東海大學

資訊工程研究所

碩士論文

指導教授: 劉榮春博士、楊朝棟博士

在 Intel Xeon Phi 上建構 Caffe 深度學習框架之實作

The Implementation of Caffe Deep Learning Framework Using Intel Xeon Phi

研究生: 郭展甫

中華民國一零六年七月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 郭 展 甫 所提之論文

在 Intel Xeon Phi 上實作 Caffe 深度學習框架

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人 許慶賢 簽章

委 員 黃明居

姜尚弦

指 導 教 授 楊朝棟 簽章

指 導 教 授 劉榮春 簽章

中華民國 106 年 6 月 27 日

摘　要

近年來隨著處理器運算能力的提升，大幅增加了許多科學應用的發展，如天氣預測、金融市場分析、醫療技術提升等。其中深度學習正是因此受惠並快速成長的領域，深度學習可以協助電腦理解影像、文字和聲音等抽象資料，而透過神經網路，電腦也可以擁有與人類一樣的觀察與學習能力，甚至更好。在本論文中，我們將使用 Caffe 深度運算框架，應用於 Xeon Phi 中，透過各項最佳化的方式，包括使用向量化 (Vectorization)、OpenMP 多執行緒平行處理、訊息傳遞介面 (MPI) 等，提高深度學習框架的可用性。Intel 於最近推出了第二代的 Xeon Phi，除了將第一代協同處理器 (Coprocessor) 的產品保留之外，也新增了最多可達 72 核心的主處理器，擁有不可忽視的計算能力。本論文利用 Caffe 深度運算框架在各種 Intel Xeon 的機器上進行訓練，並對機器彼此之間的效能做出評比，實驗的測試的項目包括在訓練模型中重複測試的次數之間的準確度比較，和使用最佳化前及最佳化後在不同機器上的的訓練時間比較，以及使用兩個 Xeon Phi 的多節點測試，結果將會列出各項測試結果的比較以及利用 Xeon Phi 可以減少 Caffe 訓練模型需要花費的時間，提供給各研究者一份測量白皮書。

關鍵字: Xeon Phi、Many-core、Caffe、OpenMP、高效能運算

Abstract

In recent years, with the increase in processor computing power, a substantial increase in the development of many scientific applications, such as weather forecast, financial market analysis, medical technology and so on. Deep Learning can help the computer understand the abstract information such as images, text and sound. Through the neural network, the computer can have the same observation and learning ability as human beings, and even better than human. In this paper, we will use the famous deep learning framework: Caffe, implement to Xeon Phi through the optimization, including the use of vectorization, OpenMP parallel processing, message transfer Interface (MPI), etc., To improve the availability of deep learning framework. Intel recently launched the second generation of Xeon Phi, in addition to the first generation of coprocessor (Coprocessor) products retained, but also added up to 72 core of the main processor, with the power can not be ignored. In this paper, we evaluate the performance of the Caffe deep learning framework across a variety of Intel Xeon platform. The experimental including 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, and the use of two Xeon Phi multi-node tests. The results will be listed in the comparison of the each test performance and the use of Xeon Phi can reduce the training time of Caffe, to provide the researchers with a white paper for measurement.

Keywords: Xeon Phi ، OpenMP ، MPI ، Caffe ، High Performance Programming

致謝詞

在東海大學的兩年碩士生活中，研究所的課程提供了我對於自己的研究領域很多的幫助，除此之外，教授們上課時也不吝於分享自己作為資工人的各種經驗談，讓我對於人生規畫有更進一步的藍圖。

能完成這篇論文要感謝的人有太多，首先我要特別感謝我的兩位指導教授：楊朝棟教授和劉榮春教授，楊老師在大學部的專題就帶領我進入高效能運算的領域，並在我研究的過程提供了許多實用的建議及正確的方向；劉老師在我研究的過程不斷地給予鼓勵，讓我能更有信心的努力去完成我的研究。最重要的是，兩位老師教導了我許多為人處世的態度，使我在磨練之中成長，有了這些經歷我才能夠順利完成這篇論文。

感謝特地抽空前來參加的口試委員們：許慶賢教授、姜自強教授以及黃國展教授，在論文口試時提出很多論文的盲點和非常多寶貴的意見，讓我能將論文修改得更加完整。

論文的完成也要感謝實驗室的學長姐、學弟妹以及最重要的同學們的協助，兩年的碩士生活中幫助我解決各種問題，不管是技術上的瓶頸或是生活中的問題，有了大家的陪伴，這兩年過得十分精采難忘。

感謝我的家人，從小到大不管做什麼事情他們都無條件的支持我，讓我能毫無後顧之憂地在自己選擇的道路上奔跑，有了他們才有現在的我。

最後感謝一路上所有幫助過我的人，有了你們的關心及協助，讓我的研究能夠順利完成，衷心感謝。

東海大學資訊工程學系 高效能計算實驗室 郭展甫 一零六年七月

Table of Contents

摘要 **i**

Abstract **ii**

致謝詞 **iii**

Table of Contents **iv**

List of Figures **vi**

List of Tables **vii**

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Thesis Goal and Contributions . . . 2
- 1.3 Thesis Organization . . . 2

2 Background Review and Related Work **4**
- 2.1 Background Review . . . 4
  - 2.1.1 Xeon Phi Processor . . . 4
  - 2.1.2 OpenMP . . . 8
  - 2.1.3 MPI . . . 10
  - 2.1.4 Caffe Deep Learning Framework . . . 11
- 2.2 Related Works . . . 13

3 System Design and Implementation **16**
- 3.1 System Design . . . 16
  - 3.1.1 Caffe . . . 16
  - 3.1.2 System Flow . . . 17
- 3.2 System Implementation . . . 18
  - 3.2.1 Vectorization . . . 18
  - 3.2.2 Parallelism and OpenMP . . . 19

4 Experimental Results **22**
- 4.1 Experimental Environment . . . 22
  - 4.1.1 Experimental Hardware . . . 22

4.1.2 Experimental Software 23
4.2 Experimental Results 25

5 Conclusions and Future Work 28
5.1 Concluding Remark 28
5.2 Future Works 29

References 30

Appendix 34

A Caffe Installation 34

List of Figures

2.1 Intel Xeon Phi x200 Series Product 5
2.2 KNL package archetecture 6
2.3 OpenMP Architecture 9
2.4 Threads process 10
2.5 MPI Architecture 11
2.6 Caffe Architecture 13

3.1 CIFAR-10 Dataset 17

4.1 LeNet Model Trainig Results 25
4.2 BLVC Caffe Execution Time Comparison 26
4.3 Intel Optimized Caffe Execution Time Comparison 26
4.4 Mutilnode execution results 27

List of Tables

2.1 Intel Xeon Phi x200 Series Specification 6

4.1 Hardware Specification 23
4.2 Software Specification 24

Chapter 1

Introduction

In the past few years with the progress of the chip made, CPU computing power of the repeated peak, accompanied by the emergence of accelerating cards, so that the overall computing power and a higher level. Regardless of scientific, medical, climate research and other issues provide excellent help. Accelerator In addition to the well-known GPU, there are choice of Xeon Phi. Xeon Phi is another choice since it is hard to learn CUDA programing language in spite of the generality of GPU [1].

