|epcc|

Deep Learning on SpiNNaker: Report

Jonas Fassbender
jonas@fassbender.dev

Abstract

This paper reports on the preliminary research done for the dissertation project "Deep Learning on SpiNNaker" (formerly "A Tensorflow Backend to SpiNNaker"). SpiNNaker is a novel neuromorphic, massively parallel and energy efficient computer architecture designed for the task of modeling the human brain. The goal of this dissertation project is to investigate the acceleration of training deep neural networks on SpiNNaker. The preliminary research done for the dissertation phase in the summer of 2020 is described and all relevant findings are presented. Changes made to the initial project proposal are discussed. An outline for the dissertation phase in the form of a work plan, risk analysis and relevant background information are also presented. A literature review of Nita (2018) is given at the end of this report.

Contents

1 Introduction

SpiNNaker is a digital neuromorphic computer architecture, designed to model the human brain. According to the SpiNNaker project's website:

> SpiNNaker is a novel massively-parallel computer architecture, inspired by the fundamental structure and function of the human brain, which itself is composed of billions of simple computing elements, communicating using unreliable spikes (SpiNNaker Team, 2020).

This is rather a gross simplification of what SpiNNaker was designed for. Understanding the human brain is one of the biggest frontiers in science and there are many ideas on how one can model such a complex and still mostly unknown structure. There are many different proposed neuromorphic computer architectures. From early analogue approaches for vision systems like presented in Mead (1989) to large scale digital architectures like Blue Brain (Markram, 2006). All are aiming to implement different types of neurons and models efficiently (Furber and Temple, 2007).

SpiNNaker is an architecture for the large scale and real-time modeling of spiking neural networks (Furber et al., 2006a,b; Furber and Temple, 2007). Maass (1997) describes spiking neural networks as the third generation of neural networks. Unlike the second generation neural networks—deep neural networks—spiking neural networks incorporate a concept of time in their action potentials (spikes). Neural networks of the second generation do not have a concept of time and fire every time they are activated. Spiking neural networks are derived from evidence collected from biological neural systems, which use the timing of spikes to encode information; they are therefore more closely representing neural systems of biological life than deep neural networks (Maass, 1997).

The SpiNNaker chip is a multiprocessor chip, which has 18 ARM cores, with a Network-on-Chip system for communication between cores (Furber and Temple, 2007; SpiNNaker Team, 2020). Each core can model up to 1000 spiking neurons. The spiking neurons in the neural network communicate with spike events (Furber and Temple, 2007). Spike events are—like their biological counterpart—inherently unstable and small. This is represented in the interconnection hardware. Small packets (40 or 72 bits) can be sent without guarantee of successful arrival. Even without this guarantee, one can still perform meaningful computations on SpiNNaker. Fault detection and recovery mechanisms are implemented (SpiNNaker Team, 2020). 48 SpiNNaker chips are mounted on a SpiNNaker board. One can connect many SpiNNaker boards together, to build a massively parallel and energy efficient supercomputer that is able to run up to a billion spiking neurons in real time (Furber and Temple, 2007). Photos of the SpiNNaker hardware can be found in Appendix A.

SpiNNaker is targeted at three main areas of research: (i) neuroscience: understanding the human brain, (ii) robotics: low power hardware for implementing neural decision systems and (iii) computer science: new approach to supercomputing and massive parallelism (SpiNNaker Team, 2020). The dissertation project will concern itself with (iii) in the context of utilizing SpiNNaker for the acceleration of training deep neural networks.

Deep learning is one of the research areas of computer science, which has emerged a s a driving force behind advancements in many fields like speech and image recognition, drug discovery and genomics (LeCun et al., 2015). Deep neural networks face a major problem: the sheer amount of computation needed for training them. Researchers at OpenAI have estimated that the amount of computation needed for training state-of-the-art deep neural networks increases exponentially, doubling every 3.4 months (Amodei et al., 2019). In order to cope with such unprecedented amounts of computation and energy, the search for specialized hardware is well underway. Current hardware trends for accelerating the training of deep neural networks

are ASICs (application specific integrated circuits) like TPUs and general purpose GPUs (Jouppi et al., 2017; Mittal and Vaishay, 2019).

The goal of this dissertation project will be to analyze if SpiNNaker could be an energy efficient, scalable and fast alternative to the above mentioned hardware. Since deep neural networks are derived from the human brain and nervous system (Goodfellow et al., 2016), and SpiNNaker was designed to model the human brain, it seems plausible that SpiNNaker will be a good target for accelerating the training of deep neural networks.

The findings of the preliminary work and the changes made to the original project scope are the focal point of this report. The original title of the dissertation project was "A Tensorflow Backend to SpiNNaker". Tensorflow is a library for running fast linear algebra operations on distributed, heterogeneous systems, mainly designed for implementing computationally demanding machine learning algorithms like deep learning in a fast manner (Abadi et al., 2015). Because of SpiNNaker's specialized design (which works rather contrary to that of Tensorflow and current hardware trends for building accelerators for deep learning and fast linear algebra operations), interfacing between SpiNNaker's runtime and Tensorflow was deemed too difficult and not beneficial (see Section 6). Instead, this dissertation aims at implementing deep learning directly on SpiNNaker. Mario Antonioletti and Alan Stokes (2019) shows the original dissertation project's scope.

