



# Performance benchmarking of deep learning framework on Intel Xeon Phi

Chao-Tung Yang<sup>1</sup> · Jung-Chun Liu<sup>1</sup> · Yu-Wei Chan<sup>2</sup> · Endah Kristiani<sup>3,4</sup> · Chan-Fu Kuo<sup>1</sup>

Published online: 17 June 2020

© Springer Science+Business Media, LLC, part of Springer Nature 2020

## Abstract

With the success of deep learning (DL) methods in diverse application domains, several deep learning software frameworks have been proposed to facilitate the usage of these methods. By knowing the frameworks which are employed in big data analysis, the analysis process will be more efficient in terms of time and accuracy. Thus, benchmarking DL software frameworks is in high demand. This paper presents a comparative study of deep learning frameworks, namely Caffe and TensorFlow on performance metrics: runtime performance and accuracy. This study is performed with several datasets, such as LeNet MNIST classification model, CIFAR-10 image recognition datasets and message passing interface (MPI) parallel matrix-vector multiplication. We evaluate the performance of the above frameworks when employed on machines of Intel Xeon Phi 7210. In this study, the use of vectorization, OpenMP parallel processing, and MPI are examined to improve the performance of deep learning frameworks. The experimental results show the accuracy comparison between the number of iterations of the test in the training model and the training time on the different machines before and after optimization. In addition, an experiment on two multi-nodes of Xeon Phi is performed. The experimental results also show the optimization of Xeon Phi is beneficial to the Caffe and TensorFlow frameworks.

**Keywords** Intel Xeon Phi · Tensorflow · Caffe · Deep learning · Docker

---

✉ Chao-Tung Yang  
ctyang@thu.edu.tw

<sup>1</sup> Department of Computer Science, Tunghai University, Taichung 40704, Taiwan, ROC

<sup>2</sup> College of Computing and Informatics, Providence University, Taichung City, Taiwan, ROC

<sup>3</sup> Department of Industrial Engineering and Enterprise Information, Tunghai University, Taichung 40704, Taiwan, ROC

<sup>4</sup> Department of Informatics, Faculty of Engineering and Computer Science, Krida Wacana Christian University, Jakarta 11470, Indonesia

## 1 Introduction

Deep learning (DL) technology has blossomed in recent years due to its success in diverse applications, such as speech recognition [29, 48], computer vision [25, 42], object detection [16, 49] and natural language processing [8], etc. The success of deep learning technology is attributed to its high representational ability of input data, by using various layers of artificial neurons. However, training these deep neural network (DNN) models requires a vast amount of computational resources [24, 27].

In the past decade, CPU computing power is increasing significantly. However, with the increasing of complexity of scientific computing, improving the CPU computing power seems to be inadequate. Recently, graphics processing units (GPUs) serve as one of the most popular hardware to accelerate the training speed of DNNs. Different from the conventional CPU, a typical GPU is generally equipped with thousands of cores and large Gigabytes of memory bandwidth, which significantly accelerates the training and reasoning speed of DNNs compared to the traditional CPU. Thus, many users use the GPU as the computational accelerator in the big data analysis. Users have to use the CUDA programming language for using the GPU hardware framework. However, it is difficult for users to learn the CUDA language and to reuse algorithms by using it. Nowadays, Intel introduces the Xeon Phi processor family based on the x86 core architecture. Each core of Xeon Phi supports four hardware threads. The feature is to use C or C++ programming language. When a user adds a simple parameter using the compiler, it can be executed on the multiple consolidation core architecture (MIC). In addition, it supports open multiprocessing (OpenMP), POSIX threads (PThread), message passing interfaces (MPI), and other parallel programming languages. Compared with the GPU, it only needs to pay a small amount of overhead can achieve the same performance [34].

With the increasing popularity of the deep learning methods over the last few years, several deep learning software frameworks have been proposed to facilitate the usage of these methods. These frameworks, such as Caffe, DeepLearning4J, TensorFlow, Theano and Torch are used to optimize different aspects of training and deployment of deep learning algorithms. Choosing a framework depends on various factors, such as community and support, ease of use, prototyping, industry, and embedded computer vision. Caffe as one of deep learning framework is a good case study for computer vision. Computer vision case represented a vast amount of data processing that suitable for benchmarking the performance of the hardware [1, 45].

With the strong backends of GPU hardware framework, developers have constantly improved these frameworks by adding more features and improving speed for attracting more and more users to use these frameworks for different applications. Recently, the efficacy of deep learning software frameworks have been evaluated and proposed [26, 36]. However, the current evaluation results are mostly focused on speed performance of the convolutional frameworks.

In this work, two deep learning frameworks, which are Caffe and TensorFlow running on Xeon Phi are evaluated. In addition, the optimization of Xeon Phi

applying in deep learning jobs is demonstrated. In this case, the use of vectorization, OpenMP parallel processing, and Message Passing Interface (MPI) are examined to improve the performance of deep learning framework. Finally, in the experimental results, which consist of the accuracy comparison between the number of iterations of the test in the training model and the training time on the different machines before and after optimization. In addition, the experiment with two Xeon Phi multi-nodes is evaluated. The results with respect to the measuring metrics are shown in the section of experimental results. The main contributions of this work are summarized as follows.

- We evaluated the accuracy of the Caffe Deep Learning Framework with LeNet MNIST Classification Model training and testing data.
- We evaluated the performance of TensorFlow framework on Intel Xeon Phi 7210 with CIFAR-10 image recognition datasets.
- We evaluated the performance of Docker containers on Intel Xeon Phi 7210 with MPI parallel matrix-vector multiplication.

The remainder of this work is organized as follows. Section 2 presents the literature review and related works. In Sect. 3, the system design and implementation are presented. The experimental results are shown in Sect. 4. In addition, the discussion are also stated in this section. Finally, the concluding remarks are given in Sect. 5.

## 2 Background review and related works

### 2.1 OpenMP

OpenMP is an implementation of multi-threading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. Figure 1 describes the architecture of OpenMP.

The threads then run concurrently, with the runtime environment allocating threads to different processors. The threads then run concurrently, with the runtime environment allocating threads to different processors. The section of code that is meant to run in parallel is marked accordingly, with a compiler directive that will cause the threads to form before the section is executed. Each thread has an id attached to it which can be obtained using a function (called `omp_get_thread_num()`). The thread is an integer, and the master thread has an id of 0. After the execution of the parallelized code, the threads join back into the master thread, which continues onward to the end of the program. By default, each thread executes the parallelized section of code independently. Work-sharing constructs can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both task parallelism and data parallelism can be achieved using OpenMP in this way [46]. Figure 2 shows the OpenMP threads process.

OpenMP contains three components: directives and clauses for compilers, libraries for runtime, and variables for environment. The compiler directives are only

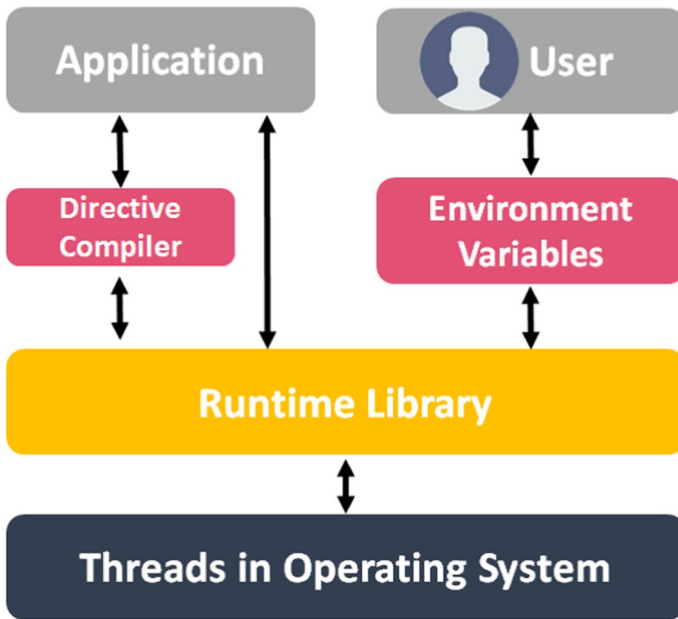


Fig. 1 OpenMP architecture

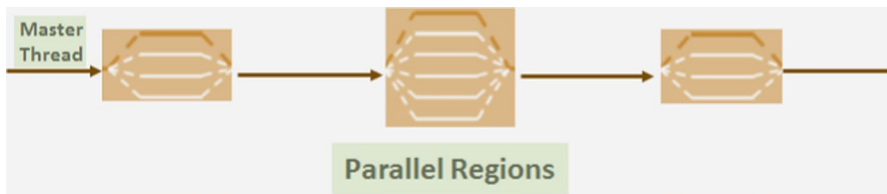


Fig. 2 OpenMP threads process

perceived when the option to compile OpenMP is switched on. OpenMP uses the execution model of “fork and join”: the master thread forks new threads at the start of parallel regions, multiple threads share work in parallel; and threads merge at the end of parallel regions.

