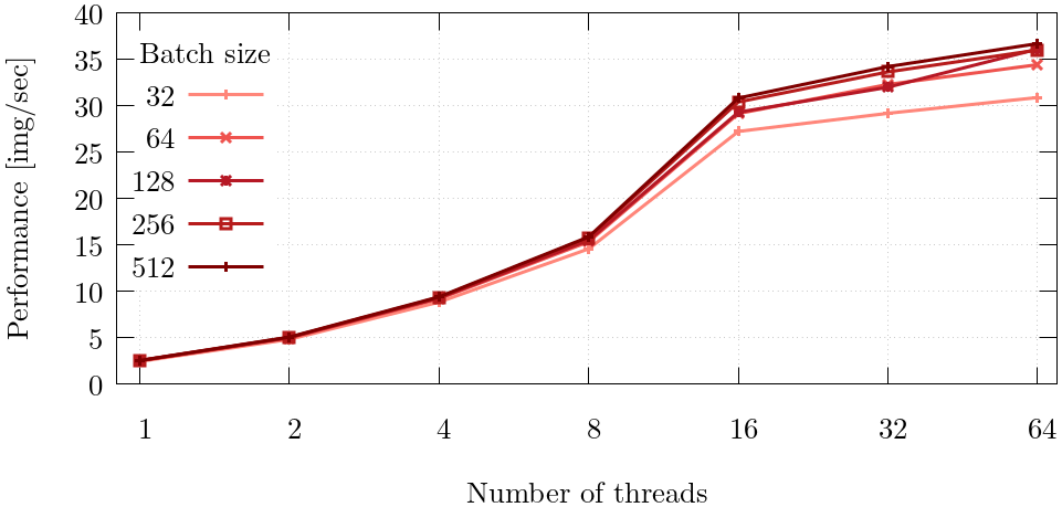


Figure 13: Thread Scaling of AlexNet with Vanilla version on one node of Dibona

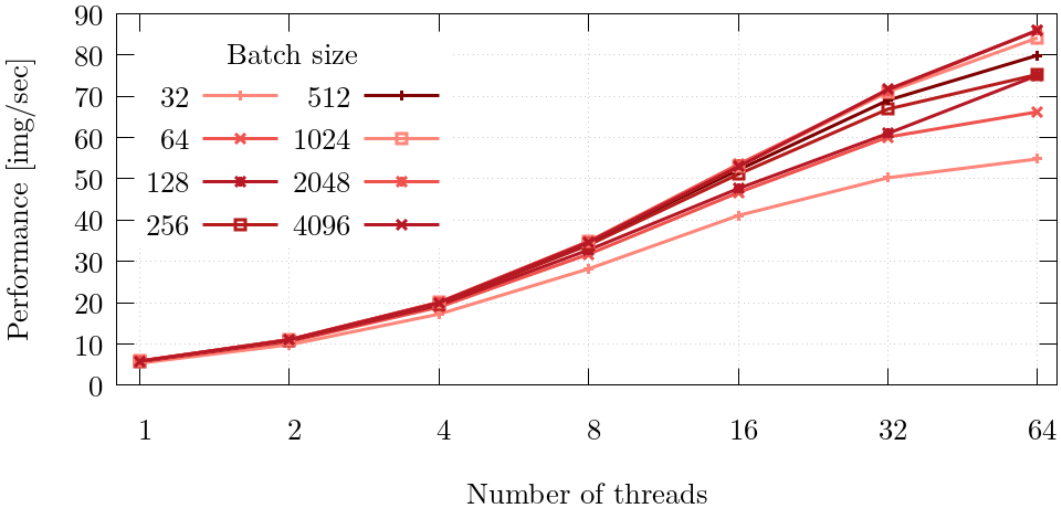


Figure 14: Thread Scaling of AlexNet with ArmPL version on one node of Dibona

In Figure 13 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.

Figure 14 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.