This paper starts with presenting the final project proposal, in particular focusing on the benchmark to be conducted in Section 2. Then related work is presented in Section 3. The main focus lies on presenting papers with benchmarks we can compare our implementation against. The paper continues by giving a work plan in Section 4, before presenting a risk analysis in Section 5. The preliminary findings outlined above are then discussed in more depth in Section 6. Finally Section 7 will contain a review of a related dissertation project: "Deep Learning Performance on Different Architectures" by Spyro Nita (Nita, 2018).

2 Final Project Proposal

This section will begin by giving an outline of the initial project proposal. The preliminary findings given in Section 6 led us to abandon it. The updated version is given below. The benchmark we will conduct in order to compare our deep learning implementation on SpiNNaker with other hardware platforms and deep learning libraries is presented.

The final project proposal will differ from the initial one in two meaningful ways: (i) we will implement deep learning algorithms and layers directly on SpiNNaker, instead of implementing a Tensorflow backend (see Section 6) and (ii) a different benchmark to measure the performance of SpiNNaker will be used. At first glance, implementing a Tensorflow backend in order to enable deep learning on SpiNNaker seemed wise. Tensorflow already supports various hardware as its targets, like CPUs, GPUs or TPUs (Abadi et al., 2015; Jouppi, 2016). Without further knowledge, adding a backend to SpiNNaker seemed the best way to get deep neural networks running on SpiNNaker. We thought that this would be the road with the least work to be done. Simply implementing the low-level linear algebra operations that Tensorflow offers was deemed less work than having to deal directly with deep learning and its complexity. As the preliminary research revealed, the discrepancy between Tensorflow's architecture and SpiNNaker's is too big to overcome and to get a working deep learning neural network running in the limited amount of time for the dissertation was estimated to be impossible. This topic is further discussed in Section 6.

Concerning (ii), the benchmark proposed by the initial proposal was to compare our native deep learning implementation against the snn toolbox (Rueckauer et al., 2017). The snn toolbox allows taking pre-trained

deep neural networks and converting them to spiking neural networks, which then can be deployed on SpiNNaker. The problem with this benchmark is the fact that only pre-trained deep neural networks can be used and only inference would be possible on SpiNNaker. We do not aim to do computationally inexpensive inference (at least compared to training). Rather we want to use SpiNNaker where it would be needed more: training a deep neural network—a computationally very costly operation (see Section 1).

Instead of benchmarking against the snn toolbox, we intend to go for a much more ambitious benchmark. We want to benchmark the performance of our implementation on SpiNNaker against other state-of-the-art hardware and software libraries for deep learning. For this we chose the domain of computer vision, which is very popular for deep learning. It provides many good benchmarks we can compare against (see Section 3).

Currently, the most famous and arguably the biggest challenge for benchmarking computer vision models is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), based on the ImageNet dataset. The ImageNet dataset consists of more than 14 million images, organized according to the WordNet hierarchy into over 21 thousand so-called synsets (synonym sets) (Russakovsky et al., 2015; Miller, 1995). The ILSVRC is an annual competition running since 2010. It has produced many well-known models, e.g. AlexNet, VGG16 or ResNet-50 (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2015). We intend to benchmark by training the most complex of these three models: ResNet-50. The benchmark will be to train ResNet-50 with 1000 categories (1.28 million images) of the ImageNet dataset (setting of the ILSVRC 2012) and measure the time it takes to finish the training. The top-1 accuracy of the model is evaluated on 50,000 test images and should be at least 74.9%, in order to compare it to the other benchmarks presented in Section 3.

In conclusion, our final project proposal will be implementing deep learning algorithms directly on SpiNNaker, instead of providing a backend to Tensorflow. We will benchmark this implementation and SpiNNaker against other benchmarks based on state-of-the-art deep learning libraries and accelerators for fast training of neural networks, instead of against the snn toolbox. For this benchmark we will train the ResNet-50 model based on the settings described above and compare our results against the results others generated with the same settings but with different libraries and hardware, presented below.

3 Related Work

This section lists and describes related work important for the final project proposal. The main focus lies on benchmarks with which we can compare our implementation on SpiNNaker to state-of-the-art deep learning libraries and—more importantly—hardware. Further literature of importance for this project, e.g. He et al. (2015), Goodfellow et al. (2016) or Russakovsky et al. (2015), can be found in the References.