## 2.2 Message passing interface (MPI)

MPI [11–13, 31] is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. There are several well-tested and efficient implementations of MPI, many of which are open-source or in the public domain.

These fostered the development of a parallel software industry, and encouraged development of portable and scalable large-scale parallel applications. The architecture of MPI is shown as Fig. 3.

### 2.3 Caffe

The Berkeley Vision and Learning Center (BVLC) and community contributors created Caffe as a deep learning framework. The framework supports Python, C++, MATLAB, and CUDA. Caffe's command line tool has several functions, it can train a model, or use a well-trained model for the effectiveness of the test. When it was training, it would build a Solver object, and its main function was to coordinate the operation of the neural network to carry out training. One can use a configuration file to specify the Solver parameters, such as learning rate or Solver types, like SGD Solver and so on. In the profile, user can specify a training net parameters, testing nets may have more than one. For example, if user want to use different data set to verify the effectiveness of the model can be used. Although the network definition can also be written directly in the Solver configuration file, but the example code is usually written in a separate profile.

Next, Solver will create the corresponding training and testing Net objects based on these profiles. Then Net will according to the definition of the entire network to establish each Layer, also create a lot of Blobs to place the Layer and Layer between the input and output information, and they are connected. Among them, a layer of input is called bottom blobs, the output is top blobs. Blob is basically a multidimensional array, except to its use of data, it contains a corresponding set of Diff, Gradient can be used to calculate the results. These Blobs provide a simple interface that allows Layer to access the data from the GPU or CPU. The entire architecture process is shown in Fig. 4

### 2.4 TensorFlow

TensorFlow is an open-ended machine learning platform. It has an extensive and versatile tool, library and community resources ecosystem that allows scientists to advance the cutting edge in ML and allows ML powered apps to be readily built and deployed by developers. TensorFlow has several abstraction levels so it can choose

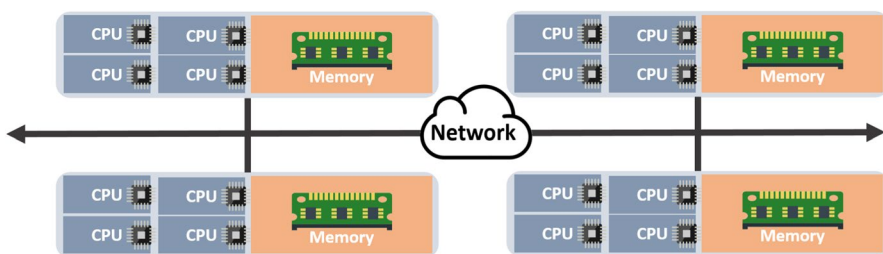


Fig. 3 MPI architecture

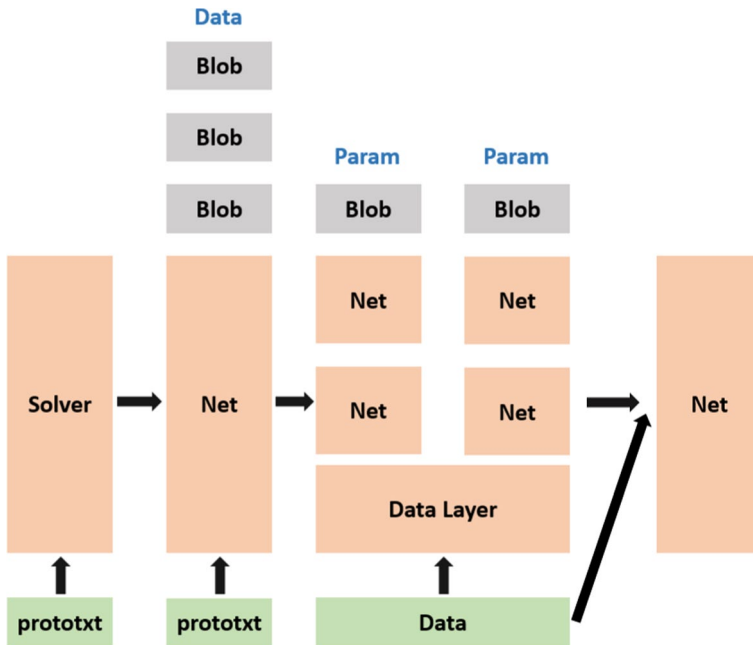


Fig. 4 Caffe architecture

the one to suit your requirements. Use the high-level Keras API to build and work models, making it simple to start with TensorFlow and machine learning. When more flexibility is needed, eagerness to execute iters immediately and intuitively. Use the Distribution Strategy API for distributed training on various hardware settings for major ML training activities without altering the definition of the model [32, 39, 43].

## 2.5 Docker containers

Docker is a collection of paired software products and platforms as a service to create and offer software in packages called containers that use the operating system-level virtually. Docker Engine is the software hosting the containers. This is a conventional software unit, that packages code with all its dependencies so that the application operates fast and reliably from one computer setting to the other. It was first launched in 2013 and is created by Docker Inc. A Docker container picture is an easy, standalone, executable software package which contains all the necessary software for the implementation: code, run-time, system instruments, system libraries and setups[14].

Container images are transformed into containers during run-time and in the Docker case-when running on Docker Engine pictures are converted into containers. Containerized software is always the same, regardless of the infrastructure, and is available for both Linux and Windows applications. Containers isolate software

from its setting and guarantee that, despite variations between growth and staging, it functions in a uniform manner [9, 15, 21].

## 2.6 Related works

Caffe enables experimentation and seamless switching between platforms to facilitate creation and deployment from prototyping machines to cloud environments. Jia et al. [19] separates model representation from real implementation. With the support of an active GitHub group of contributors, Caffe is supported by the Berkeley Vision and Learning Center (BVLIC). It supports study initiatives on the basis of vision, voice and multimedia, large scale industrial apps and start-up prototypes.

Tanno et al. [37] created Caffe2C which converts CNN (Convolutional Neural Network) models trained with the existing CNN framework, Caffe, C-language source codes for mobile devices. Since Caffe2C generates a single C code which includes everything needed to execute the trained CNN, csCaffe2C makes it easy to run CNN-based applications on any kinds of mobile devices and embedded devices without GPUs. Moreover, Caffe2C achieves faster execution speed compared to the existing Caffe for iOS/Android and the OpenCV iOS/Android DNN class. The reasons are as follows: (1) directly converting of trained CNN models to C codes, (2) efficient use of NEON/BLAS with multi-threading, and (3) performing pre-computation as much as possible in the computation of CNNs. In addition, in this paper, they demonstrate the availability of Caffe2C by showing four kinds of CNN-base object recognition mobile applications.