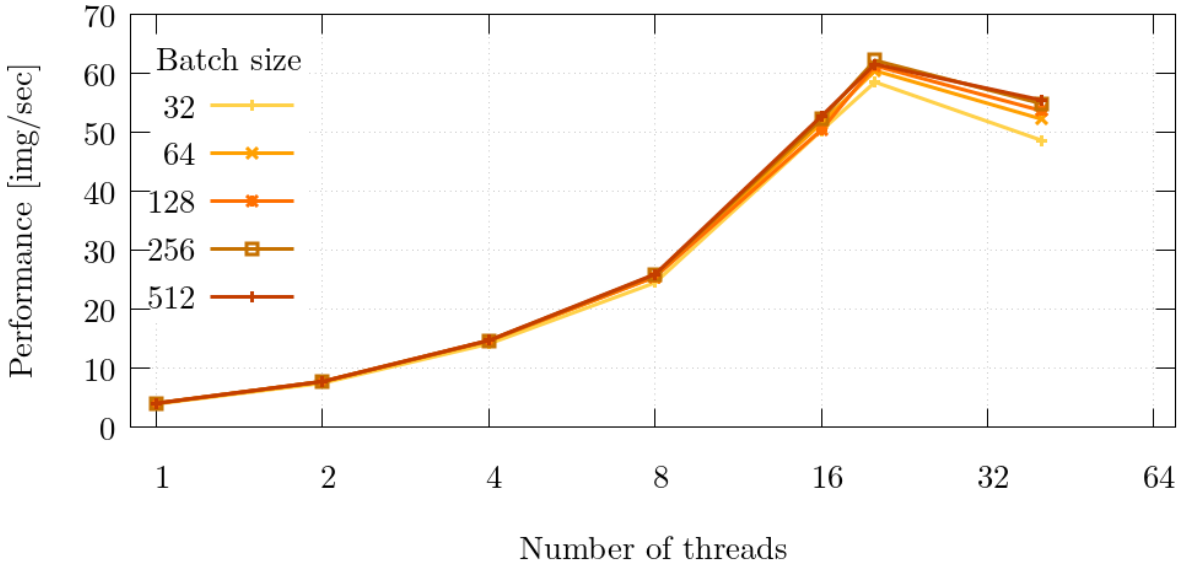


Figure 15: Thread Scaling of AlexNet with Vanilla version on one node of Power9

In Figure 15 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, 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.

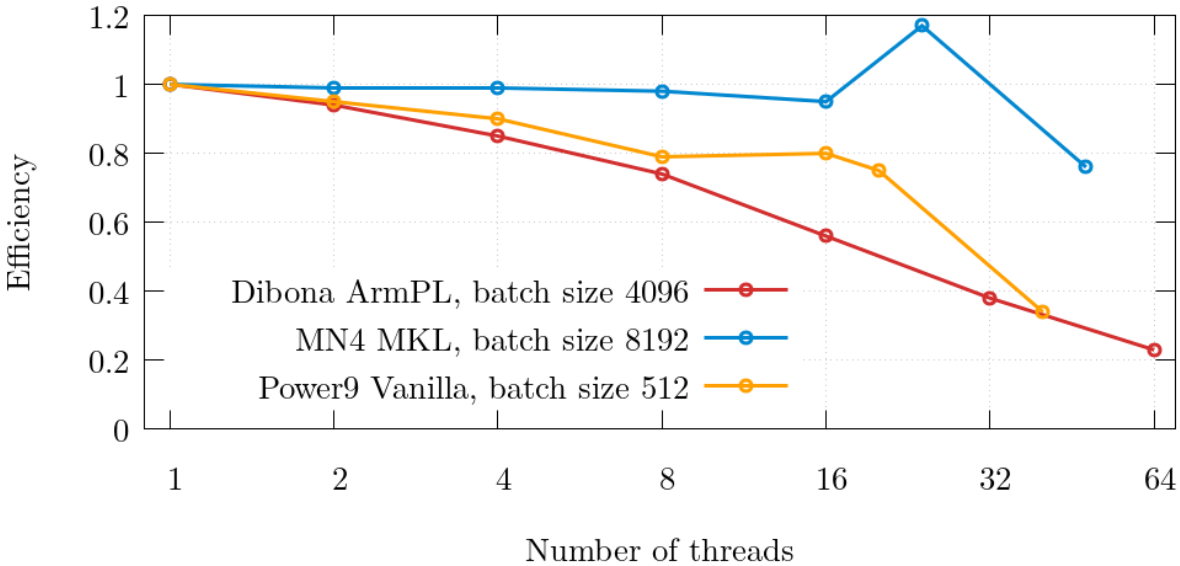


Figure 16: Efficiency of AlexNet thread scalability in the different platforms

Figure 16 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.

6.2.2 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.

