Performance and Energy Optimisation of Artificial Neural Networks on Hybrid Heterogeneous Platforms

PhD Transfer Report

Atefeh Khazaei Ghoozhdi
atefeh.khazaei@ucdconnect.ie

Student Number
20202243

PhD Start Date
01/09/2020

Supervisor
Dr. Alexey Lastovetsky

University College Dublin

March 16, 2022

Contents

Abstract

Nowadays, ANNs are behind many great achievements, such as automatic image recognition, conversing, interpreting textual documents, and driving vehicles. Producing these great achievements needs lots of time, computing resources, and electricity.

At the same time, the modern high-performance computing platforms have become highly heterogeneous due to the tight integration of multicore CPU processors and accelerators (such as GPUs, Intel Xeon Phis, or FPGAs) to maximize the dominant objectives of performance and energy efficiency.

In this research, we are going to use modern heterogeneous servers to reduce the concerns about the growth of ANN applications and their performance and energy consumption. Optimal resources usage on heterogeneous platforms can significantly increase performance and reduce energy consumption.

This research objective is to minimize the execution time and the energy consumption of different types of ANNs (Fully Connected, CNNs, and RNNs) to execute on hybrid heterogeneous platforms. Our approach to achieve this goal is through the distribution of the workload between heterogeneous devices.

To obtain this goal, we implement different combinations of ANNs' types (Fully Connected, CNNs, and RNNs), using different parallelism methods (Operators and Networks parallelization), and present our ANNs' applications. Then, we develop model-based workload partitioning algorithms to minimize the performance and energy consumption of ANN applications on heterogeneous systems.

1 Introduction

Artificial Neural Networks (ANNs) routinely produces great achievements, as computers learn to recognize images, converse, interpret the textual data, beat humans at sophisticated games, and drive vehicles [1].

ANNs, computing systems inspired by biological neural networks, have three important types:

- **Fully Connected Neural Networks:** In short, we call them Fully Connected (FC) in this report. In this type of artificial neural network, all the nodes (neurons) in one layer are connected to the neurons in the next layer. A fully connected layer can be expressed and modeled as a matrix multiplication of the weights and the neuron values [1].
- **Convolution Neural Networks** This type of ANNs, also called CNN or ConvNet, is most commonly applied to analyze visual imagery. The layers of a CNN consist of an input layer, an output layer, and a hidden layer. The hidden layer includes multiple convolutional layers, pooling layers, fully connected layers, and normalization layers [1].
- **Recurrent Neural Networks:** In this type of ANNs, also called RNN, the connections between nodes form a directed or undirected

graph along a temporal sequence. This feature leads to exhibit temporal dynamic behavior, so RNNs can use their internal state (memory) to process variable length sequences of inputs. This type of ANNs is commonly used for ordinal or temporal problems, such as language translation, natural language processing (NLP), speech recognition, and image captioning [1].

These three different types of neural networks, in different research and industry fields, have achieved amazing achievements, but all those advances require huge amounts of computing power and electricity to invent and train algorithms. The damage caused by climate change becomes more apparent and AI experts are increasingly bothered by those energy demands [2].

Over the last few years, the modern HPC platforms have become highly heterogeneous owing to the tight integration of multicore CPU processors and accelerators (such as GPUs, Intel Xeon Phis, or FPGAs) [3]. These heterogeneous platforms can be one of the best options to reduce the concerns about the growth of ANN applications. Optimal resources usage on heterogeneous platforms can significantly increase performance and reduce energy consumption.

Our research objective is to minimize the execution time and the energy consumption of different types of ANNs (Fully Connected, CNNs, and RNNs) on hybrid heterogeneous platforms. Our approach to achieve this goal is through the optimal distribution of the workload between heterogeneous devices.

The rest of this report is organized as follows: Section 2 presents related works in 'ANN parallelization', 'performance optimisation', and 'energy optimisation' on heterogeneous platforms. In Section 3 our research questions and hypotheses are introduced. Our proposed approach is addressed in Section 4. The done steps of this research will be explained in Section 5. The current state and future plans are discussed in Section 6. Finally, Section 7 conclusions the report.

2 Related Work

In this section, first, the related research works in ANN parallelisation areas are reviewed and categorised. After that, the workload partitioning algorithms for performance optimisation of parallel platforms will be addressed. Then, the latest research works addressing the energy optimisation problem in High-Performance Computing (HPC) platforms will be discussed.