The benchmark settings described in the previous chapter are the image classification benchmark for training deep learning models of the recently launched MLPerf benchmark project. The MLPerf benchmark was proposed to ensure fair performance comparisons between various hardware and software libraries for deep learning (Mattson et al., 2019). We intend to stick to the rules of the MLPerf benchmark. If the dissertation project succeeds as planned, we will submit our benchmark results to the MLPerf Training benchmark v0.7, which is due on the 21st of February 2021 (submitting the results to the MLPerf Training benchmark v0.7 is not part of the dissertation phase anymore—instead it is part of the follow-up on the dissertation and is therefore not further regarded in this paper).

The results of MLPerf Training v0.6 can be found at `https://mlperf.org/training-results-`

paper	processor/accelerator	#	library	time
He et al. (2015)	Tesla P100	8	caffe	29 hours
Goyal et al. (2017)	Tesla P100	256	caffe2	1 hour
Smith et al. (2017)	TPU pod	1	Tensorflow	30 minutes
Akiba et al. (2017)	Tesla P100	1024	chainer	15 minutes
You et al. (2017)	Xeon Phi KNL	2048	caffe	14 minutes
Jia et al. (2018)	Tesla P40	2048	Tensorflow	6.6 minutes
Mikami et al. (2018)	Tesla V100	2176	NNL	3.7 minutes

Table 1: Selected timings from benchmarks other than MLPerf with a similar setting.

0-6/. Currently the fastest system for the image classification benchmark is based on Tensorflow and 1024 TPUv3s (one full TPUv3 pod) (Stone, 2019). It is able to finish in 1.28 minutes. Besides the MLPerf benchmark, which is quite new, there are a few older papers which use a similar setting for benchmarking their systems. These papers and their results are listed in Table 1. None can beat the fastest implementation on the TPUv3 pod. This is not surprising given the fast pace of improvements in both hardware and software for deep learning and the fact that the MLPerf benchmark results are the most recent.

4 Work Plan

This section will describe the time table for the actual dissertation period. The dissertation period starts on Monday the 25th of May 2020 and will end on Friday the 21st of August 2020. This covers 13 weeks, calendar week 22 to calendar week 34 inclusive.

There are three distinct main tasks to do for the dissertation project, which can be further broken down into more fine-grained subtasks:

- Implementing ResNet-50 on SpiNNaker
  1. Dense layer forward pass (Inference) (see Goodfellow et al., 2016)
  2. Stochastic gradient descent with dense layer backward pass (SGD) (see Goodfellow et al., 2016)
  3. Convolutional layer and ReLu layer (Conv) (see Goodfellow et al., 2016)
  4. Pooling layer (Pool) (see Goodfellow et al., 2016)
  5. Residual learning (RL) (see He et al., 2015)
  6. Optimize implementation (Optimize)
- Benchmarking
- Dissertation writing
  - Background
  - Log
  - Benchmark

CW / Task	22	23	24	25	26	27	28	29	30	31	32	33	34
Implement													
Inference													
SGD													
Conv													
Pool													
RL													
Optimize													
Benchmark													
Writing													
Background													
Log													
Benchmark													
Conclusion													
Proof reading/editing													

Table 2: Gantt chart displaying the work plan for the dissertation period.

- Conclusion
- Proof reading/editing

There are a few dependencies between the subtasks. Benchmarking obviously depends on a finished implementation. Optimizing the implementation and benchmarking will depend on each other and will be done iteratively to the final benchmark. The background for the dissertation can be written independently from implementing and benchmarking. The log section and the benchmark section will depend on the implementation and benchmarking efforts. The conclusion section of the report depends on the log and the benchmark section of the dissertation. The subtasks of implementing ResNet-50 are enumerated, because they are done in succession, in the order they are presented. Only optimizing will depend on the benchmark, all else is independent from the rest.

The Gantt chart presented in Table 2 is derived from these dependencies and displays the work plan for the dissertation period. The most challenging week will be calendar week 32 with four tasks to do in parallel. It will be the week the final benchmark must be finished and the writing of the benchmark section will begin. A lot of time is allocated for the background section of the dissertation. Not all of the time will be necessary, or only a little time per week will be dedicated to the writing of this section. The last week before the deadline will solely be spent on proof reading. Calendar week 33 could be used as a buffer week if the benchmark is not finished after calendar week 32.

Implementation is broken down into many subtasks. One of the risks this project faces will be the author's inability to implement ResNet-50 (see Section 5). But even if that should be the case, one could theoretically run a simple neural network on SpiNNaker, after the SGD task is finished. This would be after the third week and it would allow us to benchmark our implementation on a far smaller scale than ResNet-50 would provide. So presentable results should be achieved early in the dissertation phase, even if after the Inference and SGD tasks problems are encountered. This ensures a successful completion of the dissertation, even if the implementation should not be completable in its full extend.