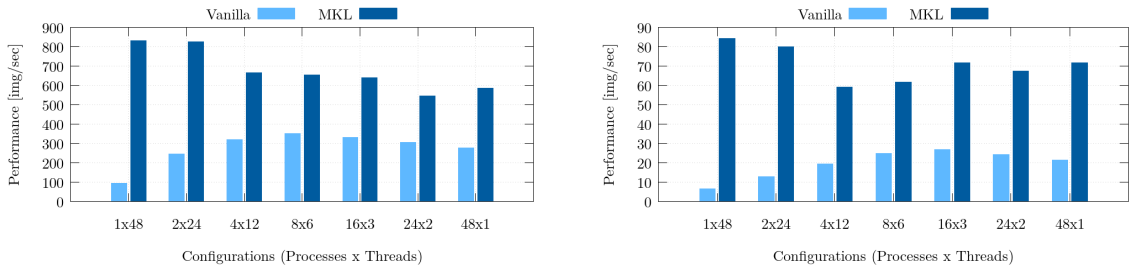
For this evaluation, we use two different models: AlexNet and ResNet-50, and we vary the distribution of resources among processes and threads always using all the computational resources available.

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 11 we can see the different batch sizes used for each configuration on each HPC platform.

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	

Table 11: Batch size assigned to each process, for each configuration of process × thread



(a) Performance of hybrid configuration of AlexNet model in MareNostrum4

(b) Performance of hybrid configuration of ResNet-50 model in MareNostrum4

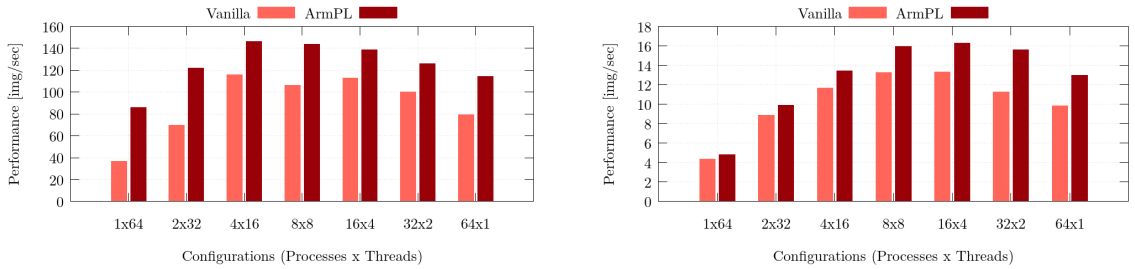
Figure 17: Performance of hybrid configuration in MareNostrum4

In Figure 17a 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 17b 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.



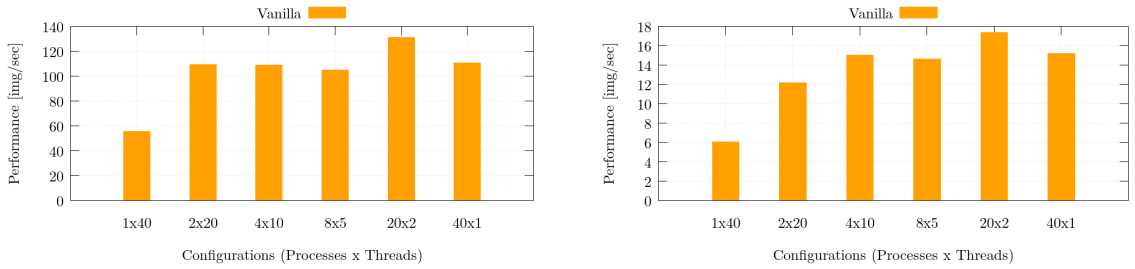
(a) Performance of hybrid configuration of AlexNet model in Dibona

(b) Performance of hybrid configuration of ResNet-50 model in Dibona

Figure 18: Performance of hybrid configuration in Dibona

In Figure 18a 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 configuration in both cases is using 4 MPI processes and 16 threads per MPI process.

Figure 18b 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.



(a) Performance of hybrid configuration of AlexNet model in Power9

(b) Performance of hybrid configuration of ResNet-50 model in Power9

Figure 19: Performance of hybrid configuration in Power9

Figures 19a and 19b 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.

6.3 Internode evaluation

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

In Figure 20 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

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

Table 12: Configurations used for each cluster and model.