2.1 ANN Parallelization

Up to now, there are lots of researches that have focused on the parallel execution of ANNs to reduce the execution time. Most of these studies have focused on homogeneous environments, not heterogeneous ones. The majority of these reseraches have never focused on optimizing energy consumption as a separate objective. Only some of them have investigated

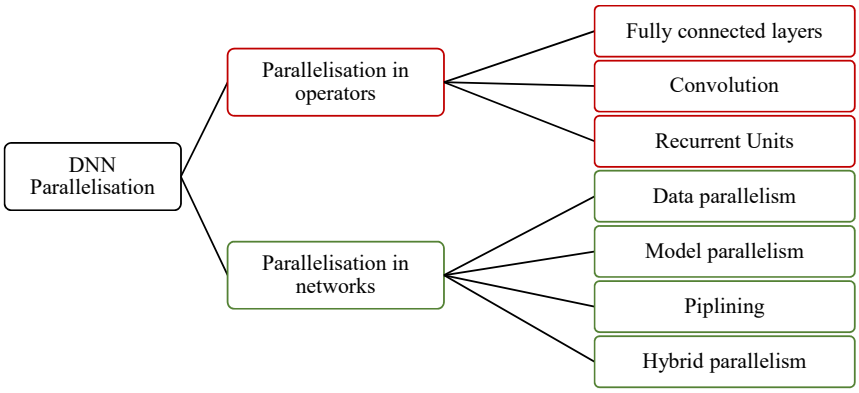

Figure 1: Categorisation of ANNs parallelisation researches.

the reduction of energy consumption as a positive side effect of these parallelization. Despite these differences between previous researches and ours, the investigation of these studies has given us a good idea of the state of the art in parallelization methods and which methods should be implemented in ANNs to be used in our research.

ANN parallelisation methods can be categorized in 2 main categories: Parallelisation in Operators, and Parallelisation in Networks. Each of these two main categories has several subcategories (Figure 1).

In parallelisation in operators studies, the researchers have tried to take advantage of the opportunities for parallelizing layer execution. In some of ANNs' types, computations can be parallelized directly; and in other networks types, computations have to be reshaped to reveal parallelism [4]. In the following the parallelism of three popular operators have been addressed:

- **Fully Connected Networks:** A fully connected layer can be expressed and modeled as a matrix-multiplication of the weights and the neuron values. To this aim, efficient linear algebra libraries, such as CUBLAS [5], MKL [6], and ESSL [7], can be used.

  Vanhoucke et al. [8] have presented some different methods to further optimize CPU implementations of fully connected layers. In particular, this research shows efficient loop construction, vectorization, blocking, unrolling, and batching. These researchers also proved how weights can be quantized to use fixed-point math instead of floating-point.

- **Convolutional neural network (CNNs):** The research community and the industry have paied considerable efforts into optimizing CNNs computation on different platforms. The first algorithmic change proposed for CNNs was the use of the famous technique to transform a discrete convolution into matrix multiplication, using Toeplitz matrices (usually known as im2col) [9].

  The second method proposed for CNNs is using of the Fourier do-

main, in which convolution is defined as an element-wise multiplication. In this method, both the data and the kernels are transformed using FFT, multiplied, and the inverse FFT is applied on the result [10].

The third and the prevalent method used today to perform CNNs is Winograd's algorithm for minimal filtering [11]. This method, first, proposed by Lavin and Gray [12], and modifies the original CNN algorithm for multiple filters that there are in convolutions.

- **Recurrent Neural Networks (RNNs):** The gate systems that situate within RNN units (e.g., LSTMs) contain multiple operations, each of which does a small matrix multiplication or an element-wise operation. Due to this reason, these layers were commonly implemented as a series of high-level operations; but the further acceleration of such layers is possible. On the other hand, RNN units are usually chained together (forming consecutive recurrent layers), so two types of parallelism can be considered: within the same layer, and between consecutive layers [4].

The high average parallelism in neural networks may not only be done to compute individual operators efficiently but also to parallelise the whole network with respect to different dimensions. In the following the main partitioning strategies (parallelisation in networks) have been addressed: partitioning input samples (data parallelism), partitioning the network structure (model parallelism), and partitioning the layer (pipelining) [4].

- **Data Parallelism:** This method is a straightforward approach for ANN parallelization; in this method the work of the minibatch samples is partitioned among multiple computational resources (cores or devices). Today, data parallelism is supported by the vast majority of ANN frameworks, using a single GPU, multiple GPUs, or a cluster of multi-GPU nodes [4].

  In this method, all ANN parameters have to be accessible for all processors, which means that they should be replicated. The scaling of data parallelism is naturally defined by the minibatch size. In the weight update phase, the results of the partitions have to be averaged to obtain the gradient [13].