5 Risk Analysis

This section will look at risks for the successful completion of the dissertation and how they can be overcome. There are four main risks that will endanger the progress of the dissertation:

- Losing direct access to SpiNNaker knowledge
- Hardware destruction
- Failure to implement ResNet-50 on SpiNNaker
- University closes, because of Covid-19

Alan Stokes is one of the dissertation's supervisors and part of the core team of SpiNNaker developers. He is the main source of information concerning the SpiNNaker hardware and runtime. In the case he should not be able to participate and help during this project, the successful completion of the implementation and benchmarking phases would be in jeopardy. To cope with this case, a second source of SpiNNaker knowledge was already established. There exists a Google group with most of the people using SpiNNaker in it, including the other members of the SpiNNaker core team. The SpiNNaker core team has the goal of answering any question asked in the group within 24 hours. Therefore, the risk of failure is reduced to the risk of having to adjust the benchmark, because the implementation phase will take longer.

There is a SpiNNaker board (see Appendix A for a picture of a SpiNNaker board) allocated for work on this dissertation. A SpiNNaker board is a costly piece of hardware. While the destruction of the board would put the author into financial problems, it would not endanger the successful completion of the dissertation, since there is the supercomputer in Manchester (see Appendix A for a picture of the supercomputer) where the work on the dissertation can be done. The benchmarks will be run on the supercomputer anyway. Only the implementation phase could take longer, because rapid prototyping on a dedicated piece of hardware would not be possible, since the supercomputer must be shared with other researchers.

Writing scientific programs like deep neural networks is not a trivial task. So is writing software for a complex runtime such as the SpiNNaker runtime. The benchmark proposed is more ambitious than the originally proposed benchmark. It could be that unknown complications make it impossible to implement ResNet-50 on SpiNNaker in the time span of the dissertation. However, even if the final benchmark is impossible to achieve due to an unfinished implementation, there is the log section of the dissertation (see Section 4). The log section will contain everything done during the implementation and benchmarking phase. Problems encountered will be documented and even if the final benchmark cannot be achieved or the performance is off, the log section will show where the problems were. This would still be a result usable for future efforts of implementing deep learning on SpiNNaker.

The last risk would be the event of a quarantine in Edinburgh, where the university campus will be closed. Already events are canceled and regions in Europe are put under quarantine, because of the Covid-19 virus (Borghese and John, 2020). In case of campus closure, direct contact to the supervisors would be lost. Meetings and discussions would have to be remotely over Skype or an other service. This is currently the main way of communication and would not impact the project. The dedicated SpiNNaker board is also stored on campus. Like stated above, remote access to the supercomputer in Manchester is possible. While taking the dedicated SpiNNaker board away from campus would be possible, it would increase the risk of hardware destruction.

6 Preliminary Findings

This section will present the preliminary findings. The research phase up to this point has been all about setting the right background for a successful dissertation phase. As stated in Section 2, the final project proposal has changed quite a bit from the initial one, thanks to the preliminary findings. There are three main pieces of knowledge gained: (i) the main problem will be the author's ability to program ResNet-50 on SpiNNaker, (ii) implementing a Tensorflow backend is not a feasible thing to do and (iii) the discovery of the MLPerf benchmark, which gives us a perfect setting for finding out if SpiNNaker can compete with state-of-the-art hardware for deep learning.

The first finding was that the programming will be non-trivial. SpiNNaker has a complex runtime. The neuromorphic computer architecture is unlike any other and therefore the programming model differs drastically from more common ones. Overcoming the difficulties of programming on SpiNNaker will be the major challenge of the dissertation. Every task not directly concerned with implementing and writing the report must be minimized or even removed completely.

This was accomplished by the preliminary research in two major ways. From the beginning it was clear that the initially proposed benchmark against the snn toolbox (see Section 2) is not the desired one and really only an afterthought. At the beginning Tensorflow was much more of a focal point. It is yet unknown if SpiNNaker is even a good platform for deep learning. A proper benchmark for finding this out should be the first step, before one can worry about how to interface SpiNNaker with already established libraries for deep learning.

Another reason for disregarding the initial benchmark is the fact that training deep learning neural networks is the computationally expensive operation for which high performance solutions must be developed. Inference is far less demanding. Also, benchmarking SpiNNaker vs. SpiNNaker implementation does not even offer half the insights. The native deep neural network implementation would be compared against the deep neural network transformed to a spiking neural network, using the snn toolbox. Only the implementation would be benchmarked, not SpiNNaker, even though the much more interesting insight would be SpiNNaker's performance compared to accelerated architectures like GPU or TPU clusters.

Because programming SpiNNaker is a hard enough task, having to interface the complex SpiNNaker runtime with the complex Tensorflow runtime was simply deemed too difficult. At the beginning, implementing a Tensorflow backend seemed sensible. Tensorflow already offers interfaces to accelerators like GPUs and TPUs. Research on how to implement hardware backends to Tensorflow revealed that the Tensorflow community recommends working with Tensorflow's extended linear algebra compiler (XLA). A description of XLA can be found at: `https://www.tensorflow.org/xla`. Instead of having to implement every Tensorflow core operation, one has only to implement a XLA backend, which is significantly simpler and more scalable (Tensorflow Team, 2020). Tensorflow Team (2020) describes three types of XLA backend implementations:

1. CPUs without XLA support, either with LLVM backend or not
2. Non-CPUs with LLVM backend
3. Non-CPUs without LLVM backend

SpiNNaker would fall under category three, which is the category requiring the most effort (Tensorflow Team, 2020). Interfacing SpiNNaker with XLA in the time span of the dissertation is not deemed possible.