Bottleson et al. [3] presented OpenCL acceleration of a well-known deep learning framework, Caffe, while focusing on the convolution layer which has been optimized with three different approaches, GEMM, spatial domain, and frequency domain. Their research, clCaffe, greatly improves the performance on all types of OpenCL devices to utilize deep learning use cases, specifically on small form factor devices in which discrete GPUs are uncommon and integrated GPUs are far more common. Compared to CPU-only AlexNet on the ImageNet dataset, their benchmark shows 2.5 cm speedup on the Intel Integrated-GPU. As such, our research provides the ability for the deep learning community to adopt a wide range of devices through OpenCL.

CaffePresso, a Caffe-compatible framework for creating an optimized mapping of ConvNet user-supplied requirements to target multiple accelerators such as FPGAs, DSPs, GPUs, RISC multicores, was suggested by Hegde et al. [18]. They use an automated code generation and autotuning strategy based on ConvNet requirements experience, as well as platform-specific limitations such as on-chip memory ability, bandwidth, and potential for ALU. While the Jetson TX1 + cuDNN may be expected to deliver high performance for ConvNet configurations, it shows a slow-move GPU transformation with a faster and energy-efficient implementation on older 28 nm TI KeystoneIIDSP compared to most other systems for smaller embedded-friendliness-based data sets like MNIST and CIFAR10 over new 20 nm NVIDIA TX1 SoC in all instances.

Kurth et al. [23] discussed various alternatives on modern SuperComputing systems for the Tensorflow Framework to be scaled to thousands of nodes.

Tarasov et al. [38] unravel Docker's multi-faceted nature and show its effect on system and working capacity. As we reveal fresh features of the famous Storage Docker drivers, this reminds us that new techniques are used extensively and can often be evaluated in advance.

In order to assess containers acquired from heterogeneous providers, Venkateswaran et al. [41] suggested a fresh metric called fitness quotient (FQ). They leverage machine training methods in injecting automation into the following two stages: the first-level K-mean clustering to correctly categorize IaaS costs and performance information, and the second-level provisioning time polynomial retrenchment to find linkages between SaaS performance and container strength.

Purushotham et al. [33] present the results of benchmarking for several medical forecasting processes such as death prediction, duration of stay analysis, and ICD-9 code category analysis using Deep Learning models, set of machine learning models (Super Learner algorithm), SAPS II and SOFA ratings. Regarding the benchmarking activities, we used the publicly accessible Medical Information Mart regarding Intensive Care III (MIMIC-III) (v1.4) which includes all patients assigned to an ICU at the Beth Israel Deaconess Medical Center from 2001 until 2012. Their findings show that deep learning models consistently outperform all other approaches particularly when the data from the real clinical time series is used as features of input to the models.

### 3 System design and implementation

In the section, the system introduces system design and implementation. In the system design, the dataset used and the system flow are explained in detail. In the system implementation, the vectorization and parallelism of OpenMP are described in detail.

#### 3.1 System design

##### 3.1.1 Caffe deep learning framework

Caffe architecture has been used in this paper with CIFAR-10 [5, 6, 22] complete sigmoid model, CNN model [20, 35, 47] involves convolution, biggest pool, batch standardization, complete connection, multi-layer and softmax layer. The CIFAR-10 dataset is shown in Fig. 5, comprises of 60,000 color pictures, each with  $32 \times 32$ , similarly split and labeled as to the following ten categories of dimensions: catalogue, aircraft, vehicle, bird, frog, deer, horse, dog, (such as sedans or sports utility instruments) or defeated all vehicles (which contain only big vehicles) without overlapping.



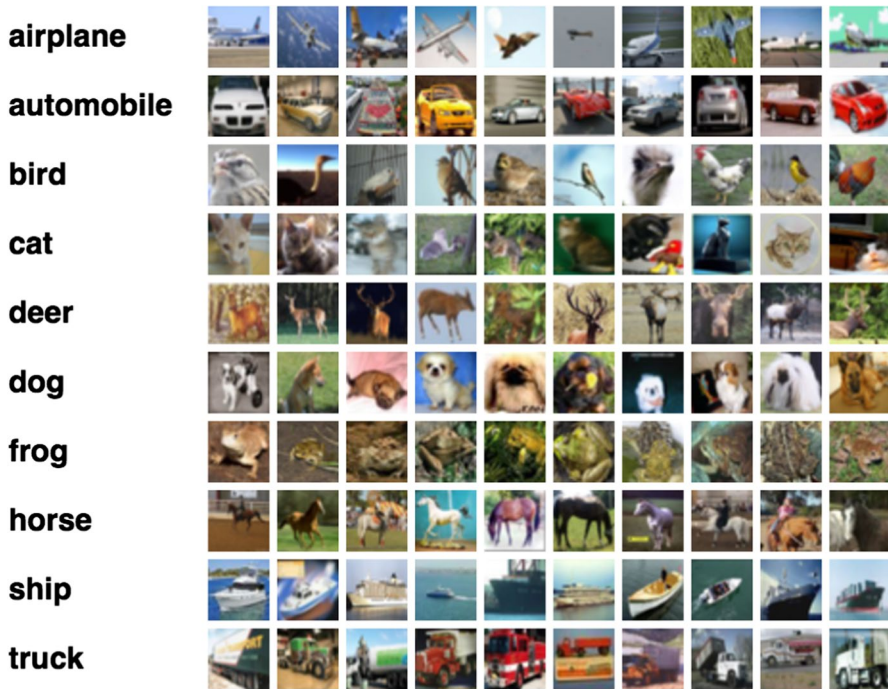


Fig. 5 CIFAR-10 dataset

### 3.1.2 TensorFlow deep learning framework

TensorFlow was implemented for benchmarking the performance of Intel Xeon Phi Processor 7210 Platform. In this case, Cifar 10 was trained on single Bare Metal with Intel MKL-DNN optimized Tensor. For this experiment, tests were done for 1000 steps, for a batch size of 128, and logging frequency of 1.

### 3.1.3 Docker containers benchmark

Docker containers were examined in the experiment on Intel Xeon Phi Processor 7210 Platform. In this case, the experiment run two parallel MPI processes on MPI Matrix action on a vector, with 2000 iterations of size 1000 (length of vector  $v$ ). Demonstrating a MPI parallel Matrix-Vector Multiplication that run the iterations of following equation:

$$v(t + 1) = M * v(t) \quad (1)$$

The  $t$  value is defined as iteration, where  $v$  is a vector of length and  $M$  a dense size.

The other experiments were MPI Latency Test, MPI Bandwidth Test, and MPI Bi-Directional Bandwidth Test.

### 3.1.4 System flow

Caffe framework which optimized for Intel architecture now includes the latest version of Intel Math Core Library (Intel MKL) 2017 Optimized Advanced Vector Extensions (AVX)-2 and AVX-512 instructions to support Intel Xeon with the Intel Xeon Phi (and others) processor. All the benefits found in BVLC Caffe on Intel architectures and training courses that can be used for different nodes effectively. The system flow for the design is

- Install Caffe environment on Xeon Phi Processor.
- Train and test on LeNet MNIST[4, 17]
- Test pre-trained models such as bvlc\_googlenet.caffemodel, certain pictures.
- Fine-tune the Cats vs Dog Challenge the trained model.
- Install TensorFlow and Intel MKL-DNN optimized Tensor environment on Intel Xeon Phi Processor 7210.
- Train and Test CIFAR10 images classification dataset.
- Install Docker environment with MPI Parallel Processing.
- Run two parallel MPI processes on MPI Matrix action, MPI Latency Test, Bandwidth Test, and Bi-Directional Bandwidth Test.

## 3.2 System implementation

### 3.2.1 Vectorization

In the analysis of the BVLC Caffe code, and find the wireless Internet site—function call, consume the maximum CPU time, this project applied the vectorization optimization. These optimizations include the following:

- Basic Linear Algebra Complex (BLAS) [2] Library (Intel MKL to Switch from Auto-Adjust Linear Algebra System [ATLAS] [30])
- Optimized components (Xbyak just-in-time [JIT] [44] group translator)
- GNU Compiler Collection (GCC) and OpenMP code vectorization

BVLC Caffe has used the Intel MKL BLAS feature call or other implementation options. For example, for vectorization, multi-threading, and better cache memory traffic optimization GEMM functions. For better vectorization, this project also use the Xbyak-JIT translator (ia-32) for x86 and x64 (AMD64 or x86-64). Xbyak currently supports vector instruction sets for MMX, Intel SSE, Intel SSE3, Intel SSE4, floating point units, Intel AVX, Intel AVX2 and Intel avx-512.