- **Model Parallelism:** Model parallelism is also known as network parallelism. This strategy divides the work according to the neurons in each layer. In this method, the minibatch samples are copied to all processors, and different parts of the ANNs are computed on different processors. This method can conserve memory (since the full network is not stored in one place) but cause additional communication after every layer [14].

- **Piplining:** In neural networks, pipelining can either refer to overlapping computations (for example, between one layer and the next one as data becomes ready); or to partitioning the ANNs according to depth, assigning layers to specific processors.

  In another view, pipelining can be a form of data parallelism, since the samples are processed through the network in parallel as model parallelism.

The first and widely used in practice form of pipelining is overlapping of feedforward, backpropagation, and weight updates [15]. This scheme increases utilization by mitigating processor idle time.

- **Hybrid Parallelism:** In some researches, researchers try to combine multiple parallelism schemes to overcome the drawbacks of each scheme [16–18].

In this research, we intend to implement different types of ANNs (Fully Connected, CNNs, and RNNs) with different related parallelization methods (categorised above), and then by analyzing the applications' profiles and applying our workload partitioning algorithms, we will optimize their performance and energy consumption on heterogeneous platforms.

2.2 Performance Optimisation

The simplest approach for performance optimisation of parallel applications executing on parallel platforms with a few cores is the Constant Performance Model (CPM). In this technique, the speed of applications is characterized using positive constant numbers such as normalized cycle time, normalized processor speed, average execution time, etc [19–21]. In this approach, it is assumed that there is no dependency between the performance of processors and workload size, and the optimal solutions balance workloads between processing elements (cores).

The advanced load-balancing algorithms use application-specific models such as the Functional Performance Model (FPM). In FPM, the speed of processors is modelled by continuous functions of problem size where the shapes of these functions are supposed to be smooth enough. This way, it is guaranteed that optimal solutions minimizing the execution time are always load-balanced [22, 23].

However, new developments such as increased number of cores and Non-Uniform Memory Access (NUMA) lead to unprecedented complexities in nodal architectures of modern HPC platforms. Due to these complexities, the shapes of speed profiles of applications on modern parallel platforms are no longer smooth and deviate significantly from the conditions assumed by traditional load-balancing models such as FPM. To overcome the limitations of the FPM-based load-balancing algorithms, new model-based optimisation algorithms are proposed considering the real-life performance profiles of applications [24–28]. These algorithms take the most general shapes of performance profiles as input and find optimal workload distributions minimising execution time. Optimal solutions found by these algorithms, unlike load-balancing algorithms, may not load-balance an application.

2.3 Energy Optimisation

Prior to the many-core era, energy profiles of applications executing on platforms with a few number of cores exhibited linear shapes with minor variations. That is why research works analytically modelled energy consumption of parallel applications executing on multi-core platforms and did not consider workload size as an input parameter [29–32]. In [33], it

is mathematically proved that in platforms with linear performance and energy profiles, performance optimisation leads to energy optimisation.

Due to the complex nodal architectures of recent HPC systems, energy profiles, like performance profiles, exhibit a non-linear correlation between energy consumption and workload size with lots of variations in their shapes. Because of this feature, the energy optimisation algorithms developed for single-cores fail to find optimal workload distributions minimising energy consumption in many-core platforms. Research works in [33–38] propose novel variation-aware workload partitioning algorithms for energy optimisation of applications executing on modern many-core HPC platforms. These algorithms take advantage of a novel component-level energy measurement approach to build real-life dynamic energy profiles of parallel applications executing on hybrid many-core platforms.

3 Research Questions and Hypotheses

One of the most important experts' concerns is that ANNs (in general, AI and machine-learning algorithms) are consuming more energy, using more data, and training for a longer time, day by day. It is not just a concern for academic research centers. As more industries begin to use AI, this concern is increasing that the technology will deepen the climate crisis [2].

On the other hand, over the last few years, the modern HPC platforms have become highly heterogeneous owing to the tight integration of multicore CPU processors and accelerators (such as GPUs, Intel Xeon Phis, or FPGAs) [3].

Our main objective is to minimize the execution time and the energy consumption of ANNs via workload partitioning algorithms. The target platform executing ANNs is a modern hybrid heterogeneous server integrating multi-core CPUs and various accelerators (more details Section 4.1), and our approach to achieve this goal is through the distribution of the workload between heterogeneous devices of the server.