The deep learning framework is the field of rapid growth in artificial intelligence in recent years. However, the earliest concept of neural network can be traced back to the neuron mathematical model proposed by Warren McCulloch and Walter Pitts in 1943, the traditional neural network technology. It is done by randomly assigning weights and using recursive operations to correct weights one by one compared to the input training data, minimizing the overall error rate. At that time the class of neural network technology as a wave, but can not be sustained, because soon encountered a difficult, lack of computing power. Thanks to the rapid development of computer chips, the deep learning has again become a research and application of a wide range of scientific projects.

1.1 Motivation

In the past few years, CPU computing power is increasing. However, with the application of scientific computing more and more, alone to improve the CPU computing power seems to be inadequate. Recently, it is common for many people to use the GPU as a computational accelerator. To use the GPU, one need to understand the CUDA language; however, it is not easy to learn CUDA and it is difficult to reuse algorithms written with CUDA. Today, 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), messaging 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 [2].

1.2 Thesis Goal and Contributions

This work will implement caffe deep learning framework into the Intel Xeon Phi Platform. Through several optimized function such as Vectorization, Parallelism and OpenMP, can improve the performance and reduce the learning time. Caffe Also we compare the performance with other CPU product, such as and Intel E5 processor, etc.

1.3 Thesis Organization

Chapter 2 will describe some background information, including Xeon Phi Processor, OpenMP, MPI, and Caffe Deep Learning Framework. Chapter 3 will introduce our experimental environment and methods, and the overall architecture. Chapter

4 presents and analyses experimental results. Finally, Chapter 5 summarizes this work by pointing out its major contributions and directions for future work.

Chapter 2

Background Review and Related Work

In this section, we review some background knowledges for later use of system design and implementation.

2.1 Background Review

2.1.1 Xeon Phi Processor

On June 20, 2016, Intel introduced the code-named Knights Landing's Intel Xeon Phi product line x200, emphasizing its not only for traditional analog workloads, but also for machine learning. It is worth noting that, x200 series Xeon Phi in addition to the first generation x100, code Knights Corner the same coprocessor processor version, but also itself is a powerful computing power of the Processor version. The previously introduced Xeon Phi co-processor family includes the 3100, 5100 and 7100 series, which are installed on the x86 server as a GPU via a PCIe x16 slot. The process of accessing the main memory of the system requires control via the CPU's memory , but this will affect the overall efficiency. The new x200 series is now a stand-alone Xeon Phi Processor, the difference is that more

than the extension of the finger with the integration of Intel Omni-Path Fabric (Intel OPA) high-speed network. As shown in figure2.1

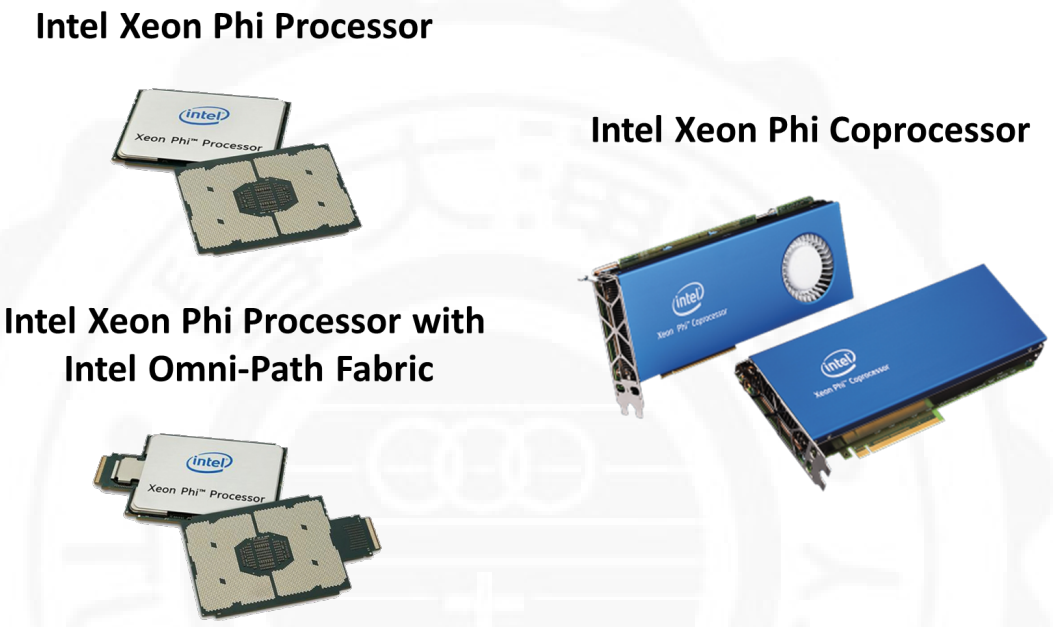

FIGURE 2.1: Intel Xeon Phi x200 Series Product

The Intel Xeon Phi series, based on Many Integrated Core (MIC) architecture, provides high-performance computing capabilities that were never available on the Multi-Core architecture. Intel MIC architecture integrates the core of multiple Intel processors on a single chip, and the use of standard C, C + + and FORTAN program code, the code written for Intel MIC architecture can also use the standard Intel Xeon processing to compile and execute, and provide developers with the well-known programming model used directly on Xeon Phi, eliminating the need to redesign software engineering time and improve the efficiency of solving problems. The new Knights Landing core architecture, which uses more than 60 Silvermont architecture cores, not only achieves 3 TFLOPS computing power on the overall processor performance, but also faster in single-threaded performance than the first generation of Knights Corner architecture three times. Moreover, the processor built-in 16GB memory, bandwidth almost reached DDR4 5 times, and the use of 6-channel memory technology, the maximum support 384GB DDR4 memory capacity. As shown in figure2.2

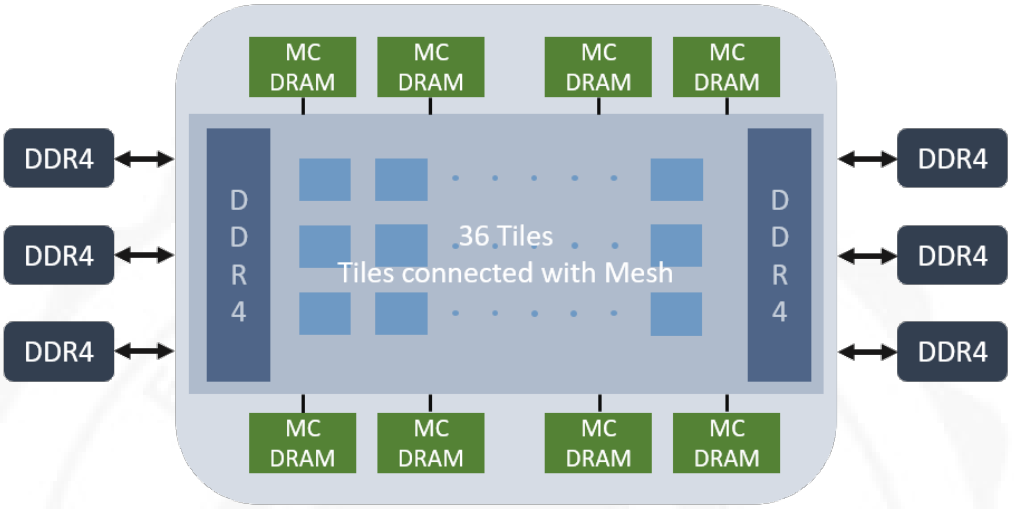

FIGURE 2.2: KNL package archetecture

The new KNL consists of a 14-nanometer process, with more than 8 billion transistors in the 72-core processor, and is divided into up to 36 dual-core distributions in the grid configuration. Each quad-threaded Silvermont core has two AVX-512 VPUs (vector processing units) with a total of 144 VPUs. In addition to each dual-core with 1MB shared L2 cache, equal to a total of up to 36MB L2 cache. The second generation of Intel Xeon Phi contains several types, divided into 7210 series, 7230 series, 7250 series and 7290 series. As shown in 2.1.