The Xbyak translator is an C++ x86/x64 JIT translator, especially for libraries that efficiently develop code. The code that is executed only on the title is provided by the Xbyak group translator. It can also dynamically combine x86 and x64 amusement keys. The JIT binary code generated by the code is executed while allowing several optimizations, quantization, such as using a job that can be used to specify

the array of elements of the second array, with the polynomial calculated item, Stable, variable  $x$ , new, sub, mul, div, etc. Intel Advanced Vector Extensions and Intel AVX2 Vector Instruction Set support, Xbyak can achieve a better vectorization of Caffe's optimized for Intel architecture. The latest version of Xbyak with Intel avx-512 vector instruction set support, which can improve operational efficiency, using Intel Xeon Phi processor  $\times 200$  products. This improved vectorization ratio allows Xbyak to process more information, along with single instruction, multiple data (SIMD) instructions, and more efficient use of data parallel processing. The use of Xbyak vector for this job can improve the performance of the large shared layer of the program. If know the parameters of the cluster, the code of the component can be generated to handle the particular shared model that applies to a particular shared window or shared algorithm. The result is that the proven, more efficient than the C++ code is superior to the general component.

### 3.2.2 Parallelism and OpenMP

OpenMP threads parallel processing is used to optimize the neural networks layers in the following list:

#### *Convolution layers*

Convolution layers learn to weigh or filter, as the name indicates, with each program input producing a function graph in the picture output. This optimization protects a hardware group from using the infrequent input feature.

#### *Shared or Sub-sampling*

The largest pool, the average area, and the stochastic area are different methods that can down-sampling the most popular methods with the largest pool. The common layers are usually not overlapping with the results of a layer of rectangle dynamic bricks. Each of these sub-regions, the layer re-output, the maximum value, the arithmetic meaning, or the stochastic value of the samples formed by each partition is enabled for multinomial delivery. The Pooling function is useful for CNNs in three main reasons:

- The area can be reduced and the dimension of the layer at the top right of the load is calculated.
- The lower level of shared functionality allows the core convolutional to be higher in multi-layered coverage of larger areas of input data and thus learn more complex functions. For example, lower-level cores usually learn to identify small edges, while high-level cores may learn to judge forests or beaches.
- The largest pool can provide some form of translation invariance. Eight possible directions, a  $2 \times 2$  partition (a typical partition of the area) can convert it to a single pixel, from three will return the same maximum.  $3 \times 3$  windows, the five will not return the same maximum value.

Pooling a single function on the map of the job mode, Xbyak was used to build efficient programs with the largest average shared one or more input feature maps. This set of programs can be implemented as a batch input function corresponding to the execution program when parallel to OpenMP.

Shared levels are parallel and multi-threaded; OpenMP images are independent because they can handle different threads in parallel.

#### *Softmax and the loss layer*

The lost (cost) function is a key component that compares the predicted output to the target or the label that will guide the network training program to the machine, and then readjusting the calculation of the gradient to minimize the cost, for the weighting part of the lost part of the derivative project. Softmax [10, 40] (through the normalization index) is the classification of the probability of distribution gradient—logon normalization program function. In general, this is used to calculate the possible results of a random event that allows one of the possible outcomes of  $K$ , with the probability of specifying each result individually. Specifically, in the multinomial logistic regression (multilevel classification problem), the input of this function is the result of a different linear function of  $K$  and the possibility of  $j$  prediction. For example the vector  $x$  class is:

$$P(y = j | x) = \frac{e^{x^T w_j}}{\sum_{k=1}^K e^{x^T w_k}} \quad (2)$$

where  $T$  is exponentiation value, where  $w$  is a vector of the inputs to the output layer (if it has 10 output units, then there are 10 elements in  $z$ ,  $j$  indexes the output units), so  $j = 1, 2, \dots, K$ .

Multi-threading, OpenMP applies these calculations, is a bifurcation of a specific number of subordinate threads, and a way of working between them to use the main thread parallel processing. Threads are executed at the same time, and they are assigned to different processors.

#### *Rectified Linear Unit (ReLU)*

ReLU [7, 28] presently use the deep learning algorithm's most common non-linear features. Allows the element-wise neuron layer operator to place a reduced point block and generate the same size top dot. (Integrated memory interface point-for-architecture with a standard array. Using the product and derivative information via the Internet, Caffe storage, communication, and management information.) The ReLU layer must enter the value  $x$  as  $x$  positive values to calculate the output and extend it to negative slope adverse values:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \text{negativeslope} * x, & \text{otherwise} \end{cases} \quad (3)$$

If  $x = 0$  then the function is not smooth. This is the reason why the derivative of the ReLU function is not defined at  $x = 0$ .

## 4 Experimental results

In this section, the experiments are demonstrated. The experimental environment, including experimental hardware and software, and experimental design are described in detail.

## 4.1 Experimental environment

In the experimental environment, there are two kinds of experiments. Single node, including one Xeon E5 and one Xeon Phi 7210 spect processor. Multinode distributed training, including two Xeon E5 and two Xeon Phi 7210 spect processor were used.

## 4.2 The experimental hardware

In this experiment, the employed hardware are presented in detail in Table 1.

## 4.3 Experimental software

In the experimental software, list of the software version were used in the experiments, and describe its function. The detail can be shown in the Table 2.

## 4.4 Caffe framework benchmark

The LeNet is trained in this section, which is Caffe's MNIST Classification Model. Start the experiment with the following primary phase: dataset preparation, model training, and model timing. First, download the MNIST dataset and generate the LMDB format dataset. Next to train the dataset, the amount of the steps is set to 1 K in order to run fast. Then, running the iteration of propagations forward and backward in 50, 100, 500, 1000, 2000, and 10.000 iteration. Finally, in the validation test, the trained model is examined. Figure 6 demonstrates the findings.

Then, in two platforms, Intel Xeon E5-2560 and Intel Xeon Phi 7210, were benchmarked using BLVC Caffe and Intel optimized Caffe. The time command calculates the forward and backward propagation time of the layer-by-layer. It estimates the time spent in each layer and provides a distinct model with the comparative execution times. The results are presented in Figs. 7 and 8.

Use the distributed multi-node training on two Intel Xeon Phi 7210, as well. The training can be distributed across two primary methods: parallel model and parallel data. The model is split between the nodes in model parallel and each node has the complete data batch. The data batch is split between the nodes in data parallelism,

**Table 1** Hardware specification

	Intel Xeon E5-2650	Intel Xeon Phi 7210
CPU clock	2 GHz	1.30 GHz
CPU core	12 core	64 core
RAM	132 GB	384 GB
Disk	1 TB	10 TB
OS	CentOS 7.2	
Linux kernel	2.6.32-504.el6.x86_64	3.10.0-327.el7.x86_64

**Table 2** Software specification

Name	Version	Description
Intel Parallel Studio XE	2017 update 3	Includes compilers, performance libraries, and parallel models optimized to build fast parallel code
Intel Advisor XE	2017 update 2	Intel Advisor XE is a threading prototyping tool for C, C++, C# and Fortran software architects
Intel Inspector XE	XE 2017	Intel Inspector XE is an easy to use memory and threading error debugger for C, C++, C# and Fortran applications that run
Intel VTune Amplifier	2017 update 2	Intel® VTune™ Profiler collects and presents key profiling data with a powerful interface which simplifies its analysis and interpretation
Intel MPI	2017 update	MPI library, along with MPI error checking and tuning to design, build, debug and tune fast parallel code that includes MPI
Intel MPSS	3.8.1	Is necessary to run the Intel Xeon Phi Coprocessor