The main research questions that we are going to answer in this research are:

- The use of workload partitioning algorithms, how much can affect the performance and energy consumption of each ANNs types (Fully Connected, CNNs, and RNNs) on hybrid heterogeneous platforms?
- For each type of ANNs, which method of parallelization has the best performance and minimum energy consumption in hybrid heterogeneous platforms?
- Which type of parallelization methods has the best performance and minimum energy consumption when it uses workload partitioning algorithm in heterogeneous platforms?

The following hypotheses are considered to find the answers to the above questions:

- There are many available implementations of ANNs, the code of which can be used to develop a set of configurable parallel applica-

tions for execution on hybrid heterogeneous servers for the use in our research.
    - After spending several months, we realized that this hypothesis was wrong (more details in section 5.1).
- After investigating the ANN applications' profiles, we can decide how to distribute the workload between heterogeneous devices, and this workload partitioning leads to optimize performance and energy consumption.
- Our proposed approach is a general-purpose method that can be applied to different platforms without dependence on specific platforms or ANNs' types.
- Our ANNs applications and our optimisation methods can compete with TensorFlow [39], Torch [40], and NVIDIA DIGITS [41] (The reason for selecting these packages is presented in Section 5.1.), in two following aspects:
    - Comparing the execution time and energy consumption of our ANNs with the other applications while the structures, configurations, and datasets are the same.
    - Applying our optimisation method to these selected ANNs packages to find our optimisation method effects on their execution time and energy consumption.

In the different stages of this research, we are looking to find the answers to the above research questions and evaluate our hypotheses.

4 Proposed Approach

Our main research goal is to minimize the execution time and the energy consumption of ANNs. To obtain this goal, the workload partitioning algorithms will be applied. The target platforms to execute ANNs are modern hybrid heterogeneous servers that integrating multi-core CPUs and various accelerators.

Our proposed main task blocks to achieve this research objective are:

- Implementing serial version of different types of ANNs (Fully connected, CNNs, and RNNs)
- Parallelise the ANNs' applications on CPUs and GPUs using different parallelisation methods, separately.
    - Table 1 shows the all combinations of ANNs' types and parallelization methods (there are 15 combinations).
    - For the first time, we are going to implement comprehensive configurable parallelised ANNs' applications that includes different types of ANNs (Fully Connected, CNN, and RNN) paralleised with different parallelisation methods. These applications can be the first benchmark in this research area.
    - We have implemented the Fully Connected ANN parallelised via the Data Parallelism method (the **X** in Table 1). The idea

is that during this research we fill all cells of this table. Finally, our presented ANN applications will be a comprehensive configurable parallelised ANNs applications.

- Develop the hybrid-application of each ANNs' types and the parallelization methods. It means the applications can run on CPUs and GPUs resources simultaneously.
- Study performance and energy behaviours of our ANNs (the profile of ANNs),
- Develop the workload partitioning algorithms,
- Optimise the performance and energy consumption of our ANNs apllication.

Table 1: Combinations of different ANN types and parallelization methods.

	Fully Connected	**CNNs**	**RNNs**
Operators: Fully Connected, CNN, RNN			
Networks: Data Parallelism	**X**		
Networks: Model Parallelism			
Networks: Pipelining			
Networks: Hybrid Parallelism			

Figure 2 is our Gantt chart for this research. As shown in this Gantt chart, the profile extraction step (Studying ANNs' Behaviours in chart) is time-consuming, so in the meantime, in parallel, the other combinations of ANNs types and the parallelisation methods are implemented (Implementing ANNs' Types, Parallelising on CPUs and GPUs, and Developing Hybrid ANNs steps in chart).

In a nutshell, our main contributions and novelties in this research are:

- For the first time, we are going to implement a comprehensive parallelised ANNs applications. These applications include different types of ANNs (Fully Connected, CNN, and RNN) that they have been paralleised with different parallelisation methods. These applications can be the first benchmark in this research area.
- For each combinations of ANNs' types and parallelisation methods, we will study performance and energy behaviours of ANNs on heterogeneous platforms to obtain and analyse their profiles.
- We are going to develop and apply the workload partitioning algorithms. The outputs of these algorithms will minimize the execution time and the energy consumption of ANNs.

In the following of this section, our experiments platforms in this research will be addressed.

4.1 Our Experiments Platforms

Experiments of this research are executed on two servers containing an Intel Haswell multicore CPU, NVidia K40c and 100 PCIe GPU, and Intel Xeon Phi and Gold (specifications in Tables 2 and 3).