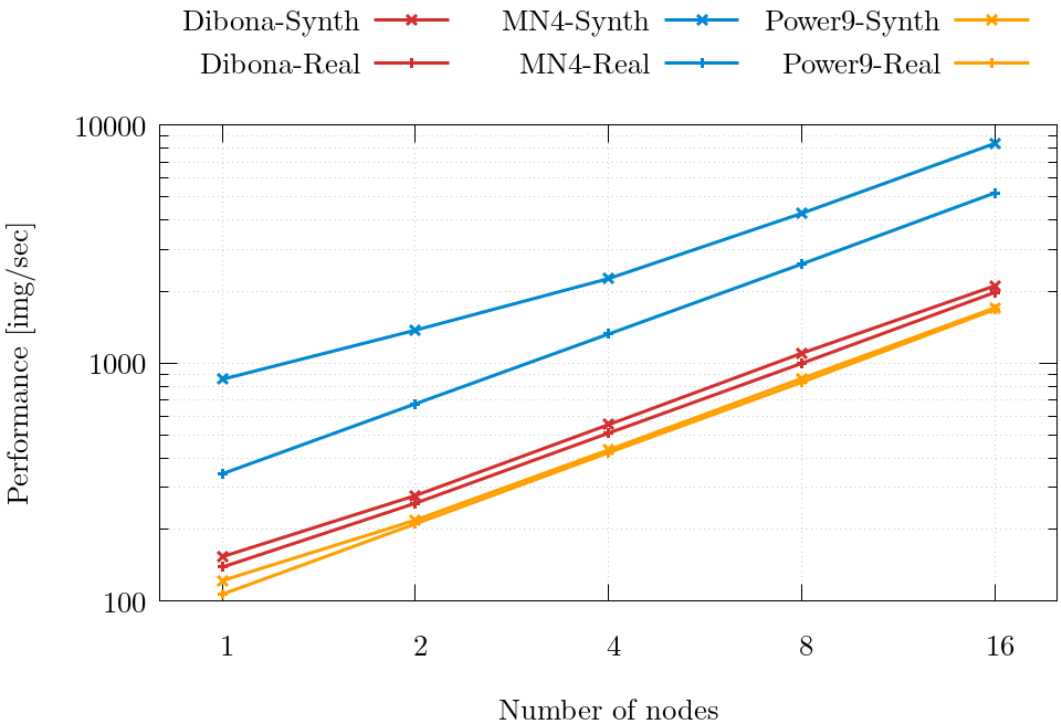


Figure 20: Scalability of Alexnet model on the different platforms

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.