Another point against a Tensorflow backend would be the fact that SpiNNaker's neuromorphic architecture is quite different from the architecture of common deep learning accelerators and libraries. Tensorflow implements fast linear algebra operations on heterogeneous and distributed hardware (Abadi et al., 2015). Accelerators like GPUs and TPUs are designed for doing fast linear algebra operations using instruction level parallelism. For example, NVIDIA GPUs come with Tensor Cores. A Tensor Core is a specialized piece of hardware performing a 4×4 fused multiply-add-accumulate matrix operation (MAC) per clock cycle (Markidis et al., 2018). Google's first version of the TPU comes with 256×256 MACs performing 8-bit integer operations (Jouppi et al., 2017). The ARM cores on SpiNNaker on the other hand do not have a fast MAC unit. It is designed to run small neurons communicating with each other over unstable spiking events (Furber and Temple, 2007). This does not require fast matrix operations, so there are no dedicated hardware units for it in the core.

Deep learning operations in Tensorflow and on the accelerators are implemented on a layer-level with matrix operations (Goodfellow et al., 2016). Deep learning offers an abstraction much more convenient and closer to SpiNNaker's architecture than matrix operations over the layers of a deep neural network: the neurons inside each layer. For Tensorflow, we would have to implement fast and distributed linear algebra operations, without having the benefit of MAC units. This is not what SpiNNaker was originally designed for and would most certainly lead to an sub-optimal implementation. Implementing deep neural networks on the abstraction level of neurons instead of layers is a more natural fit.

The last argument against a Tensorflow backend was the difference of programming SpiNNaker, compared to other accelerators. TPUs and GPUs do not come with the complexity of a runtime environment like SpiNNaker. GPUs and the TPU come with their own instruction set. The TPU, for example, has a CISC instruction set, with instructions like `MatrixMultiply`, `Read_Host_Memory` or `Write_Host_Memory` (Jouppi et al., 2017). Both GPUs and the TPU have a User Space Driver, which compiles the Tensorflow API to their respective instruction set, generating a binary loaded in the instruction buffer of the device (Jouppi et al., 2017).

SpiNNaker can not be utilized simply by generating an application binary that gets loaded and executed on the device. A SpiNNaker program is written as a graph much like a Tensorflow graph. A vertex does some sort of computation and data is communicated over edges. While edges in a Tensorflow graph are used for passing tensors between vertices, edges in a SpiNNaker graph are used to multicast small packages (40 or 72 bits) from a vertex to the vertices it is connected to (SpiNNaker Team, 2020; Abadi et al., 2015). This design is derived from neurons in the human brain, which communicate the same way (spike events) (SpiNNaker Team, 2020). In order to write a SpiNNaker program, one has to write all vertex types used in the graph using the C programming language, calling SpiNNaker intrinsic functions (e.g. `spin1_send_mc_packet` for sending a multicast to the connected vertices). The graph representation of the program is build on the host, which also sets up the SpiNNaker device it is connected to. For this, a Python interface is provided, which is called a frontend. For the dissertation project, we will use the SpiNNaker Graph Frontend, designed for general computing. There is also a frontend for spiking neural networks using the PyNN modeling language (Davison et al., 2009).

Integrating the complex Tensorflow runtime with the complex SpiNNaker runtime was not deemed possible (limiting factor is the small time span for the project), nor useful. In order to—one day, not as part of this project—integrate commonly used deep learning libraries with SpiNNaker, we think it is more sensible to support the ONNX (Open Neural Network Exchange) format, based on an optimized deep learning library for SpiNNaker (Bai et al., 2020).

With the removal of Tensorflow from the project proposal, we were able to reduce a fairly big chunk of work,

which we probably would not have been able to fit into the time span of the dissertation. To fully reduce the tasks for the successful completion of the dissertation to implementing, benchmarking the implementation and writing the report (see Section 4), MLPerf offers the last bit (see Section 3). At the beginning—already after the ssn toolbox benchmark was disregarded—we were planning on comparing the SpiNNaker implementation against a deep neural network running on a different supercomputer, such as Cirrus (EPCC, 2020). That would have meant that we would have to conduct two benchmarks, one on SpiNNaker, another one on Cirrus. Thanks to MLPerf and the other benchmarks presented in Section 3, we can remove the benchmark on a different architecture, which will save a lot of time needed for implementing ResNet-50 on SpiNNaker.

With removing Tensorflow from the project proposal and having the MLPerf benchmark for comparison, we are able to reduce the tasks to do for a successful dissertation to a minimum. This enables us to focus on the most important tasks, while still maintaining a reasonable level of impact in the case of success and even reducing the risk of failure.