### LeNet Model Training Result on Intel Xeon Phi 7210

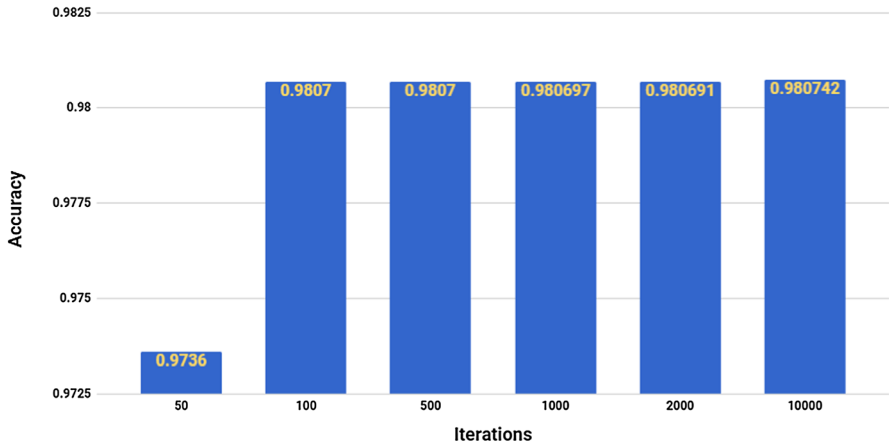


Fig. 6 LeNet model training results

### Caffe CIFAR-10 Dataset Execution Time Output before Intel optimized

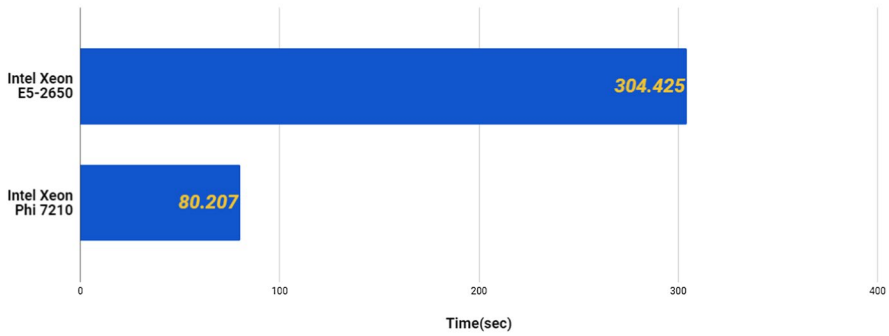


Fig. 7 BLVC Caffe execution time comparison

### Caffe CIFAR-10 Dataset Execution Time Output after Intel optimized

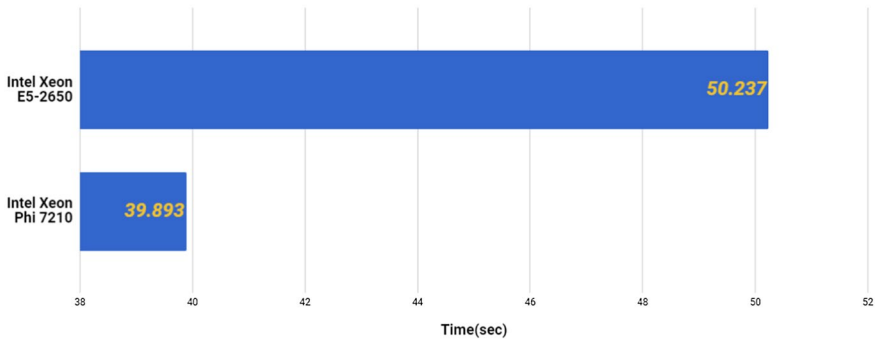


Fig. 8 The execution time comparisons with respect to the Intel optimized Caffe

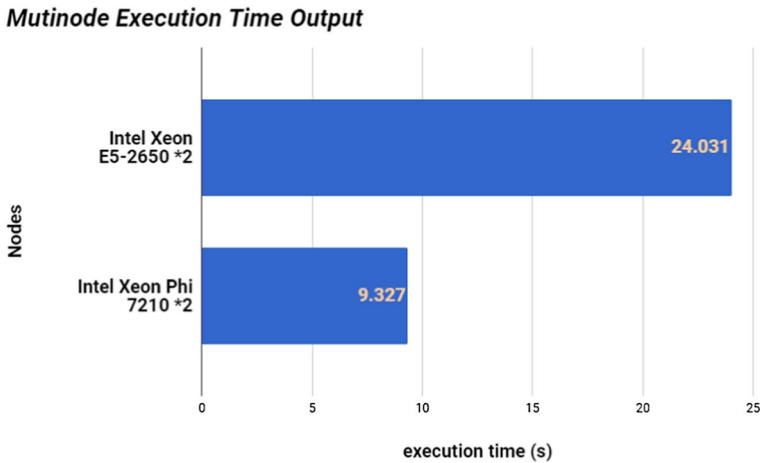


Fig. 9 Multinode execution results

```
Instructions for updating:
Use standard file APIs to check for files with this prefix.
2019-03-31 23:55:18.516048: precision @ 1 = 0.608
```

Fig. 10 The accuracy after 1000 steps

and every node has the complete model. Parallel data is particularly helpful if there is a tiny amount of weights and if the data batch is big. A hybrid model and data parallelism are feasible, where the data-parallel method is used for the training of layers with few weights, such as convolutional layers, and layers with many weights, such as fully-connected layers, with a parallel model strategy. The training results shows as Fig. 9

#### 4.4.1 TensorFlow benchmark

In this experiments, it can be evaluated the training of CIFAR10 image recognition dataset achieved 60.8% accuracy after 1000 steps for a batch size of 128, and logging frequency of 1 as shown in Fig. 10.

In term of speed, the benchmark processes a single batch of 128 images in 1.515–2.599 s (i.e. 49–85 images /s). Figure 11 shows the amount of pictures being processed per seconds.

The model reaches 60% accuracy after 1000 steps in 30 min of training time. Figure 12 shows the graph of image processing in 1758 s per batch on average.

#### 4.4.2 Docker containers benchmark

In the Docker containers performance tests, it based on one mpi head container and three mpi node containers. Figure 13 describes the installation on Intel Xeon Phi.