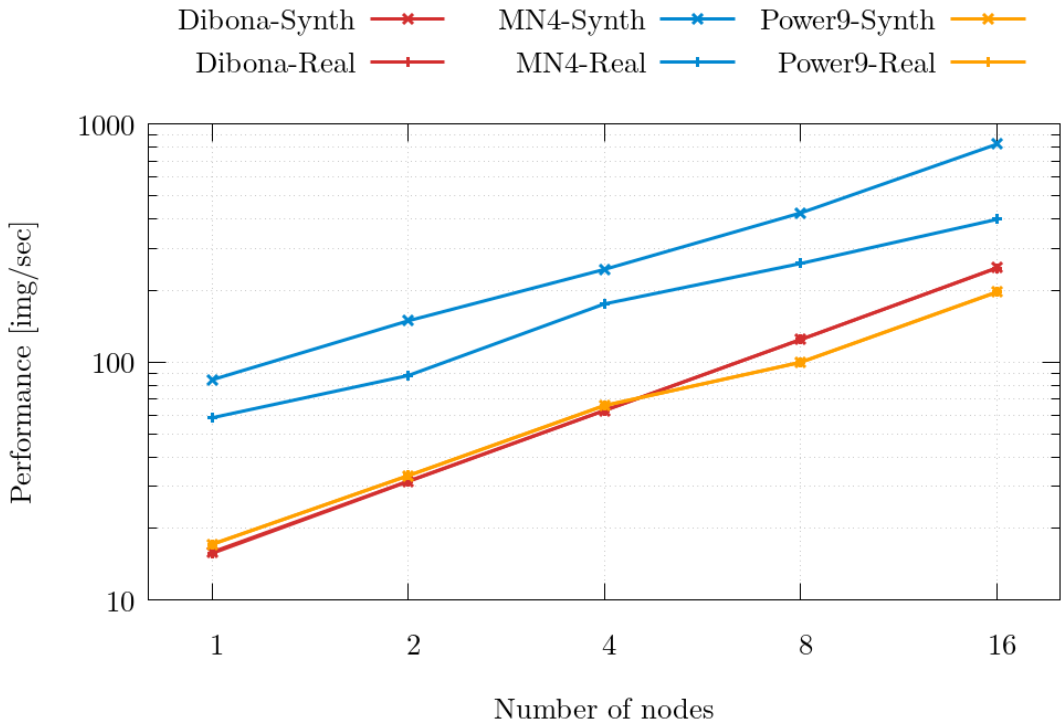


Figure 21: Scalability of ResNet-50 model on the different platforms

Figure 21 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 because of its deeper 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 technology, Dibona and Power9 use Infiniband while MareNostrum4 uses Intel OnmiPath. 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 20 and 21 can be complemented with the efficiency heat map shown in Table 13. 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 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 13 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.

6.4 Conclusions

One of the main goals is to evaluate TensorFlow coupled with an algebra library optimized for Arm. As proof of concept, we plug the Arm Performance Libraries into some of the most compute-intensive kernels of TensorFlow. We evaluate 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

	# 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

Table 13: Parallel efficiency AlexNet and ResNet-50 models on the different platforms

latest Arm CPU, and we measured a speed-up between 1.5× and 2.3×.

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-DNN on x86, we measure a performance improvement between 1.7× and 6.9×.

Concerning parallelization strategies with processes and threads, we notice 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-DNN 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× 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.

7 Conclusions

While the technical conclusions have been presented in Section 6.4, this section is dedicated to my personal conclusions about my TFG project experience.

7.1 Summary of the results

7.1.1 Project planning

Thanks to this project I acquired new knowledge, the tools and the possibility to make a plan of the project with many points of view, from taskify to human resources. It allowed me to study the state-of-the-art of a field of study and evaluate others researcher works. Also, make me the possibility to create and think about the scope and organization of four month project. Other important parts that enrich me are thinking about project tasks and describe them precisely in content and time. The steps performed to pass from main ideas to split into small tasks is a very practical skill that I have developed. To make the project possible is necessary to know what are the risks and the hardness of every task and have possible alternatives to keep the project going on. Knowing and planning the risks before doing the task is a very difficult job. Finally, during the project planning, I take into account the amount of money and human resources that a project would have, making a quantitative approach to these resources.

7.1.2 Analysis

This project helps me to make a naive performance analysis of hundreds of thousands of lines state-of-the-art AI framework. I read the official documentation to know how the user interface works and acquiring the methodology to identify the most time-consuming routines. Also, leverage to debugging the new changes to make sure we performed the same operation after the changes made in the code.

7.1.3 Evaluation

For the evaluation conclusions, I learn the importance of making a good representation of the state of the software state and hardware real performance. We make the tests with multiple build configurations and doing the experiments with generic and specifics linear algebra libraries. In the Machine Learning world and concrete in TensorFlow, the importance of using optimized linear algebra libraries could make the difference, we see differences of 6.9× in the Intel platform. Moreover, we try different hybrid configurations in intra-node that in case of the Arm platform, we get a speedup of 1.6× to regard a more standard Machine Learning configuration with one process per node. Another conclusion is about the batch-size, we see the performance is getting higher with bigger batch-sizes this will be useful for our Machine Learning training with CPUs because we have more memory available than the typical GPUs.

7.2 Contribution to the community

The contributions to the community are performed in different ways. This journey starts within the context of the Mont-Blanc 3 European project [14]. In particular, I developed a section inside a report for the work package 6 [15]. After that, we publish a paper into a conference presenting the evaluation inside the HPML2019 workshop of IEEE/ACM CCGRID 2019 conference [26]. Finally, I get contacted by Arm researcher in order to publish the code changes in the Arm HPC Users Group repositories [16] where the code is available for everyone with simple instructions to get deployed the TensorFlow installation.

[14] https://cordis.europa.eu/project/id/671697

[15] https://www.montblanc-project.eu/wp-content/uploads/2019/02/MB3_D6.9_Performance-analysis-of-applications-and-benchmarki
v1.0.pdf

[16] https://gitlab.com/arm-hpc/packages/-/wikis/packages/tensorflow

7.3 Further work

For the future works, it will be helpful to find a Machine Learning benchmark that would be independent of the frameworks from operations and depending on Machine Learning frameworks to performing training benchmarks. This benchmarks would help us benchmarking from a pure microarchitectural point of view to real uses of machine learning models. Also, the other next steps will be focusing on performance analysis of the frameworks applying old and new HPC techniques to be able to explain the actual behaviour of the Machine Learning frameworks. With this knowledge, it would be easier to contribute and help developers to improve the performance of the frameworks.