7 Deep Learning Performance on Different Architectures: Review

This section will concern itself with a review of the dissertation of Spyro Nita, which he did for earning his Master of Science in High Performance Computing with Data Science at the University of Edinburgh: "Deep Learning Performance on Different Architectures" (Nita, 2018). First, a summary of his dissertation will be given, before the review of his work is presented. The review will start by looking at how well the context of the dissertation is explained and how well the scope of the dissertation's problem is defined. The last part of the review will be the consideration of the dissertation's layout and formatting. After the review, the importance of the dissertation for "Deep Learning on SpiNNaker" will be discussed and evaluated.

Nita (2018) concerns itself with measuring the computational performance of two different approaches for training deep neural networks: distributed training using GPUs and distributed training using CPUs. The dissertation is a work derived from the efforts of Team EPCC at the Student Cluster Competition at the International Supercomputing Conference 2018, held in Frankfurt. Team EPCC competed against other teams by building a small supercomputing cluster and running various benchmarks on it, to see which team built the best supercomputing cluster. One of these benchmarks concerned itself with measuring the performance of training a deep neural network on the clusters, which serves as the GPU benchmark in Nita (2018).

The benchmark is based on the famous ImageNet dataset, like the ILSVRC (see Section 2). Contrary to the ILSVRC, the benchmark for the Student Cluster Competition only concerns itself with the throughput of images during training and not with the accuracy of the trained models. The throughput is measured in images per second. Only if two teams should have the same throughput during training is the training accuracy taken into account as the secondary criterion for tie-breaking. The teams participating in the Student Cluster Competition had to train the VGG16 deep neural network—a famous model architecture introduced in the ILSVRC 2014 by researchers from Oxford University (Simonyan and Zisserman, 2014). The main limitation for the clusters were their power budget of 3kW and—for the ImageNet benchmark—a maximum runtime of three hours (Nita, 2018).

Nita (2018) compares the throughput of the supercomputing cluster of Team EPCC for the performance of a distributed GPU system against Cirrus—a supercomputer hosted by the EPCC—for the performance of a distributed CPU system (EPCC, 2020).

Concerning the results of Nita (2018), two major problems were encountered: (i) cluster damage and failure of the Team EPCC cluster shortly before the competition and (ii) the fact that the CPU benchmark on Cirrus could not be distributed over multiple backend nodes. Fortunately benchmarks were done on the Team EPCC cluster, before its failure. Nita (2018) presents the results of using the first node of the cluster with six NVIDIA V100 GPUs. Concerning (ii), Nita (2018) presents the results for a single backend node and assumes that Cirrus would be able to linearly scale to multiple backend nodes, when comparing the two clusters. Linear scaling is an overly optimistic assumption and it would have been better, if Nita (2018) had used another CPU benchmark for its comparison, for example You et al. (2017). The single node of the Team EPCC cluster with its six NVIDIA V100 GPUs generated a throughput of 2,052 images per second, while a single backend node of Cirrus (two Intel Xeon E5-2695 processors, each with 18 physical cores) can generate a throughput of just 18 images per second. Assuming linear scaling over multiple backend nodes of Cirrus, 21 nodes would be needed for the throughput of a single NVIDIA V100 GPU. Nita (2018) concludes that GPUs offer a significant advantage compared to CPUs, when it comes to training deep neural networks.

The dissertation starts with an introduction, which includes a description of the dissertation's layout. The second chapter concerns itself with a description of convolutional neural networks for image classification, describing ImageNet and VGG16 as a special convolutional neural network. Chapter three describes everything concerning the Student Cluster Competition, including the benchmarks, rules and various features of the Team EPCC cluster. It also describes used hardware, operating system and other important installed software libraries. The difficulties and hardware damage to the cluster and—concerning the ImageNet benchmark—how the images were preprocessed and reformatted to the TFRecord format are also outlined (Abadi et al., 2015). Chapter four concerns itself with the optimization of the VGG16 neural network, mainly through different parameters which are passed to Tensorflow, which is used for implementing VGG16. It thoroughly describes how one can distribute training across multiple GPUs. The benchmark for the Student Cluster Competition is described and analyzed, as is the benchmark for Cirrus. The chapter ends by comparing both. Nita (2018) concludes by summarizing the results presented in chapter four and gives an outline for future work, which could shed more light on the performance of GPU and CPU clusters for training deep neural networks. The two main points of emphasis for future work presented are: scaling training to hundreds of CPU nodes (see e.g. You et al., 2017, for distributing training onto multiple CPUs) and comparing power consumption of CPU nodes against GPU nodes.

The best thing to say about Nita (2018) is how well the whole experience of the Student Cluster Competition is communicated by the author. Nita (2018) does not simply give a description of everything that is important for the benchmark, but gives a very conclusive and elaborate review of the whole competition.