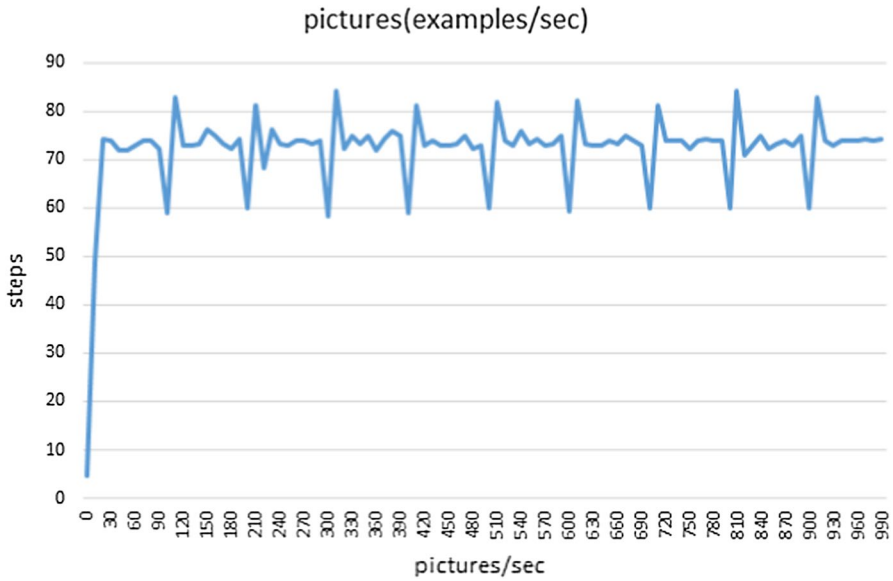


Fig. 11 The amount of pictures being processed per seconds

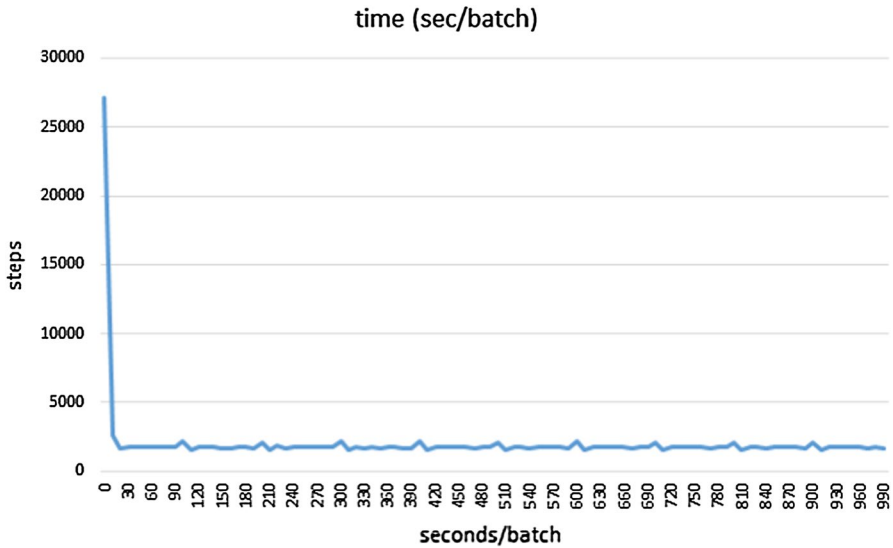


Fig. 12 The image processing time in seconds per batch

First, the experiment was running two parallel MPI Matrix action processes on a vector, 20 iterations of size 1000. Table 3 and Fig. 14 describe the visualization of MPI Matrix action on Docker clusters. Overall, the duration went through the throughput of fluctuation.

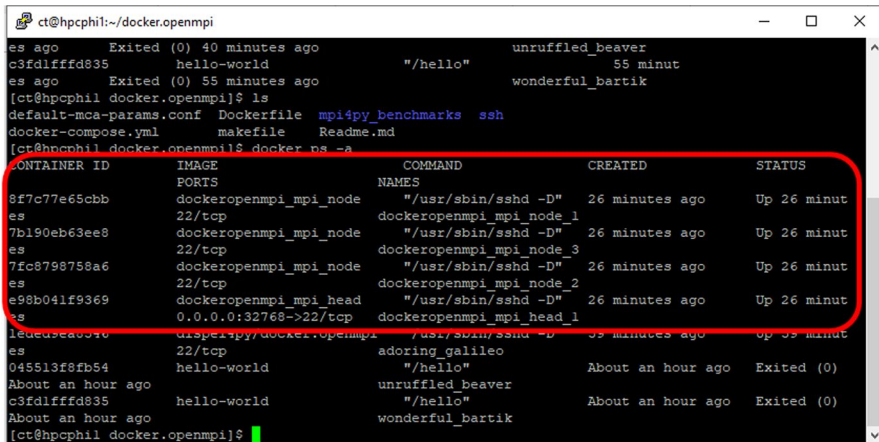


Fig. 13 Docker installation

Table 3 MPI matrix action on a vector, 20 iterations of size 1000

# Duration (s)	Throughput (#/s)
4.725	04.23
4.662	04.29
4.619	04.33
4.694	04.26
4.534	04.41
4.733	04.23
4.657	04.29
4.545	04.40
4.568	04.38
4.681	04.27

Second, in term of MPI latency test, when the size is below than 32,768 B, the latency is stable at 53.18 μs on average. While more than 32,768 B, the latency increasing is random. Table 4 and Fig. 15 shows MPI latency test on Docker clusters.

Third, in term of bandwidth test, the bandwidth increasing in random size. Table 5 and Fig. 16 describes MPI bandwidth test on Docker clusters.

Lastly, in term of bidirectional bandwidth test, when the size is below than 8192 B, the increasing of bandwidth is twice per MB/s. While more than 8192 B, the bandwidth increasing is random. Table 6 and Fig. 17 describes MPI bidirectional bandwidth test on Docker.

### 4.5 Discussion

The training output can be reduced 60.59% at Intel Xeon E5-2650, 20.10% at Intel Xeon Phi 7210 by using the vectorization and parallelism of OpenMP optimization

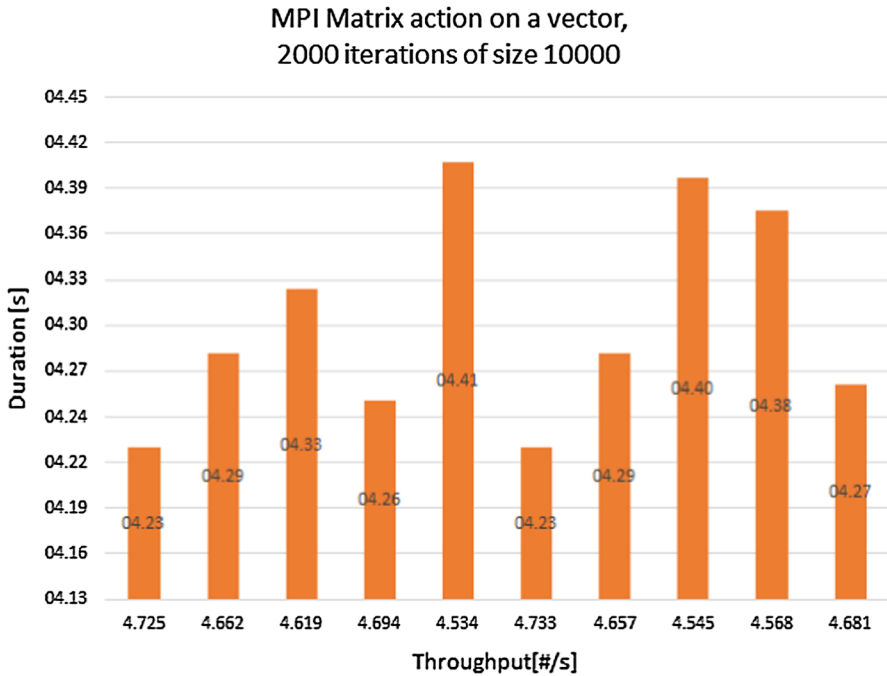


Fig. 14 MPI matrix action on Docker clusters

technique. The optimization was successfully implemented on the Caffe framework, and the training time was reduced significantly. It can be assumed that Intel Xeon E5-2650 has a very bad output without optimization, but it can be reduced by 3795 for Intel Xeon Phi 7210. The training time is decreasing three times after optimization on Intel Xeon E5-2650 comparing with Intel Xeon Phi 7210. In terms of accuracy, we tested the accuracy only on Intel Xeon Phi 7210 on the different iterations of 50, 100, 500, 1000, 2000, and 10.000 iterations. The LeNet model training shows the high accuracy at 0.980742 on Intel Xeon Phi 7210. In addition, using the multi-node of two Intel Xeon Phi 7210, it can be achieved even better performance at 9.327 s.

In the TensorFlow experiments, it can be evaluated from the graph of the training on CIFAR10 image recognition dataset. The amount of the pictures being processed per second and the speed of training processes are quite stable.

In the Docker clusters tests, 20 iterations of size 1000 on two parallel MPI Matrix action processes on a vector. Overall, the duration had passed into the fluctuation throughput. There are a stable phase and unstable phase of processing the specific size on the MPI latency test, bandwidth test, and bidirectional bandwidth test.