TABLE 2.1: Intel Xeon Phi x200 Series Specification

	7210 Series	7230 Series	7250 Series	7290 Series
Cores	64	64	68	72
Processor Freq.	1.30 GHz	1.30 GHz	1.40 GHz	1.50 GHz
L2 Cache	32 MB	32 MB	34 MB	36 MB
Mem. Capacity	384 GB	384 GB	384 GB	384 GB
Mem. Channels	6	6	6	6
Mem. Bandwidth	102 GB/s	115.2 GB/s	115.2 GB/s	115.2 GB/s
TDP	215 W	215 W	215 W	245 W

On the instruction set, the new Xeon Phi supports Intel Advanced Vector Extensions 512 (AVX-512) with 512-bit vector scratchpad, hardware convergence / divergence, DP-superior feature support, and non-Intel compilers such as GCC. Details of the AVX-512 categories are as follows:

- AVX-512F (Fundamentals) extends most of the AVX2 instruction set to a 512-bit vector register.
- AVX-512CD (Conflict Detection) Effective conflict detection, such as Data Bining.
- AVX-512ER (Exponential and Reciprocal) beyond the function support, such as exp, rcp and rsqrt.
- AVX-512PF (Prefetch) pre-fetch for divergence and convergence.

In the KNL memory structure is directly access to MCDRAM (Multi-Channel DRAM) and the system of DDR4 memory. Which MCDRAM has three kinds of memory mode, respectively, Flat Mode, Cache Mode and Hybrid Mode.

- Flat Mode: In this mode, MCDRAM is treated as a NUMA node, and the user can control what is going to enter MCDRAM.
- Cache Mode: In Cache Mode MCDRAM is considered Last Level Cache (LLC) and automatically uses MCDRAM.
- Hybrid Mode: Combines the above two modes, the assigned ratio can be selected in the BIOS.

Which in the cluster mode there are several different ways to control, the demand is from the core memory, a detailed description of the following:

- All-to-All: In this mode, memory access can be from any core to any memory channel while supporting user optimization MPI and OpenMP or Intel Threading Building Blocks.
- Quadrant: Quadrant mode can be selected if the user wants to achieve a higher performance than All-to-All without having to change the code. The benefit of this mode is that as long as the symmetric configuration of DIMMs is used, the Quadrant is selected as the default mode at the beginning of the KNL system.

- SNC (Sub NUMA Clustering): There are three parts that support the memory access requirements, include the core, the tag directory and the memory channel. In order to achieve consistent performance optimization in this mode, the need for code changes or the control of the NUMA environment. This mode is suitable for decentralized memory models such as MPI or mixed MPI-OpenMP.

2.1.2 OpenMP

OpenMP [3] [4] [5] [6] is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer. An application built with the hybrid model of parallel programming can run on a computer cluster using both OpenMP and Message Passing Interface (MPI), such that OpenMP is used for parallelism within a (multi-core) node while MPI is used for parallelism between nodes. There have also been efforts to run OpenMP on software distributed shared memory systems, to translate OpenMP into MPI and to extend OpenMP for non-shared memory systems. The architecture of OpenMP is shown as figure2.3.

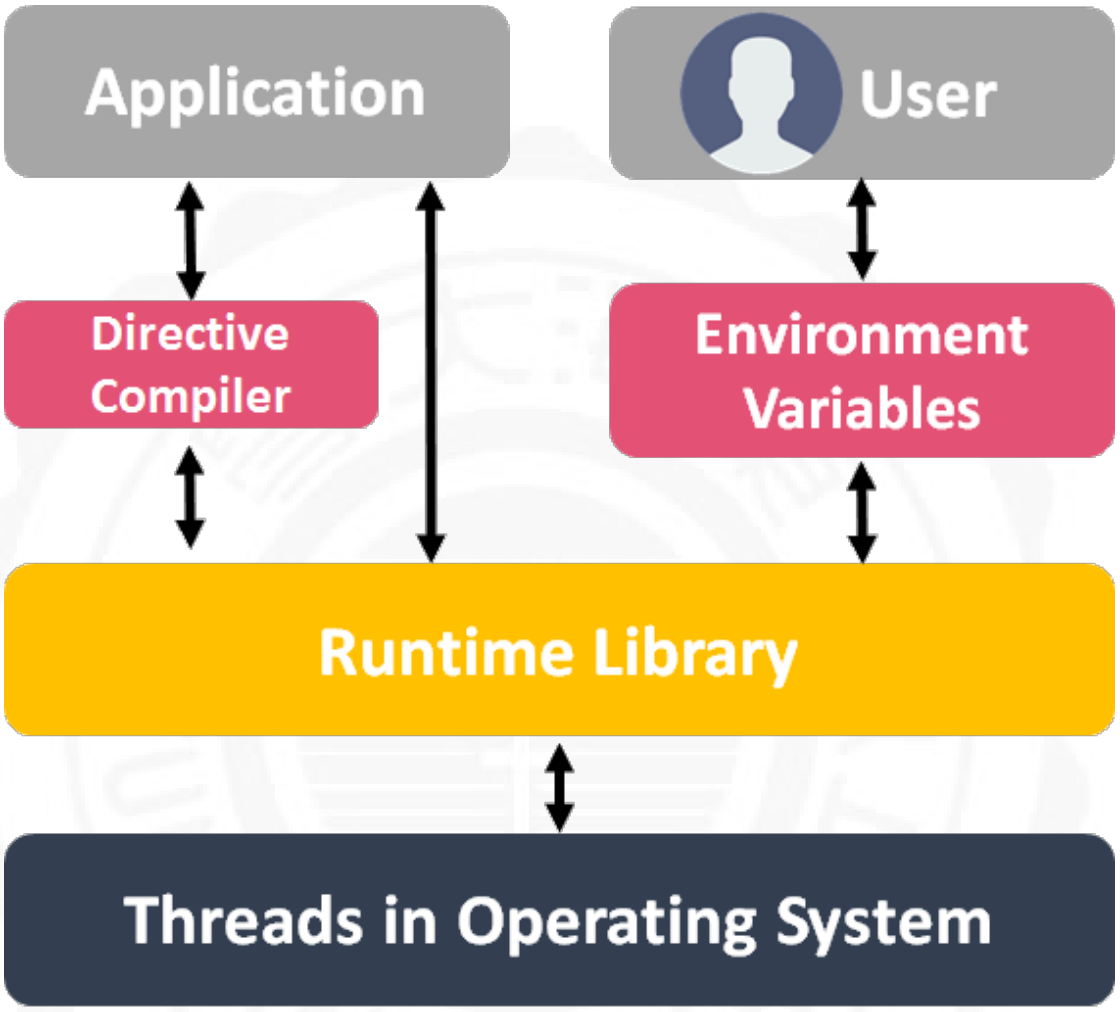

FIGURE 2.3: OpenMP Architecture

OpenMP is an implementation of multithreading, 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. The threads then run concurrently, with the runtime environment allocating threads to different processors. As shows on figure2.4, OpenMP is an implementation of multithreading, 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. 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.