Table 2: HCLServer1: Specifications of the Intel Haswell multicore CPU, Nvidia K40c, and Intel Xeon Phi 3120P.

Intel Haswell E5-2670V3	
No. of cores per socket	12
Socket(s)	2
CPU MHz	1200.402
L1d cache, L1i cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30720 KB
Total main memory	64 GB DDR4
Memory bandwidth	68 GB/sec
NVIDIA K40c	
No. of processor cores	2880
Total board memory	12 GB GDDR5
L2 cache size	1536 KB
Memory bandwidth	288 GB/sec
Intel Xeon Phi 3120P	
No. of processor cores	57
Total main memory	6 GB GDDR5
Memory bandwidth	240 GB/sec

Table 3: HCLServer2: Specifications of the Intel Skylake multicore CPU and Nvidia P100 PCIe.

Intel Xeon Gold 6152	
Socket(s)	1
Cores per socket	22
L1d cache, L1i cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30976 KB
Main memory	96 GB
NVIDIA P100 PCIe	
No. of processor cores	3584
Total board memory	12 GB CoWoS HBM2
Memory bandwidth	549 GB/sec

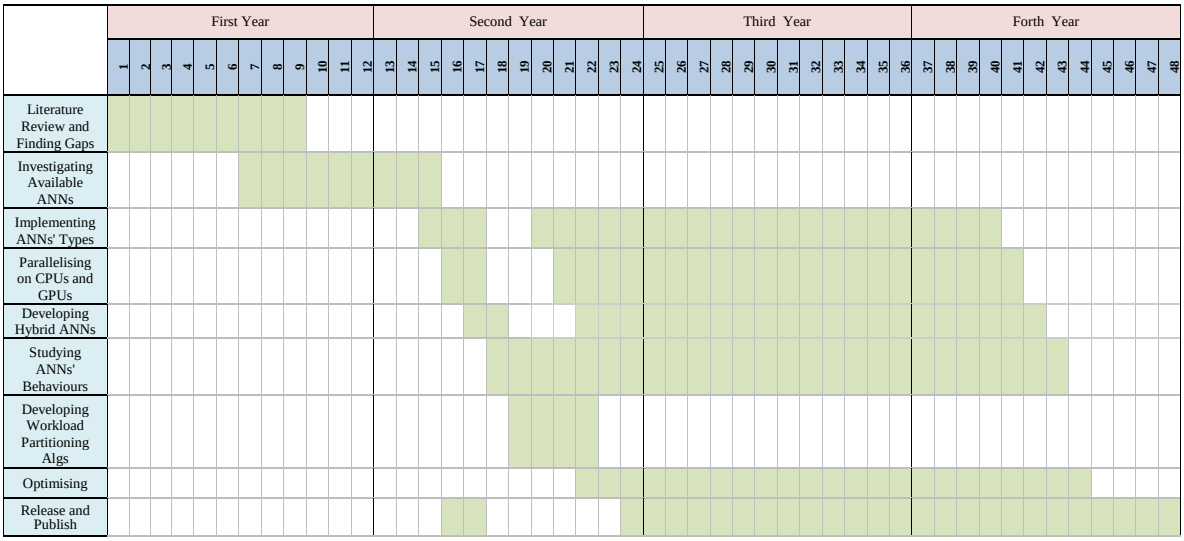

Figure 2: The research Gantt chart.

5 Progresss to Date

First, we explain how to evaluate our first hypothesis and the obtained results. Then, we mention more details of the first version of our ANNs (Fully Connected / Data Parallelism method) that has been implemented.

5.1 Available ANN Applications

There are many available implementations for ANNs in different programming languages. As mentioned before, our first hypothesis is that "There are many available implementations of ANNs, the code of which can be used to develop a set of configurable parallel applications for execution on hybrid heterogeneous servers for use in our research.".

To evaluate this hypothesis, we have investigated lots of available ANN implementations. Considering the goals of this research, we were looking for implementations that satisfy the following specifications:

- Open Source,
- Parallelised on at least CPUs and GPUs,
- Run on hybrid platforms (Simultaneous execution on different types of processors),
- General usage (No dependency on specific platforms),
- Simple to extend it and apply different optimisation methods

Table 4 lists some of the most well-known available ANN packages [42–44]. This table compares the applications based on the supported types of ANNs (Fully Connected (FC in the table), CNN, and RNN),

their parallelisation status (parallelised on CPUs, GPUs, and Hybrid platforms), and Open Source (OS). When we applied the above features as our filters to this list of ANN implementations, we lost them one by one.