**Table 4** MPI latency test

# Size (B)	Latency ( $\mu$ s)
0	49.42
1	51.04
2	51.15
4	51.55
8	51.12
16	51.95
32	51.35
64	48.63
128	49.02
256	49.05
512	50.07
1024	49.89
2048	51.13
4096	52.98
8192	54.16
16,384	58.38
32,768	83.17
65,536	211.59
131,072	289.97
262,144	410.39
524,288	684.62
1,048,576	1116.26
2,097,152	1901.83
4,194,304	3568.79

## 5 Conclusion and future works

This research has optimized Caffe and TensorFlow deep learning frameworks on the Intel Xeon Phi Processor. From the experiments, it can be seen the best practice in the training of big data. The analysis process will be more efficient in terms of time and performance. The presented results can be used as a considerable action on machine learning processing. The optimization was successfully on the Caffe framework usage, and the training time was reduced significantly. In the TensorFlow experiments, the amount of the pictures being processed per seconds and the speed of training processes are quite stable. While in the Docker clusters experiments, there is a stable phase and unstable phase on processing specific size.

In the future, advantage comparison can be performed due to the limitation of this research in which we evaluate three kinds of performance benchmarking and not to do a comparison ranking between Hypervisor and Container. The further improvements and future enhancements might described as follows:

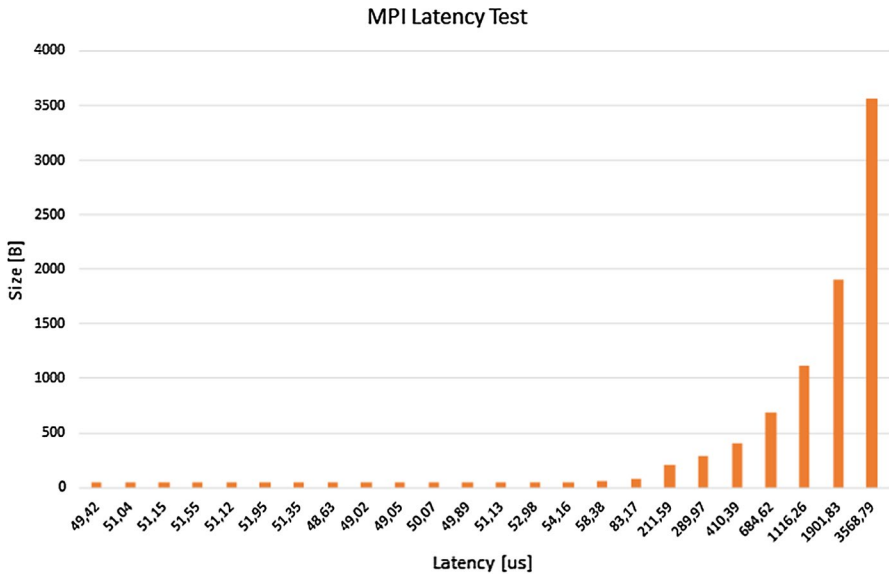
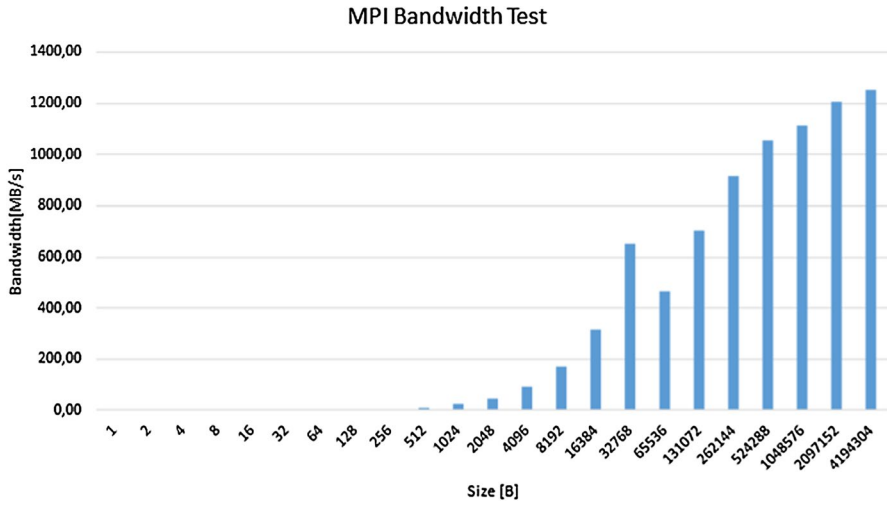


Fig. 15 MPI latency test on Docker clusters

Table 5 MPI bandwidth test

# Size (B)	Bandwidth (MB/s)
1	0.02
2	0.05
4	0.09
8	0.19
16	0.34
32	0.75
64	1.50
128	3.00
256	6.00
512	11.82
1024	23.84
2048	45.88
4096	90.75
8192	171.50
16,384	317.20
32,768	654.34
65,536	467.31
131,072	701.81
262,144	918.13
524,288	1056.43
1,048,576	1113.48
2,097,152	1207.08
4,194,304	1253.42



**Fig. 16** MPI bandwidth test on Docker clusters

**Table 6** MPI bi-directional bandwidth test

# Size (B)	Bandwidth (MB/s)
1	0.02
2	0.04
4	0.08
8	0.16
16	0.31
32	0.63
64	1.25
128	2.49
256	4.82
512	9.98
1024	17.54
2048	37.56
4096	75.06
8192	155.08
16,384	268.05
32,768	363.07
65,536	242.08
131,072	387.76
262,144	463.88
524,288	486.19
1,048,576	533.05
2,097,152	567.72
4,194,304	585.46

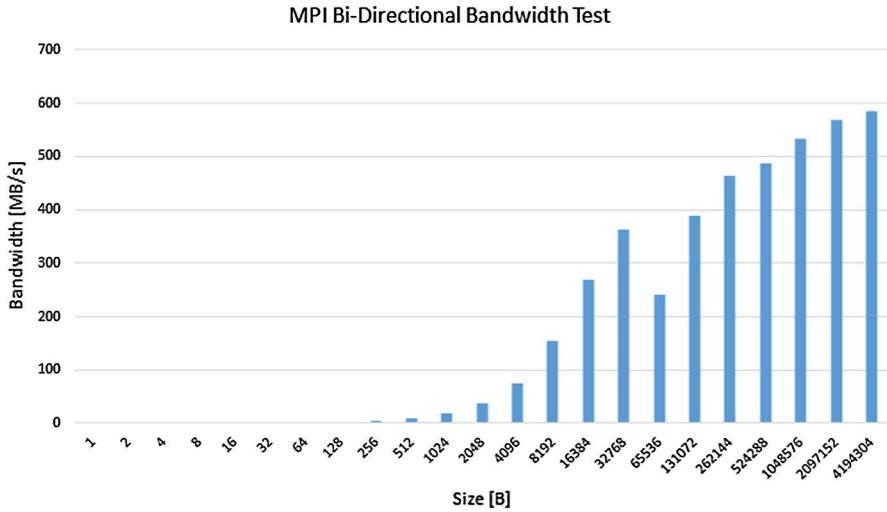


Fig. 17 MPI bidirectional bandwidth test on Docker clusters

- Compare the accuracy of the Caffe Deep Learning Framework with LeNet MNIST Classification Model training and testing data between Hypervisor and Container.
- Compare the performance of TensorFlow Framework on Intel Xeon Phi 7210 with CIFAR-10 image recognition datasets between Hypervisor and Container.
- Compare the MPI parallel Matrix-Vector Multiplication calculation between Hypervisor and Container.

**Acknowledgements** This work was supported by the Ministry of Science and Technology, Taiwan (R.O.C.), under Grant Number 108-2221-E-029-010-.