Chapter four contains the benchmark and its results. It is a conclusive chapter and well annotated with useful references. The difficulties of the environment are clearly described. There were failures and the benchmarks on both the cluster of Team EPCC and Cirrus are not optimal. These difficulties and failures are openly discussed and the presented results are sensible, except the comparison between the Team EPCC cluster and Cirrus. Linear scaling over multiple Cirrus nodes is not probable. This is especially important, because the scaling is not properly discussed. There can be no confidence in the assumption of linear scaling and none is given by Nita (2018). That it will take 21 Cirrus nodes to equal the throughput of a single V100 is very questionable.

Overall, Nita (2018) describes the context and problem scope well. Its only problem is that it does not make it easy for the reader to see that. The layout can only be described as cluttered and a few passages and paragraphs are simply unnecessary. For example, Chapter 3.4—describing the TFRecord format for the image processing—does not belong in the chapter that concerns itself with the Student Cluster Competition,

if it belongs into the dissertation at all. While the chosen data format could be seen as an optimization and therefore should be described in chapter four, problems during the download before the competition are irrelevant to the benchmark. Also chapter three and two could be swapped, which would have made for an easier introduction to the dissertation, where one is first confronted by the whole context and rather gentle chapter about the Student Cluster Competition, instead of by a description of convolutional neural networks and ImageNet. The first paragraph of the introduction to the dissertation is just inappropriate and seems to pay homage to Goodfellow et al. (2016), which also starts with a similar introduction about Greek mythology and the craving of humankind for thinking machines. This is absolutely irrelevant for both: benchmarking deep neural networks and deep learning in general.

A second point of critique could be the absence of a chapter on related work, which gives a rough overview of other, comparable benchmarks. Some other benchmarks, like You et al. (2017), are referenced in the dissertation, but the reader does not get a general overview over what else is out there. Comparing Nita's results on the Team EPCC cluster with already existing benchmarks would have been better than comparing it against Cirrus, which did not even work properly.

Nita (2018) and "Deep Learning on SpiNNaker" overlap in their problem scope, in the sense that both concern itself with benchmarking the training speed of deep neural networks. Both focusing on image recognition based on the ImageNet dataset. While probably not directly important for the benchmark conducted as part of "Deep Learning on SpiNNaker", since Nita (2018) benchmarks the VGG16 model and "Deep Learning on SpiNNaker" will focus on ResNet-50 (see Section 2), Nita (2018) still provided a great first point of reference for the literature review done so far.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: training resnet-50 on imagenet in 15 minutes. *CoRR*, abs/1711.04325, 2017. URL `http://arxiv.org/abs/1711.04325`.

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. AI and Compute. `https://openai.com/blog/ai-and-compute/`, 2019.

Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`, 2020.

Livia Borghese and Tara John. Coronavirus cases soar in Italy as authorities scramble to find patient zero, 2020. URL `https://edition.cnn.com/2020/02/23/europe/italy-novel-coronavirus-spike-intl/index.html`.

Andrew P. Davison, Daniel Brüderle, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. doi: 10.3389/neuro.11.011.2008 pynn: a common interface for neuronal network simulators, 2009.

EPCC. Cirrus, 2020. URL https://www.cirrus.ac.uk.

S.B Furber, Steve Temple, and Andrew Brown. On-chip and inter-chip networks for modelling large-scale neural systems. In *Proceedings - IEEE International Symposium on Circuits and Systems*, pages 1945–1948, 2006a. ISBN 0780393902.

Steve Furber and Steve Temple. Neural systems engineering. *Journal of the Royal Society, Interface / the Royal Society*, 4:193–206, 05 2007. doi: 10.1098/rsif.2006.0177.

Steve Furber, Steve Temple, and Andrew Brown. High-performance computing for systems of spiking neurons. 2, 01 2006b.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018. URL http://arxiv.org/abs/1807.11205.

Norm Jouppi. Google supercharges machine learning tasks with TPU custom chip, 2016. URL https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip.

Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017. URL http://arxiv.org/abs/1704.04760.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(97)00011-7. URL http://www.sciencedirect.com/science/article/pii/S0893608097000117.

Mario Antonioletti and Alan Stokes. A Tensorflow Backend to SpiNNaker, 2019. URL https://www.wiki.ed.ac.uk/display/hpcdis/A+TensorFlow+BackEnd+to+SpiNNaker.

Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S. Vetter. NVIDIA tensor core programmability, performance & precision. *CoRR*, abs/1803.04014, 2018. URL http://arxiv.org/abs/1803.04014.

Henry Markram. The blue brain project. *Nature Reviews Neuroscience*, 7:153–160, 2006.

Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. Mlperf training benchmark, 2019.

Carver Mead. Analog vlsi and neural systems. 1989.

Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U.-Chupala, Yoshiki Tanaka, and Yuichi Kageyama. Imagenet/resnet-50 training in 224 seconds. *CoRR*, abs/1811.05233, 2018. URL http://arxiv.org/abs/1811.05233.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):3941, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL https://doi.org/10.1145/219717.219748.

Sparsh Mittal and Shraiysh Vaishay. A survey of techniques for optimizing deep learning on gpus. *Journal of Systems Architecture*, 08 2019. doi: 10.1016/j.sysarc.2019.101635.