References

[1] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[2] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[3] Laura García Cuenca, Javier Sanchez-Soriano, Enrique Puertas, Javier Fernandez Andrés, and Nourdine Aliane. Machine learning techniques for undertaking roundabouts in autonomous driving. *Sensors*, 19(10):2386, 2019.

[4] Natalie Stephenson, Emily Shane, Jessica Chase, Jason Rowland, David Ries, Nicola Justice, Jie Zhang, Leong Chan, and Renzhi Cao. Survey of machine learning techniques in drug discovery. *Current drug metabolism*, 20(3):185–193, 2019.

[5] Ethem Alpaydin. *Introduction to machine learning*. 2020.

[6] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[8] Nitin A Gawande, Jeff A Daily, Charles Siegel, Nathan R Tallent, and Abhinav Vishnu. Scaling deep learning workloads: Nvidia dgx-1/pascal and intel knights landing. *Future Generation Computer Systems*, 2018.

[9] Moustafa Alzantot, Yingnan Wang, Zhengshuang Ren, and Mani 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*, pages 7–12. ACM, 2017.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[11] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[12] Top500, November 2018.

[13] Fabio Banchelli, Marta Garcia, Marc Josep, Filippo Mantovani, Julian Morillo, Kilian Peiro, Guillem Ramirez, Xavier Teruel, Giacomo Valenzano, Joel Wanza Weloli, Jose Gracia, Alban Lumi, Daniel Ganellari, and Patrick Schiffmann. MB3 D6.9 – Performance analysis of applications and mini-applications and benchmarking on the project test platforms. Technical report, 2019. Version 1.0.

[14] Sandia National Laboratory, November 2018.

[15] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1(4):216–222, 2018.

[16] Gu-Yeon Wei, David Brooks, et al. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

[17] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500*, 2019.

[18] Amrita Mathuriya, Deborah Bard, Peter Mendygral, Lawrence Meadows, James Arnemann, Lei Shao, Siyu He, Tuomas Kärnä, Diana Moise, Simon J Pennycook, et al. Cosmoflow: Using deep learning to learn the universe at scale. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 819–829. IEEE, 2018.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] Stefan Chmiela, Huziel E Sauceda, Klaus-Robert Müller, and Alexandre Tkatchenko. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature communications*, 9(1):1–10, 2018.

[21] Botros N Hanna, Nam T Dinh, Robert W Youngblood, and Igor A Bolotnov. Coarse-grid computational fluid dynamic (cg-cfd) error prediction using machine learning. *arXiv preprint arXiv:1710.09105*, 2017.

[22] M Cummins Lancaster and EA Sobie. Improved prediction of drug-induced torsades de pointes through simulations of dynamics and machine learning algorithms. *Clinical Pharmacology & Therapeutics*, 100(4):371–379, 2016.

[23] Nikola Rajovic, Alejandro Rico, Filippo Mantovani, Daniel Ruiz, Josep Oriol Vilarrubi, Constantino Gomez, Luna Backes, Diego Nieto, Harald Servat, Xavier Martorell, et al. The mont-blanc prototype: an alternative approach for hpc systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 444–455. IEEE, 2016.

[24] Richard E Fairley and Mary Jane Willshire. Iterative rework: the good, the bad, and the ugly. *Computer*, 38(9):34–41, 2005.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[26] Guillem Ramirez-Gargallo, Marta Garcia-Gasulla, and Filippo Mantovani. Tensorflow on state-of-the-art hpc clusters: a machine learning use case. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 1–8. IEEE, 2019.