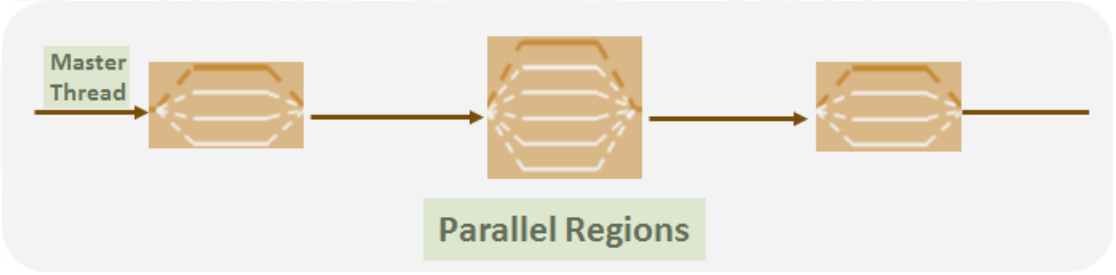

FIGURE 2.4: Threads process

2.1.3 MPI

Message Passing Interface (MPI) [7] [8] [9] [10] 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 figure2.5.

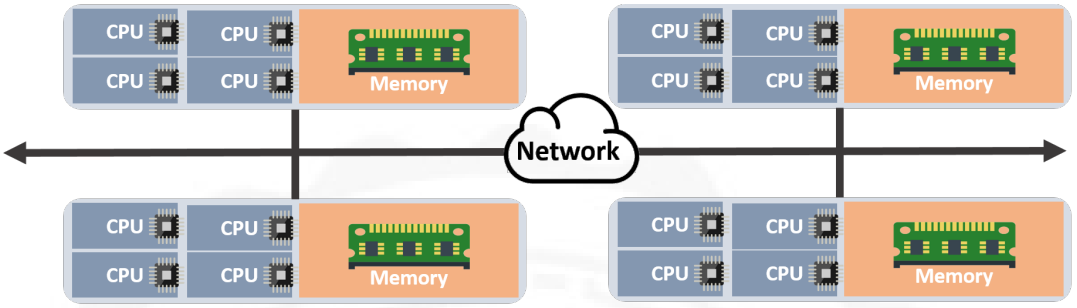

FIGURE 2.5: MPI Architecture

MPI is a communication protocol for programming parallel computers. Both point-to-point and collective communication are supported. MPI is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today. Most messaging interfaces are implemented as libraries and do not require compiler support.

2.1.4 Caffe Deep Learning Framework

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license. Caffe support C++ / CUDA, command line, Python, MATLAB interfaces. Also Caffe has features below:

- Expression: models and optimizations are defined as plaintext schemas instead of code.
- Speed: for research and industry alike speed is crucial for state-of-the-art models and massive data.
- Modularity: new tasks and settings require flexibility and extension.

- Openness: scientific and applied progress call for common code, reference models, and reproducibility.
- Community: academic research, startup prototypes, and industrial applications all share strength by joint discussion and development in a BSD-2 project.

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.

In addition to the computational functions of Layers, there are some special data layers that can be read from the file, or write the output results to a specific file. Moreover, there are some loss layer is used to calculate the final results of the score, and this information is used to optimize all the parameters in Solver. Each layer will create additional blobs to place these trained parameters, and Net will

collect these blobs when the layer is built, making it easy for Solver to calculate the updated value for each parameter based on the learning rate. When Solver calls Net Forward and Backward, the data is calculated along a layer of layer. The hole architecture of the process is shown in figure2.6

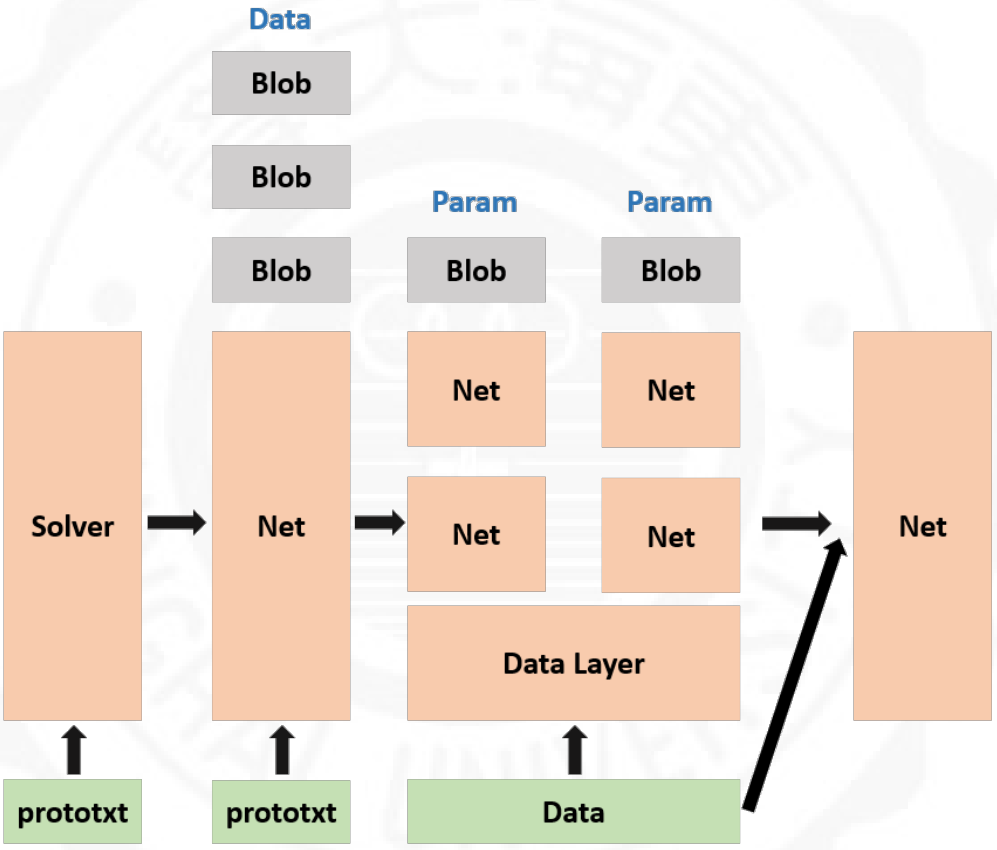

FIGURE 2.6: Caffe Architecture

2.2 Related Works

Zhang,C et .al [11] design and implement Caffeine, a hardware/software co-designed library to efficiently accelerate the entire CNN on FPGAs.Their Caffeine achieves a peak performance of 365 GOPS on Xilinx KU060 FPGA and 636 GOPS on Virtex7 690t FPGA. This is the best published result to their best knowledge. They achieve more than 100x speedup on FCN layers over previous FPGA accelerators. An end-to-end evaluation with Caffe integration shows up to 7.3x and

43.5x performance and energy gains over Caffe on a 12-core Xeon server, and 1.5x better energy-efficiency over the GPU implementation on a medium-sized FPGA (KU060). Performance projections to a system with a high-end FPGA (Virtex7 690t) shows even higher gains.