Spyro Nita. Deep learning performance on different architectures. 2018. URL https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2017-2018/Spyro_Nita-dissertation-spyro-nita.pdf.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11, 12 2017. doi: 10.3389/fnins.2017.00682.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 09 2014.

Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489, 2017. URL http://arxiv.org/abs/1711.00489.

SpiNNaker Team. SpiNNaker Project, 2020. URL http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/.

Zak Stone. Cloud TPU Pods break AI training records, 2019. URL https://cloud.google.com/blog/products/ai-machine-learning/cloud-tpu-pods-break-ai-training-records.

Tensorflow Team. Developing a new backend for XLA, 2020. URL https://www.tensorflow.org/xla/developing_new_backend.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes, 2017.

Appendices

A Photos of SpiNNaker

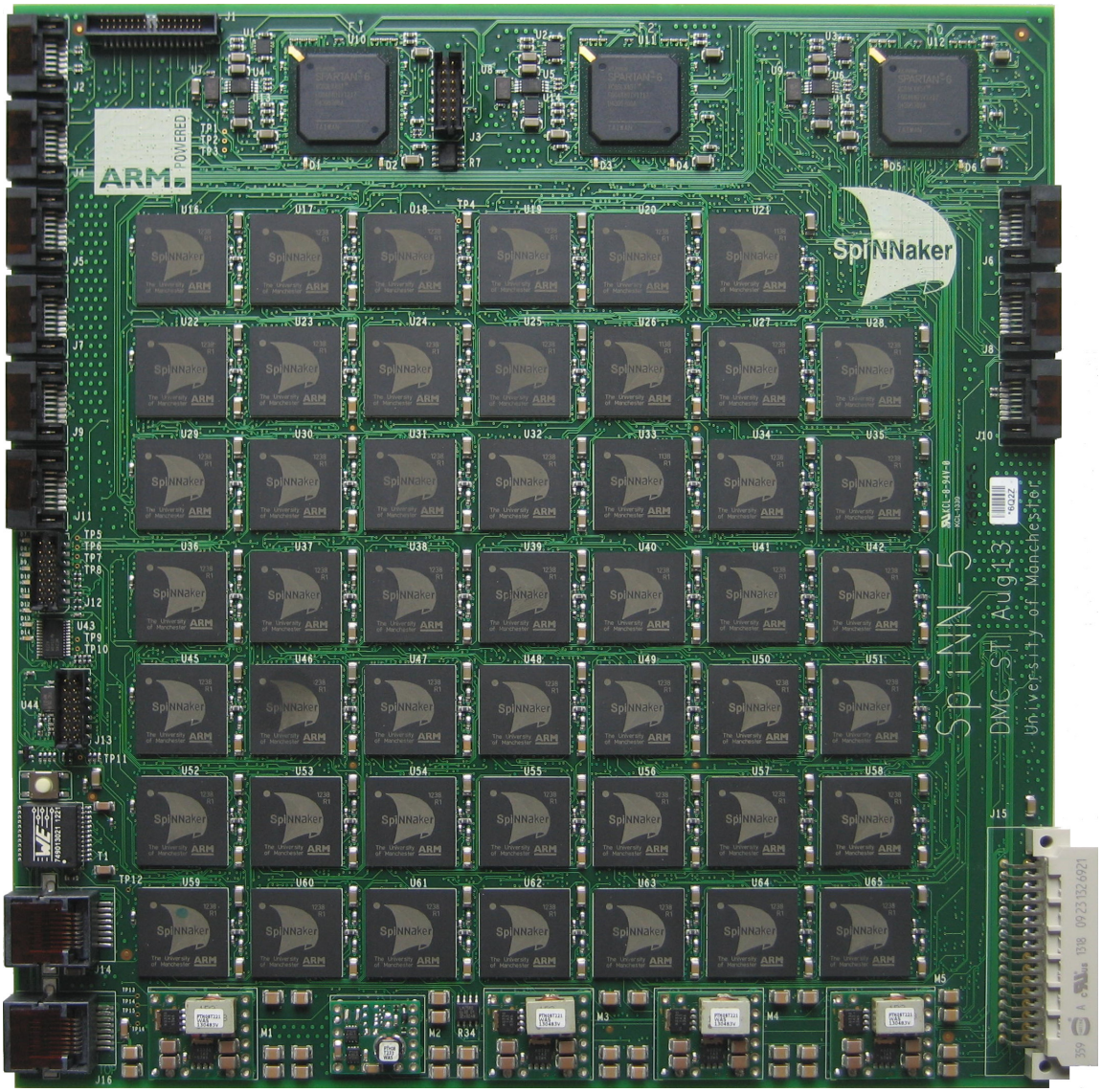

Figure 1: A single SpiNNaker board with 48 SpiNNaker chips. Each chip has 18 cores.

Figure 2: The one million core machine in Manchester. It is capable of running up to one billion spiking neurons in real-time.