As shown in Table 4, only a few numbers of investigated ANN applications have 'Y' in all columns (TensorFlow [39], Torch [40], and NVIDIA DIGITS [41]); and among them, none is simple enough and well-documented to consider as our benchmark to apply our optimization methods on heterogeneous platforms.

Table 4: Comparing some well-known available ANNs applications.

App Name	OS	FC	CNN	RNN	CPUs	GPUs	Hybrid
Neural Designer	N	Y	N	N	Y	Y	Y
Neuroph	Y	Y	N	N	N	N	N
Darknet	Y	Y	Y	Y	Y	Y	N
Keras	N	Y	Y	Y	Y	Y	Y
TensorFlow	Y	Y	Y	Y	Y	Y	Y
TFlearn	N	Y	Y	Y	Y	Y	Y
ConvNetJS	Y	Y	Y	N	N	N	N
NeuroSolutions	N	Y	N	N	N	N	N
NVIDIA DIGITS	Y	Y	Y	Y	Y	Y	Y
SNNS	N	Y	Y	Y	Y	Y	Y
Torch	Y	Y	Y	Y	Y	Y	Y
MLPNeuralNet	Y	Y	N	N	N	N	N
DNNGraph	Y	Y	Y	N	N	N	N
DeepPy	N	Y	Y	N	N	Y	N
NeuralN	N	Y	N	N	Y	Y	Y
NeuralTalk2	Y	Y	Y	N	Y	Y	N
Aforge.Neuro	N	Y	N	N	N	N	N
Cuda-convnet2	N	Y	Y	N	N	N	N
Dn2A	N	Y	Y	Y	N	N	N
Knet	Y	Y	Y	Y	Y	Y	N
HNN	Y	Y	N	N	Y	Y	N
Lasagne	N	Y	Y	Y	N	N	N
Mocha	Y	Y	Y	N	Y	Y	N
LambdaNet	N	Y	N	N	Y	Y	N
gobrain	N	Y	N	Y	N	N	N
neon	N	Y	Y	Y	N	Y	N
Deeplearn-rs	Y	Y	Y	Y	N	N	N
RustNN	N	Y	N	Y	Y	Y	N

After spending several months on our first hypothesis, we have concluded that it is wrong. So we decided to present our ANNs applications as one of our important contributions in this research.

These applications will have the following specifications:

- Implemented using C++ programming language,

- Open Source,
- Support different types of ANNs (Fully Connected, CNNs, and RNNs)
- Support different parallelisation methods (listed in Table 1)
- Parallelised on CPUs and GPUs using OpenMP and OpenACC packages,
- Run on heterogeneous hybrid platforms,
- Independent from specific platforms,
- Simple, configurable, and well-documented.

5.2 First Version of our ANNs

As first version of our ANNs' benchmark, we implemented the Fully Connected ANN using C++ programming language. To parallelise this implementation, the Data Parallelism method (partitioning by input samples) was used because:

- This method is consistent with the concept of batching/mini-batching ANNs that is even used in non-parallel implementations.
- Today, this method is supported by the vast majority of ANN frameworks for parallelisation on GPUs.
- This method can also be implemented on the CPUs efficiently.

Figure 3 shows how our Data Parallelism implementation works. First, the data-set is batched. To each available processor (Ps in Figure 3), one batch of data is assigned. Each processor has a copy of the network and for its current batch does the training steps (Feed Forward, Back-propagation, and updating the neuron's weights in a Fully Connected network). After processing the batch, the processors merge their networks together (this merge means obtaining the avg of neuron weights). The updated network will be shared to the processors and used in the next iteration. In the next iteration, each processor has a new batch of data to process.

6 Current State and Future Plans

In this section, the current state and future plans of my PhD program is presented.

- **Current state**
  1. Literature review of previous researches
     - We are writing a survey article to explain the state of the art of performance and energy optimization of ANNs applications on heterogeneous platforms.
  2. Investigation of available ANN implementations to select one of them as a benchmark to apply our optimisation method.