A TensorFlow modifications

A.1 tf-offlinator.py

```
import re, os, requests
[...]
def getIndexAndURLsFromTargets(targets):
    index = []
    targetsURLs = []
    for i, target in enumerate(targets):
        matched = re.search("urls\s*=\s*[^]]+],",target)
        if matched is None:
            continue
        urls =  re.findall('(?:http|https)://.*?\"',matched.group())
        index.append(i)
        urlsGood = [x[:-1] for x in urls]
        targetsURLs.append(urlsGood)
    return (index, targetsURLs)
[...]
def download(urls):
    for i, url in enumerate(urls):
        response = requests.head(url)
        if int(response.status_code/100) != 2 and int(response.status_code/100) !=
        ↪ 3:
            continue
        file_name = url.rsplit('/', 1)[-1]
        file_name = './TFdeps/' + file_name
        with open(file_name, "wb") as file:
            response = requests.get(url)
            file.write(response.content)
        return file_name
    return "ERR"
[...]
def addURLsToBzlFile(filePath, ori, add):
    with open(filePath, 'r') as file :
        filedata = file.read()
    matched = re.search("urls\s*=\s*[^]]+],",ori)
    if matched is None:
        return -1
    urls = matched.group()
    addOffline = urls[:-2] + '\"file:' + add + '\",' + urls[-2:]
    filedata = filedata.replace(urls, addOffline)
    with open(filePath, 'w') as file:
        file.write(filedata)
[...]
if __name__ == "__main__":
    getLoad('WORKSPACE')
```

Listing 5: tf-offlinator.py

A.2 third_party/ArmPL

In this subsection are described the code in the third_party/ArmPL folder:

```
licenses(["permissive"])

config_setting(
    name = "use_armpl",
    define_values = {
        "use_armpl": "true",
    },
    visibility = ["//visibility:public"],
)

config_setting(
    name = "use_armpl_debug",
    define_values = {
        "use_armpl": "true",
        "use_armpl_debug": "true",
    },
    visibility = ["//visibility:public"],
)

load(
    "//third_party/ArmPL:build_defs.bzl",
    "if_armpl",
    "if_armpl_debug",
)

cc_library(
    name = "arm_binary_blob",
    visibility = ["//visibility:public"],
    copts = ["-fopenmp"],
    linkopts = ["-lgfortran","-lm"],
    deps = [
        "@arm_linux//:armpl_headers",
        "@arm_linux//:armpl_libs_linux",
    ],
)
```

Listing 6: third_party/ArmPL/BUILD

```
licenses(["notice"])

exports_files(["share/LICENSE"])

filegroup(
    name = "LICENSE",
    srcs = [
        "share/LICENSE",
    ],
    visibility = ["//visibility:public"],
)

cc_library(
    name = "armpl_headers",
    srcs = glob(["include_mp/*.h"]),
    includes = ["include"],
    visibility = ["//visibility:public"],
)

cc_library(
    name = "armpl_libs_linux",
    srcs = [
        "lib/libarmpl_mp.so",
    ],
    visibility = ["//visibility:public"],
)
```

Listing 7: third_party/ArmPL/ArmPL.BUILD

```
def if_armpl(if_true, if_false = []):
    return select({
        str(Label("//third_party/ArmPL:use_armpl")): if_true,
        "//conditions:default": if_false,})

def if_armpl_debug(if_true, if_false = []):
    return select({
        str(Label("//third_party/ArmPL:use_armpl_debug")): if_true,
        "//conditions:default": if_false,})
```

Listing 8: third_party/ArmPL/build_defs.bzl

A.3 core/debug

In this subsection are described the code in core/debug/debug_gemm.cpp:

```cpp
#include "tensorflow/core/debug/debug_gemm.h"

template<typename T>
bool combinedToleranceCompare(T x, T y)
{
    T maxXYOne = std::max( (T)1, std::max( std::abs(x) , std::abs(y))) ;

    return std::abs(x - y) <= std::numeric_limits<T>::epsilon()*maxXYOne ;
}

template<typename T>
bool absoluteToleranceCompare(T x, T y)
{
    return std::abs(x - y) <= std::numeric_limits<T>::epsilon() ;
}

template <typename T>
bool areEqualMatrix(const T* a, const T* b, unsigned length){
        for(unsigned i = 0; i < length; i++){
                if(a[i] != b[i]) return false;
        }
        return true;
}

template <>
bool areEqualMatrix<Eigen::half>(const Eigen::half* a, const Eigen::half* b, unsigned length){
        return true;
}

template<>
bool areEqualMatrix<float>(const float* a, const float* b, unsigned length){
        for(unsigned i = 0; i < length; i++){
                if(!combinedToleranceCompare<float>(a[i],b[i])) return false;
        }
        return true;
}

template<>
bool areEqualMatrix<double>(const double* a, const double* b, unsigned length){
        for(unsigned i = 0; i < length; i++){
                if(!combinedToleranceCompare<double>(a[i],b[i])) return false;
        }
        return true;
}
```

Listing 9: core/debug/debug_gemm.cc