Hegde,G. et .al [12] present CaffePresso, a Caffe-compatible framework for generating optimized mappings of user-supplied ConvNet specifications to target various accelerators such as FPGAs, DSPs, GPUs, RISC-multicores. They use an automated code generation and autotuning approach based on knowledge of the ConvNet requirements, as well as platform-specific constraints such as on-chip memory capacity, bandwidth and ALU potential. While one may expect the Jetson TX1 + cuDNN to deliver high performance for ConvNet configurations, they observe a flipped result with slower GPU processing compared to most other systems for smaller embeddedfriendly datasets such as MNIST and CIFAR10, and faster and more energy efficient implementation on the older 28nm TI Keystone II DSP over the newer 20nm NVIDIA TX1 SoC in all cases.

Tanno,R et .al [13] create "Caffe2C" which converts CNN (Con-volutional 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 embedding 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 pa-per, they demonstrate the availability of Caffe2C by showing four kinds of CNN-base object recognition mobile applications.

Jia,Y et .al [14] separating model representation from actual implementation, Caffe allows experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments.

Caffe is maintained and developed by the Berkeley Vision and Learning Center (BVLC) with the help of an active community of contributors on GitHub. It powers ongoing research projects, large-scale industrial applications, and startup prototypes in vision, speech, and multimedia.

Bottleson,J et .al [15] present 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. In their work, clCaffe, greatly enhances the ability to leverage deep learning use cases on all types of OpenCL devices, particularly on small form factor devices in which discrete GPUs are rare and integrated GPUs are much more common. Our benchmark shows 2.5x speedup on the Intel integrated-GPU, compared to CPU-only AlexNet on ImageNet dataset. As such, their work provides the deep learning community with the opportunity to embrace a broad range of devices through OpenCL.

Chapter 3

System Design and Implementation

In the section, we will introduce the system design and implementation. Section 3.1 describes the system design. Section 3.2 describes the System implementation.

3.1 System Design

3.1.1 Caffe

Caffe architecture, we use the CIFAR-10 [16] [17] [18] full sigmoid model, CNN model [19] [20] [21] includes convolution, the largest pool, batch normalization, full connection, multi-layer and softmax layer. The CIFAR-10 dataset, shown as Figure3.1, consists of 60000 color images, each with 32 × 32, equally divided and marked as "to the following 10 categories of sizes: aircraft, car, bird, catalog, deer, dog, frog, horse, (Such as sedans or sports utility tools) or defeated all trucks (which contain only large trucks) without overlapping, none of the groups included to charge to defeat all the trucks.

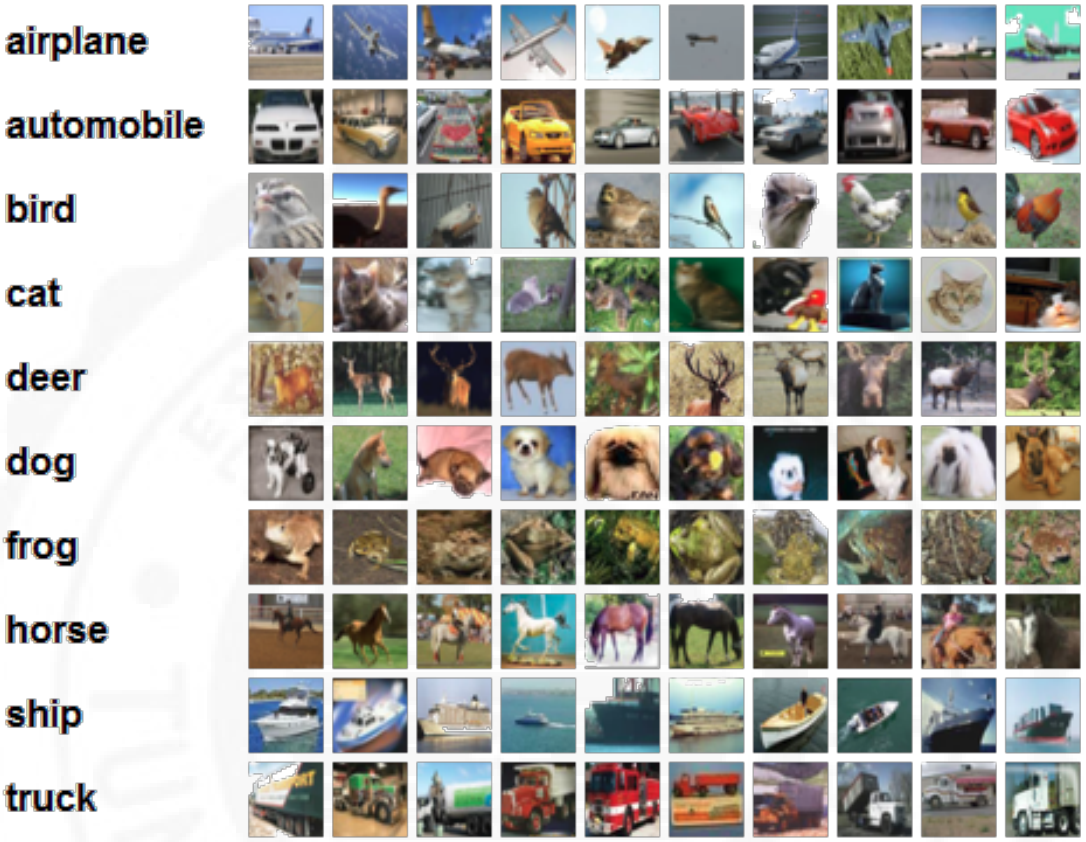

FIGURE 3.1: CIFAR-10 Dataset

3.1.2 System Flow

Caffe, optimized for the Intel architecture, now incorporates the latest version of the Intel Math Core Library (Intel MKL) 2017 Optimized Advanced Vector Extensions (AVX) -2 and the avx-512 instruction to support Intel Xeon with the Intel Xeon Phi processor (and others). That is, Caffe, which is optimized for the Intel architecture, contains all the advantages found in BVLC Caffe not only that, but also efficiently, on Intel architectures and training courses that can be used for various nodes. The system flow for our design is

- Install Caffe on Xeon Phi Processor
- Train and test on LeNet MNIST [22] [23]
- Test pre-trained models such as bvlc_googlenet.caffemodel, certain images, such as catalogs and fish-locomotives

- Fine-tune the Cats vs Dog Challenge the trained model

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, we apply the vectorization optimization. These optimizations include the following:

- Basic Linear Algebra Complex (BLAS) [24] Library (Intel MKL to Switch from Auto-Adjust Linear Algebra System [ATLAS] [25])
- Optimized components (Xbyak just-in-time [JIT] [26] 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, we 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 x200 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. We use the Xbyak vector for this job, which can improve the performance of the large shared layer of the program. If we all know the parameters of the cluster, we can generate the code of the component 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

The following of neural networks layers are optimized by using Parallel processing of OpenMP threads

Convolution layers

Convolution layers, as the name suggests, convolves learn to weight or filter, with the program input each generating a function graph in the output image. This optimization, which prevents the infrequent input function from being used to a group of hardware.

Shared or Subsampling

The largest pool, the average area, and the stochastic area are different methods that can downsampling 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 × 2 partition (a typical partition of the area) can convert it to a single pixel, from three will return the same maximum. 3 × 3 windows, the five will not return the same maximum value.