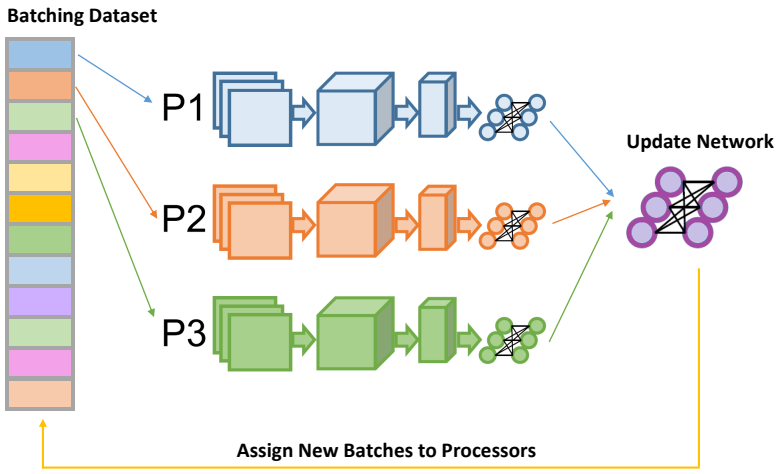

Figure 3: Data parallelisation method.

  - We spent several months in this step but finally, found that none of the available ANN implementations can satisfy all features that we need to do this research (more details in Section 5.1). So we decided to implement our ANNs.
3. Implementation of Fully Connected ANNs using C++ programming language.
  - This language is supportrd by different parallelisation packages (e.g. OpenMP [45], and OpenACC [46]).
4. Parallelisation of our Fully Connected ANN on CPUs and GPUs
  - OpenMP [45] to parallelise on CPUs; this package is efficient enough, and has relatively high level of abstraction.
  - OpenACC [46] to parallelise on GPUs; this package is similar to OpenMP. There is no need to explicitly address the hardware. It has more portability and needs less programming effort.
5. Developing hybrid application to execute our first implementation on heterogeneous platforms.
  - This step is on-going.

- **Future plans**
  1. Developing a model-based data partitioning algorithm to minimize the energy consumption for data-parallel applications executing on large scale heterogeneous systems.
  2. Studying how performance-aware data partitioning of data-parallel applications affects their energy consumption and making a trade-off between performance and energy consumption in heterogeneous platforms.
  3. Repeating the parallelisation and optimisation process on heterogeneous platforms for other types of networks:

   - Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN)
4. Publishing the outcomes. The following journals can be appropriate for publishing the articles derived from this research:
   - IEEE Transactions on Parallel and Distributed Systems
   - The Journal of Supercomputing, Springer
   - IEEE Access
   - Energies, Multidisciplinary Digital Publishing Institute (MDPI)

7 Conclusion

Today, ANNs regularly produce great achievements, like automatic images recognition, conversing, interpreting textual documents, and driving vehicles. These great achievements need lots of time, computing resources, and electricity.

On the other hand, the modern high-performance computing platforms have become highly heterogeneous due to the tight integration of multicore CPU processors and accelerators (like GPUs).

In this research, we are going to use the modern heterogeneous servers to reduce the concerns about the growth of ANN applications and their performance and energy consumption. Optimal resources usage on heterogeneous platforms can significantly increase performance and reduce energy consumption.

In a nutshell, our research objective is to minimize the execution time and the energy consumption of different types of ANNs (Fully Connected, CNNs, and RNNs) to execute on hybrid heterogeneous platforms. Our approach to achieve this goal is through the distribution of the workload between heterogeneous devices.

So far, we have reviewed the related works, investigated the available ANNs applications, and implemented our first application (that is a Fully Connected ANN and parallelised using the Data Parallelism method on CPUs and GPUs).

In the future, we are going to develop a model-based data partitioning algorithm to minimize the energy consumption for data-parallel applications on large-scale heterogeneous systems. Then, we will repeat the parallelisation and optimization process on heterogeneous platforms for other combinations of ANNs' types and parallelism methods.

References

[1] K.-L. Du and M. N. Swamy, *Neural networks and statistical learning*. Springer Science & Business Media, 2013.

[2] W. Knight, "Ai can do great things—if it doesn't burn the planet. wired," 2020. [Online]. Available: https://www.wired.com/story/

[3] Top500, "Top500," 2022. [Online]. Available: https://www.top500.org/lists/

[4] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[5] NVIDIA, "CUDA Toolkit Documentation," 2022. [Online]. Available: http://docs.nvidia.com/cuda/cublas

[6] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," in *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 2014, pp. 167–188.

[7] IBM, "Engineering and Scientific Subroutine Library (ESSL). version 6.2 guide and reference." 2019. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSFHY8_6.2/reference/essl_reference_pdf.pdf

[8] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," *In Proceedings of the Deep Learning and Unsupervised Feature Learning Workshop (NIPS'11)*, 2011.

[9] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

[10] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A GPU performance evaluation," *In Proceedings of the International Conference on Learning Representations (ICLR'15), year=2015*.

[11] S. Winograd, "Arithmetic complexity of computations. society for industrial and applied mathematics," 1980.