## References

1. Ben-Nun T, Besta M, Huber S, Ziogas AN, Peter D, Hoefer T (2019) A modular benchmarking infrastructure for high-performance and reproducible deep learning. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 66–77
2. Blackford LS, Petitet A, Pozo R, Remington K, Whaley RC, Demmel J, Dongarra J, Duff I, Hammarling S, Henry G et al (2002) An updated set of basic linear algebra subprograms (blas). *ACM Trans Math Softw* 28(2):135–151
3. Bottleson J, Kim S, Andrews J, Bindu P, Murthy DN, Jin J (2016) Clcaffe: Opencl accelerated caffe for convolutional neural networks. In: Proceedings—2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016, pp 50–57. [www.scopus.com](http://www.scopus.com)
4. Bottou L, Cortes C, Denker JS, Drucker H, Guyon I, Jackel LD, LeCun Y, Muller UA, Sackinger E, Simard P et al (1994) Comparison of classifier methods: a case study in handwritten digit recognition. In: Pattern Recognition, 1994. Vol 2-Conference B: Computer Vision & Image Processing. Proceedings of the 12th IAPR International. Conference on, vol 2. IEEE, pp. 77–82
5. Cifar10 (2017). <https://www.cs.toronto.edu/~kriz/cifar.html>

6. Coates A, Ng A, Lee H (2011) An analysis of single-layer networks in unsupervised feature learning. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp 215–223
7. Dahl GE, Sainath TN, Hinton GE (2013) Improving deep neural networks for lvcsr using rectified linear units and dropout. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp 8609–8613
8. Deng L, Liu Y (2018) Deep learning in natural language processing. Springer, Berlin
9. Docker (2019). <https://www.docker.com/>
10. Gold S, Rangarajan A et al (1996) Softmax to softassign: neural network algorithms for combinatorial optimization. *J Artif Neural Netw* 2(4):381–399
11. Gropp W, Lusk E, Doss N, Skjellum A (1996) A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Comput* 22(6):789–828
12. Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface, vol 1. MIT Press, Cambridge
13. Gropp W, Lusk E, Thakur R (1999) Using MPI-2: advanced features of the message-passing interface. MIT Press, Cambridge
14. Grupp A, Kozlov V, Campos I, David M, Gomes J, García Á L (2019) Benchmarking deep learning infrastructures by means of tensorflow and containers. In: International Conference on High Performance Computing. Springer, pp 478–489
15. Hacker SK (2018) Mastering docker: a quick-start beginner's guide. CreateSpace Independent Publishing Platform. <https://dl.acm.org/doi/book/10.5555/3235203>
16. Han J, Zhang D, Cheng G, Liu N, Xu D (2018) Advanced deep-learning techniques for salient and category-specific object detection: a survey. *IEEE Signal Process Mag* 35(1):84–100
17. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp 1135–1143
18. Hegde G, Ramasamy N, Kapre N et al (2016) Caffepresso: an optimized library for deep learning on embedded accelerator-based platforms. In: 2016 International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES). IEEE, pp 1–10
19. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. In: MM 2014—Proceedings of the 2014 ACM Conference on Multimedia, pp 675–678. [www.scopus.com](http://www.scopus.com)
20. Kim Y (2014) Convolutional neural networks for sentence classification. arXiv preprint [arXiv :1408.5882](https://arxiv.org/abs/1408.5882)
21. Kristiani E, Yang CT, Wang YT, Huang CY, Ko PC (2018) Container-based virtualization for real-time data streaming processing on the edge computing architecture. In: International Wireless Internet Conference. Springer, pp 203–211
22. Krizhevsky A, Hinton G (2010) Convolutional deep belief networks on cifar-10. Unpublished manuscript 40
23. Kurth T, Smorkalov M, Mendygral P, Sridharan S, Mathuriya A (2018) Tensorflow at scale: performance and productivity analysis of distributed training with horovod, msls, and cray pe ml. *Concurrency and Computation: Practice and Experience*, p e4989
24. Liu W, Wang Z, Liu X, Zeng N, Liu Y, Alsaadi FE (2017) A survey of deep neural network architectures and their applications. *Neurocomputing* 234:11–26
25. Liu L, Ouyang W, Wang X, Fieguth P, Chen J, Liu X, Pietikäinen M (2020) Deep learning for generic object detection: a survey. *Int J Comput Vis* 128(2):261–318
26. Liu L, Wu Y, Wei W, Cao W, Sahin S, Zhang Q (2018) Benchmarking deep learning frameworks: design considerations, metrics and beyond. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, pp 1258–1269
27. Luong NC, Hoang DT, Gong S, Niyato D, Wang P, Liang YC, Kim DI (2019) Applications of deep reinforcement learning in communications and networking: a survey. *IEEE Commun Surv Tutor* 21(4):3133–3174
28. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp 807–814
29. Nassif AB, Shahin I, Attili I, Azzeh M, Shaalan K (2019) Speech recognition using deep neural networks: a systematic review. *IEEE Access* 7:19143–19165
30. Nath R, Tomov S, Dongarra J (2010) Accelerating GPU kernels for dense linear algebra. In: VEC-PAR. Springer, pp 83–92
31. Openmpi (2017). <https://www.open-mpi.org/>



32. Panda DK, Awan AA, Subramoni H (2019) High performance distributed deep learning: a beginner's guide. In: PPOPP, pp 452–454
33. Purushotham S, Meng C, Che Z, Liu Y (2018) Benchmarking deep learning models on large healthcare datasets. *J Biomed Inform* 83:112–134
34. Rosales C (2014) Porting to the intel xeon phi: opportunities and challenges. In: Proceedings—2013 Extreme Scaling Workshop, XSW 2013, pp 1–7. [www.scopus.com](http://www.scopus.com)
35. Roska T, Hamori J, Labos E, Lotz K, Orzó L, Takacs J, Venetianer PL, Vidnyanszky Z, Zarándy Á (1993) The use of cnn models in the subcortical visual pathway. *IEEE Trans Circuits Syst I Fundam Theory Appl* 40(3):182–195
36. Soheil B, Naveen R, Lukas S, et al (2016) Comparative study of deep learning software frameworks. arXiv preprint [arXiv:1511.06435](https://arxiv.org/abs/1511.06435)
37. Tanno R, Yanai K (2016) Caffe2c: a framework for easy implementation of cnn-based mobile applications. In: ACM International Conference Proceeding Series, vol 28–November-2016, pp 159–164. [www.scopus.com](http://www.scopus.com)
38. Tarasov V, Rupprecht L, Skourtis D, Li W, Rangaswami R, Zhao M (2019) Evaluating docker storage performance: from workloads to graph drivers. *Cluster Computing* pp 1–14
39. Tensorflow description (2019). <https://www.tensorflow.org/>
40. Tokic M, Palm G (2011) Value-difference based exploration: adaptive control between epsilon-greedy and softmax. *KI 2011: Advances in Artificial Intelligence*, pp 335–346
41. Venkateswaran S, Sarkar S (2019) Fitness-aware containerization service leveraging machine learning. *IEEE Trans Serv Comput*. <https://doi.org/10.1109/TSC.2019.2898666>
42. Voulodimos A, Doulamis N, Doulamis A, Protopapadakis E (2018) Deep learning for computer vision: a brief review. *Comput Intell Neurosci*. <https://doi.org/10.1155/2018/7068349>
43. Wang H, Zhang L, Han J, Weinan E (2018) Deepmd-kit: a deep learning package for many-body potential energy representation and molecular dynamics. *Comput Phys Commun* 228:178–184
44. Xbyak (2017). <https://github.com/herumi/xbyak>
45. Yang CT, Liu JC, Chan YW, Kristiani E, Kuo CF (2018) On construction of a caffe deep learning framework based on intel xeon phi. In: International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. Springer, pp 96–106
46. Yang CT, Huang CL, Lin CF (2011) Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. *Comput Phys Commun* 182(1):266–269
47. Zarándy Á, Orzó L, Grawes E, Werblin F (1999) CNN-based models for color vision and visual illusions. *IEEE Trans Circuits Syst I Fundam Theory Appl* 46(2):229–238
48. Zhang Z, Geiger J, Pohjalainen J, Mousa AED, Jin W, Schuller B (2018) Deep learning for environmentally robust speech recognition: an overview of recent developments. *ACM TIST* 9(5):1–28
49. Zhao ZQ, Zheng P, Xu St, Wu X (2019) Object detection with deep learning: a review. *IEEE Trans Neural Netw Learn Syst* 30(11):3212–3232

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.