Pooling a single function on the map of the job mode, so we used Xbyak 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 [27] [28] (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:

$$P(y = j \mid x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \tag{3.1}$$

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)

ReLUs [29] [30] are currently using the most popular non-linear features of the depth learning algorithm. Enables the element-wise operator of the neuron layer to put a lower point block and produce a top dot of the same size. (Point-for-architecture integrated memory interface with standard array. With information on products and derivatives through the Internet, Caffe storage, communication, and management information to use.) The ReLU layer needs to enter the value x x positive values to calculate the output and extend them to negative_slope negative values:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \textbf{negativeslope} * x, & \text{otherwise} \end{cases} \tag{3.2}$$

Chapter 4

Experimental Results

In this section, we will introduce the experiment. Section 4.1 describes the experimental environment, including experimental hardware, experimental software and experimental design.

4.1 Experimental Environment

In our experimental environment, including one Xeon E5 and one Xeon Phi 7210 spect processor. In multinode distributed training, we use two Xeon E5 and two Xeon Phi 7210 spect processor.

4.1.1 Experimental Hardware

We list our hardware detail as shown in the table 4.1.

TABLE 4.1: 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 6.6	CentOS 7.2
Linux Kernel	2.6.32-504.el6.x86_64	3.10.0-327.el7.x86_64

4.1.2 Experimental Software

We list the software version used in the experiment, and describe its function. As shown in the table 4.2.

TABLE 4.2: Software Specification

Name	Version	Description
Intel Parallel Studio XE	2017 update 3	Includes compilers, performance libraries, and parallel models optimized to build fats 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 Inspector XE is an easy to use memory and threading error debugger for C, C++, C# and Fortran application that run
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.

4.2 Experimental Results

We train the LeNet, which is the MNIST Classification Model with Caffe. We start the experiment by the following main step: preparing the dataset, training a model, and timing the model. First we download MNIST dataset and create dataset in LMDB format. Next at training the dataset, we reduce the number of iterations from 10K to 1K to quickly run. Then we timing the forward and backward propagations. Finally we test the trained model in the validation test. The results shown in Figure4.1.

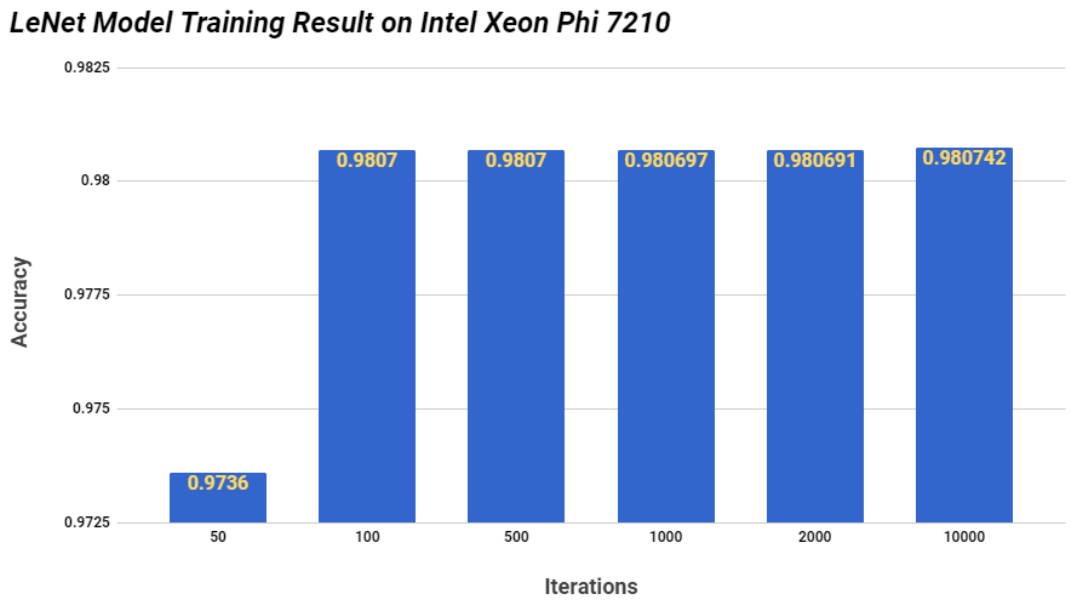

FIGURE 4.1: LeNet Model Trainig Results

Next we use the time command to benchmarking BLVC Caffe and Intel optimized Caffe in two platform, Intel Xeon E5-2560 and Intel Xeon Phi 7210. The time command will compute the layer-by-layer forward and backward propagation time. It measure the time which spent in each layer and for providing the relative execution times for different model. The results shown in Figure4.2 and Figure4.3.

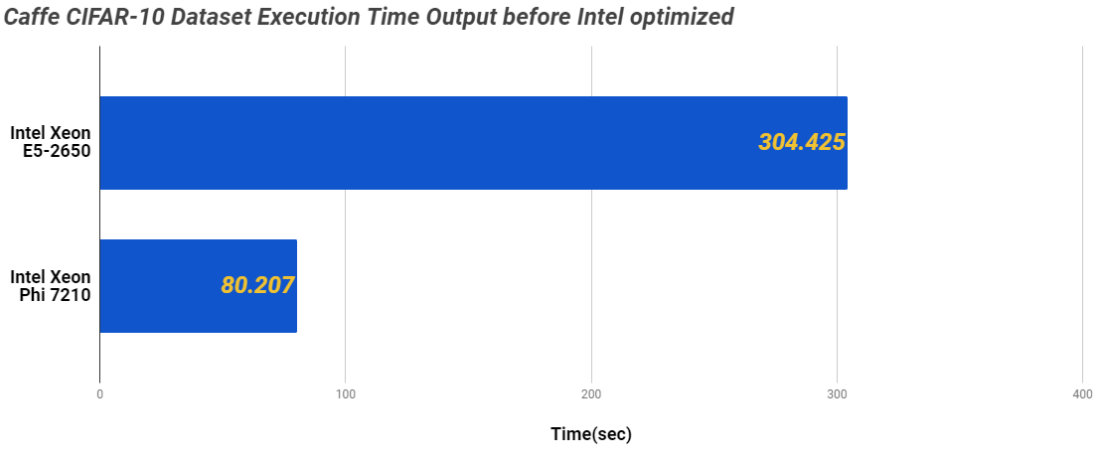

FIGURE 4.2: BLVC Caffe Execution Time Comparison

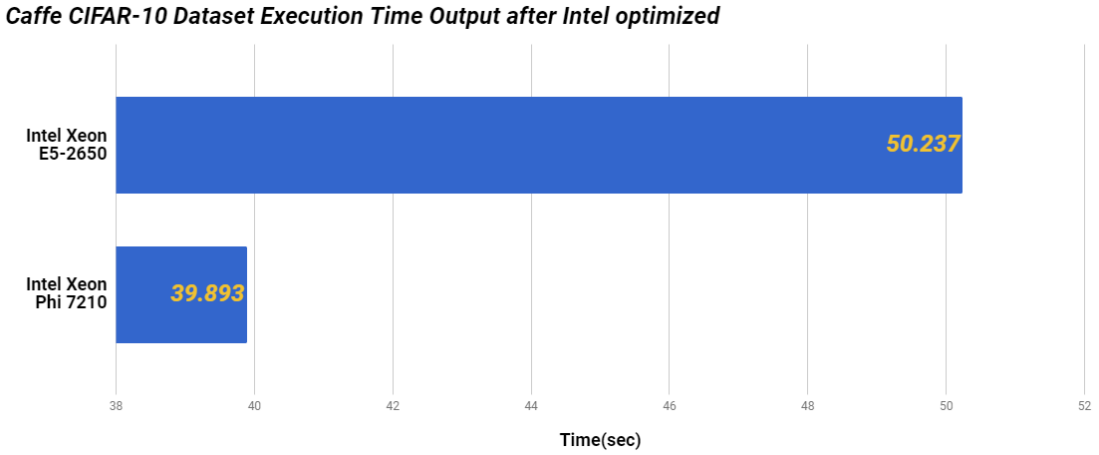

FIGURE 4.3: Intel Optimized Caffe Execution Time Comparison

Also we use multinodes distributed training on two Intel Xeon Phi 7210. There are two main approaches to distribute the training across multiple nodes: model parallelism and data parallelism. In model parallelism, the model is divided among the nodes and each node has the full data batch. In data parallelism, the data batch is divided among the nodes and each node has the full model. Data parallelism is especially useful when the number of weights in a model is small and when the data batch is large. A hybrid model and data parallelism is possible where layers with few weights such as the convolutional layers are trained using the data parallelism approach and layers with many weights such as fully connected

layers are trained using the model parallelism approach. The training results shows as Figure4.4