[12] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4013–4021.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[14] B. Forrest, D. Roweth, N. Stroud, D. Wallace, and G. Wilson, "Implementing neural network models on parallel computers," *The Computer Journal*, vol. 30, no. 5, pp. 413–419, 1987.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015.

[16] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[17] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[18] A. L. Gaunt, M. A. Johnson, M. Riechert, D. Tarlow, R. Tomioka, D. Vytiniotis, and S. Webster, "AMPNet: Asynchronous model-parallel training for dynamic neural networks," *arXiv preprint arXiv:1705.09786*, 2017.

[19] M. Cierniak, M. J. Zaki, and W. Li, "Compile-time scheduling algorithms for a heterogeneous network of workstations," *The Computer Journal*, vol. 40, no. 6, pp. 356–372, 1997.

[20] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *Journal of Parallel and Distributed Computing*, vol. 61, no. 4, pp. 520–535, 2001.

[21] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix multiplication on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1033–1051, 2001.

[22] A. Lastovetsky and R. Reddy, "Data partitioning with a realistic performance model of networks of heterogeneous computers," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* IEEE, 2004, p. 104.

[23] ——, "Data partitioning with a functional performance model of heterogeneous processors," *The International Journal of High Performance Computing Applications*, vol. 21, no. 1, pp. 76–90, 2007.

[24] P. K. Smolarkiewicz and W. W. Grabowski, "The multidimensional positive definite advection transport algorithm: Nonoscillatory option," *Journal of Computational Physics*, vol. 86, no. 2, pp. 355–375, 1990.

[25] A. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of EULAG kernel on intel xeon phi through load imbalancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 787–797, 2016.

[26] W. Zhang, X. Ji, B. Song, S. Yu, H. Chen, T. Li, P.-C. Yew, and W. Zhao, "Varcatcher: A framework for tackling performance variability of parallel workloads on multi-core," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1215–1228, 2016.

[27] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2176–2190, 2018.

[28] ——, "A hierarchical data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous multi-accelerator NUMA nodes," *IEEE Access*, vol. 8, pp. 7861–7876, 2019.

[29] J. Li and J. F. Martinez, "Power-performance considerations of parallel computing on chip multiprocessors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 2, no. 4, pp. 397–422, 2005.

[30] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH computer architecture news*, vol. 35, no. 2, pp. 13–23, 2007.

[31] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *Proceedings of the*

17th international conference on Parallel architectures and compilation techniques, 2008, pp. 177–186.

[32] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 694–699.

[33] A. Lastovetsky and R. R. Manumachu, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, 2016.

[34] A. W. Lewis, S. Ghosh, and N.-F. Tzeng, "Run-time energy consumption estimation based on workload in server systems." *HotPower*, vol. 8, pp. 17–21, 2008.

[35] R. R. Manumachu and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy," *IEEE Transactions on Computers*, vol. 67, no. 2, pp. 160–177, 2017.

[36] ——, "Parallel data partitioning algorithms for optimization of data-parallel applications on modern extreme-scale multicore platforms for performance and energy," *IEEE Access*, vol. 6, pp. 69 075–69 106, 2018.

[37] H. Khaleghzadeh, M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on heterogeneous HPC platforms for performance and energy through workload distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 543–560, 2020.

[38] H. Khaleghzadeh, M. Fahad, R. Reddy Manumachu, and A. Lastovetsky, "A novel data partitioning algorithm for dynamic energy optimization on heterogeneous high-performance computing platforms," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, p. e5928, 2020.

[39] "Tensorflow," 2022. [Online]. Available: https://www.tensorflow.org/

[40] "Scientific computing framework for luajit," 2022. [Online]. Available: http://torch.ch/

[41] "NVIDIA DIDITS," 2022. [Online]. Available: https://docs.nvidia.com/deeplearning/digits/digits-tutorial/index.html

[42] A. Beatrice, "Top 10 must-know artificial neural network software," September 2020. [Online]. Available: https://www.analyticsinsight.net/top-10-must-know-artificial-neural-network-software/

[43] "Top 27 Artificial Neural Network Software," March 2022. [Online]. Available: https://www.predictiveanalyticstoday.com/top-artificial-neural-network-software/

[44] H. M. Pandey and D. Windridge, "A comprehensive classification of deep learning libraries," in *Third International Congress on Information and Communication Technology*, X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds. Singapore: Springer Singapore, 2019, pp. 427–435.

[45] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[46] "OpenACC," 2022. [Online]. Available: https://www.openacc.org/