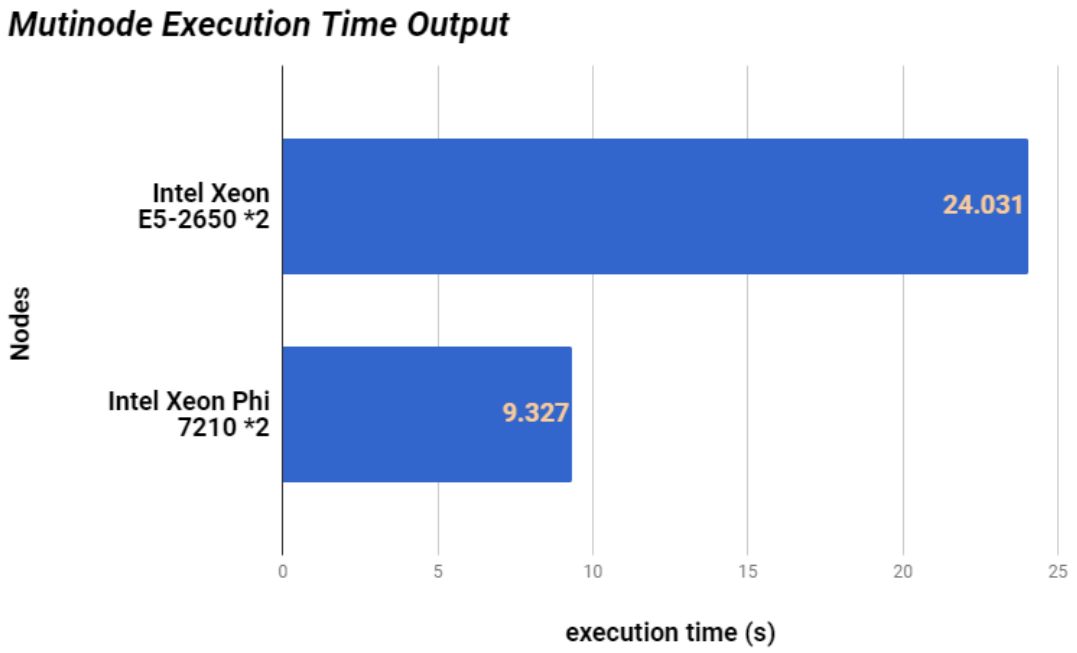

FIGURE 4.4: Mutilnode execution results

Chapter 5

Conclusions and Future Work

5.1 Concluding Remark

This work has optimized Caffe, the deep learning framework implement on Intel Xeon Phi Processor. By using optimization method such as vectorization and parallelism OpenMP, the training time output can reduce 6.059 times at Intel Xeon E5-2650, 2.010 times at Intel Xeon Phi 7210. We successfully optimized the code of Caffe also reduce the training consume, make this deep learning framework become more useful on training model. We can understand that the performance without optimized is pretty poor at Intel Xeon E5-2650, but for Intel Xeon Phi 7210 it can get 3.795 times reduce. However, after optimization, Intel Xeon E5-2650 can reduce 3 times training time than Intel Xeon Phi 7210. Although its performance still not better than Intel Xeon Phi 7210. Also the LeNet Model training results shows that on Intel Xeon Phi 7210 can get high accuracy at 0.980742. Moreover, we use multinode such as two Intel Xeon Phi 7210, to get even better performance on training model as the 9.327 second.

5.2 Future Works

Our evaluation only done by the environment of Intel Xeon Phi product. However, we hope in the future GPU testing can be perform. Due to GPU extraordinary computing capability, we are happy to see the competitive between two HPC devies. Moreover, we would like to try more nodes on multinode distributed distributed training. In addition, we could compare with three kinds of platform: Multinode CPU/GPU, Intel Xeon Phi, GPU and to find which gets better performance on Caffe deep learning framework.

References

[1] A. Heinecke. Accelerators in scientific computing is it worth the effort? In *2013 International Conference on High Performance Computing Simulation (HPCS)*, pages 504–504, July 2013.

[2] C. Rosales. Porting to the intel xeon phi: Opportunities and challenges. In *Proceedings - 2013 Extreme Scaling Workshop, XSW 2013*, pages 1–7, 2014.

[3] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[4] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. MIT press, 2008.

[5] Rohit Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[6] Openmp, 2017. `https://www.openmp.org`.

[7] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[8] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[9] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced features of the message-passing interface*. MIT press, 1999.

[10] Openmpi, 2017. `https://www.open-mpi.org/`.

[11] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, volume 07-10-November-2016, 2016.

[12] G. Hegde, Siddhartha, N. Ramasamy, and N. Kapre. Caffepresso: An optimized library for deep learning on embedded accelerator-based platforms. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES 2016*, 2016.

[13] R. Tanno and K. Yanai. Caffe2c: A framework for easy implementation of cnn-based mobile applications. In *ACM International Conference Proceeding Series*, volume 28-November-2016, pages 159–164, 2016.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, pages 675–678, 2014.

[15] J. Bottleson, S. Kim, J. Andrews, P. Bindu, D. N. Murthy, and J. Jin. Clcaffe: Opencl accelerated caffe for convolutional neural networks. In *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pages 50–57, 2016.

[16] Alex Krizhevsky and G Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40, 2010.

[17] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[18] Cifar10, 2017. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[19] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[20] Tamas Roska, Jozef Hamori, Elemer Labos, Karloy Lotz, László Orzó, Jozsef Takacs, Peter L Venetianer, Zoltan Vidnyanszky, and Ákos Zarándy. The use of cnn models in the subcortical visual pathway. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(3):182–195, 1993.

[21] Ákos Zarándy, László Orzó, Edward Grawes, and Frank Werblin. Cnn-based models for color vision and visual illusions. *IEEE transactions on circuits and systems I: Fundamental theory and applications*, 46(2):229–238, 1999.

[22] Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Lawrence D Jackel, Yann LeCun, Urs A Muller, Edward Sackinger, Patrice Simard, et al. 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*, volume 2, pages 77–82. IEEE, 1994.

[23] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[24] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[25] Rajib Nath, Stanimire Tomov, and Jack Dongarra. Accelerating gpu kernels for dense linear algebra. In *VECPAR*, pages 83–92. Springer, 2010.

[26] Xbyak, 2017. `https://github.com/herumi/xbyak`.

[27] Steven Gold, Anand Rangarajan, et al. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, 2(4):381–399, 1996.

[28] Michel Tokic and Günther Palm. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. *KI 2011: Advances in Artificial Intelligence*, pages 335–346, 2011.

[29] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[30] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

Appendix A

Caffe Installation

I. Basic Installation

```
$ sudo apt-get update &&
$ sudo apt-get -y install build-essential git cmake &&
$ sudo apt-get -y install libprotobuf-dev libleveldb-dev libsnappy-dev &&
$ sudo apt-get -y install libopencv-dev libhdf5-serial-dev protobuf-compiler &&
$ sudo apt-get -y install --no-install-recommends libboost-all-dev &&
$ sudo apt-get -y install libgflags-dev libgoogle-glog-dev liblmdb-dev &&
$ sudo apt-get -y install libatlas-base-dev
```

II. Ubuntu libraries

```
$ find . -type f -exec sed -i -e 's^"hdf5.h"^"hdf5/serial/hdf5.h"^g' -e
's^"hdf5_hl.h"^"hdf5/serial/hdf5_hl.h"^g' '{}' ;
$ cd /usr/lib/x86_64-linux-gnu
$ sudo ln -s libhdf5_serial.so.10.1.0 libhdf5.so
$ sudo ln -s libhdf5_serial_hl.so.10.0.2 libhdf5_hl.so
```

III. CentOS 7 install

```
$ sudo yum -y update &&
$ sudo yum -y groupinstall "Development Tools" &&
$ sudo yum -y install wget cmake git &&
$ sudo yum -y install protobuf-devel protobuf-compiler boost-devel &&
$ sudo yum -y install snappy-devel opencv-devel atlas-devel &&
$ sudo yum -y install gflags-devel glog-devel lmdb-devel leveldb-devel hdf5-devel
```

```
# The following steps are only required if some packages failed to install
# add EPEL repository then install missing packages
$ wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo rpm -ivh epel-release-latest-7.noarch.rpm
$ sudo yum -y install gflags-devel glog-devel lmdb-devel leveldb-devel hdf5-devel &&
$ sudo yum -y install protobuf-devel protobuf-compiler boost-devel

# if packages are still not found--download and install/build the packages, e.g.,
# snappy:
$ wget http://mirror.centos.org/centos/7/os/x86_64/Packages/snappy-
devel-1.1.0-3.el7.x86_64.rpm
$ sudo yum -y install http://mirror.centos.org/centos/7/os/x86_64/Packages/snappy-
devel-1.1.0-3.el7.x86_64.rpm
# atlas:
$ wget http://mirror.centos.org/centos/7/os/x86_64/Packages/atlas-
devel-3.10.1-10.el7.x86_64.rpm
$ sudo yum -y install http://mirror.centos.org/centos/7/os/x86_64/Packages/atlas-
devel-3.10.1-10.el7.x86_64.rpm
# opencv:
$ wget https://github.com/Itseez/opencv/archive/2.4.13.zip
$ unzip 2.4.13.zip
$ cd opencv-2.4.13/
$ mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr/local ..
$ NUM_THREADS=$(($(grep 'core id' /proc/cpuinfo | sort -u | wc -l)*2))
$ make all -j $NUM_THREADS
$ sudo make install -j $NUM_THREADS


# Once installed, the correct environment libraries can be set as follows
(the path may need
to be modified)
$ echo 'source /opt/intel/bin/compilervars.sh intel64' >> ~/.bashrc
# alternatively edit <mkl_path>/mkl/bin/mklvars.sh replacing INSTALLDIR in
# CPRO_PATH=<INSTALLDIR> with the actual mkl path: CPRO_PATH=<full mkl path>
# echo 'source <mkl path>/mkl/bin/mklvars.sh intel64' >> ~/.bashrc

# Clone and prepare Caffe optimized for Intel architecture for compiling as follows
$ cd ~
# For BVLC caffe use:
# git clone https://github.com/BVLC/caffe.git
# For intel caffe use:
$ git clone https://github.com/intel/caffe.git
$ cd caffe
$ echo "export CAFFE_ROOT=`pwd`" >> ~/.bashrc
$ source ~/.bashrc
$ cp Makefile.config.example Makefile.config
```

```
# Open Makefile.config and modify it (see comments in the Makefile)
$ vi Makefile.config

# Edit the Makefile.config
# To run on CPU only and to avoid installing CUDA installers, uncomment
 CPU_ONLY := 1

# To use MKL, replace atlas with mkl as follows
# (make sure that the BLAS_DIR and BLAS_LIB paths are correct)
$ BLAS := mkl
$ BLAS_DIR := $(MKLROOT)/include
$ BLAS_LIB := $(MKLROOT)/lib/intel64

# To use MKL2017 DNN primitives as the default engine, uncomment
# (however leave it commented if using multinode training)
# USE_MKL2017_AS_DEFAULT_ENGINE := 1

# To customized compiler choice, uncomment and set the following
# CUSTOM_CXX := g++

# To train on multinode uncomment and verify path
# USE_MPI := 1
# CXX := /usr/bin/mpicxx
```

IV. Build Caffe optimized for Intel architecture

```
# If using Ubuntu 16.04, edit the Makefile
$ INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include /usr/include/hdf5/serial/

# and create symlinks
$ cd /usr/lib/x86_64-linux-gnu
$ sudo ln -s libhdf5_serial.so.10.1.0 libhdf5.so
$ sudo ln -s libhdf5_serial_hl.so.10.0.2 libhdf5_hl.so

# If using CentOS 7 and ATLAS (instead of the recommended MKL library),
edit the Makefile
# Change this line
$ LIBRARIES += cblas atlas
# to
$ LIBRARIES += satlas

# Build Caffe optimized for Intel architecture
$ NUM_THREADS=$(($(grep 'core id' /proc/cpuinfo | sort -u | wc -l)*2))
$ make -j $NUM_THREADS
# To save the output stream to file makestdout.log use this instead
# make -j $NUM_THREADS 2>&1 | tee makestdout.log
```

V. LeNet on MNIST

```
# Preparing datasets
$ cd $CAFFE_ROOT
$ ./data/mnist/get_mnist.sh # downloads MNIST dataset
$ ./examples/mnist/create_mnist.sh # creates dataset in LMDB format

# Training datasets
# Reduce the number of iterations from 10K to 1K to quickly run through this example
$ sed -i 's/max_iter: 10000/max_iter: 1000/g' examples/mnist/lenet_solver.prototxt
$ ./build/tools/caffe train -solver examples/mnist/lenet_solver.prototxt

# Timing the forward and backward propagations (not including weight updates)
$ ./build/tools/caffe time --model=examples/mnist/lenet_train_test.prototxt
-iterations 50
# runs on CPU

# For consistency in the timings, the utility numactl can be used to allocate memory
buffers in MCDRAM
$ numactl -i all /path/to/caffe/build/tools/caffe time --model=train_val.prototxt -
iterations $NUMITER

# Testing the trained model
# the file with the model should have a 'phase: TEST'
$ ./build/tools/caffe test -model examples/mnist/lenet_train_test.prototxt
  -weights examples/mnist/lenet_iter_1000.caffemodel -iterations 50
```

VI. Caffe Execution

```
# Initial Performance Profiling
$ ./build/tools/caffe time \
    --model=examples/cifar10/cifar10_full_sigmoid_train_test_bn.prototxt \
    -iterations 1000
```