2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

swFLOW: A Dataflow Deep Learning Framework on Sunway TaihuLight Supercomputer

Han Lin*, Zeng Lin*, Jose Monsalve Diaz†, Mingfan Li‡, Hong An§ and Guang R. Gao¶
*‡§School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
†¶University of Delaware, Delaware, USA
Email: *{linhan09, linzeng}@mail.ustc.edu.cn, †josem@udel.edu, ‡mingfan@mail.ustc.edu.cn,
§han@ustc.edu.cn, ¶ggao@capsl.udel.edu

***Abstract*—Deep learning technology is widely used in many modern fields and a number of deep learning models and software frameworks have been proposed. However, it is still very difficult to process deep learning tasks efficiently on traditional high performance computing (HPC) systems with specialized architectures such as Sunway TaihuLight. In this paper, we propose swFLOW: a TensorFlow-based dataflow deep learning framework on Sunway TaihuLight. Based on the performance analysis results on convolutional neural network (CNN), we optimize the convolution layer, reduce the data layout transpose operation and get 10.42x speedup compared to single management processing element (MPE) version. As for distributed training, we use elastic averaging stochastic gradient descent (EASGD) algorithm to reduce communication and use data prefetch to avoid data fetch being a performance bottleneck. On 512 processes, we get a parallel efficiency of 81.01% with communication period $\tau = 8$. Limited by the maximal executable batch size, the current performance of swFLOW is far from optimal. It is very necessary to further optimize using technology like remote direct memory access (RDMA) and model parallelism.**

***Index Terms*—deep learning, high performance computing, convolutional neural networks**

I. Introduction

Artificial intelligence (AI) technology based on deep learning is widely used in fields like image recognition, speech recognition, natural language generation and self-driving cars. Convolutional neural network (CNN) is one of the most popular deep learning models. Correspondingly, a number of deep learning software frameworks have been proposed, such as Caffe [1], MXNet [2] and TensorFlow [3].

Sunway TaihuLight [4] was released in 2016. It was the fastest supercomputer in the world for two consecutive years following its release. It reached a theoretical peak performance of 125.44 PFlops, and a Linpack performance of 93.02 PFlops. SW26010, the 260-core processor used in Sunway TaihuLight, achieves very high energy efficiency by using technologies like vectorization and collaborative computing. Sunway TaihuLight is composed of 40 cabinets. Each cabinet contains 4 super nodes and each super node contains 256 computing nodes. Each computing node includes 32 GB of memory and one SW26010 processor with 260 cores.

Traditional high performance computing (HPC) mainly focuses on scientific computing. Many supercomputers like Sunway TaihuLight are not designed for deep learning applications. But newer and more powerful supercomputers such as Summit and Sierra are equipped with the latest generation of general purpose GPUs (GPGPU) which specialized hardware that provides an excellent solution for deep learning applications. Some researches have demonstrated the advantages of these supercomputers when combining HPC and deep learning together [5], [6]. It is foreseeable that the combination of deep learning and HPC will become the mainstream for a long period of time in the future.

However, on the Sunway TaihuLight supercomputer the work on deep learning is still very limited. Current work on swDNN [7] and swCaffee [8], [9], designed in combination with the Caffe framework [1], target CNNs that fit the architectural characteristics of the Sunway processor. Yet, projects considering more practical applications and support for other deep learning frameworks on the Sunway supercomputer are almost blank.

In this paper, we propose swFLOW: a TensorFlow-based dataflow deep learning framework on Sunway TaihuLight. Apart from basic porting of the Tensorflow base code, hot spots analysis and optimization are done based on one of the most popular CNNs, i.e. VGG-16 [10]. Following this analysis, we redesign an implement the convolution layer based on the already existing swDNN implementation. Apart from this, swFLOW also supports distributed deep learning including data parallelism and model parallelism. We mainly focus on the former in this paper, since it is more popular and in general more efficient. To this end, we reduced the amount of communication in distributed model training by using elastic averaging stochastic gradient descent (EASGD) algorithm. Data prefetching effectively reduces the pressure on large-scale training data reading. These measures have enabled our applications to achieve good performance in distributed training.

These experiences have a good impact on verifying the effectiveness of Sunway TaihuLight in real deep learning applications and distributed training.

The main contributions of this paper include:

- We deconstruct and re-design TensorFlow on Sunway TaihuLight and construct a software framework for deep learning: swFLOW. It provides a good case reference for deep learning software construction on similar systems.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00345

- Using CNN as a benchmark, we analyze the performance of CNN and optimize it accordingly. The focus is on redesigning and optimizing the convolution layer based on swDNN and implementing efficient data layout transpose according to swFLOW's data layout. The results demonstrate the effectiveness of Sunway architecture in actual CNN applications.
- We implement distributed deep learning models training based on MPI APIs using EASGD algorithm. Data prefetching is used to further improve training data reading and global communication. These methods provide a good reference for distributed deep learning models training in actual scenarios.

The remainder of this paper is organized as follows. Section II shows Sunway processor architecture. Section III describes swFLOW's design and optimization including convolution layer, data layout transpose and distributed deep learning by data parallelism. Section IV discusses related work including deep learning libraries on Sunway TaihuLight and large scale distributed deep neural network (DNN) training. Section V evaluates swFLOW and section VI presents conclusion and future work.

II. Sunway Processor Architecture

Sunway TaihuLight, one of the fastest supercomputer in the world, is equipped with SW26010 many-core processor. As showed in Fig. 1, a SW26010 processor consists of 4 core groups (CGs) interconnected by network-on-chip (NoC). Each CG has 65 cores: one management processing element (MPE) and 64 computing processing elements (CPEs) organized as an 8x8 mesh. Each CG has its own memory space and the 65 cores inside it share the space through the memory controller (MC) in the CG. The processor connects to other outside devices (PCIe devices, Ethernet devices, etc.) through a system interface (SI).

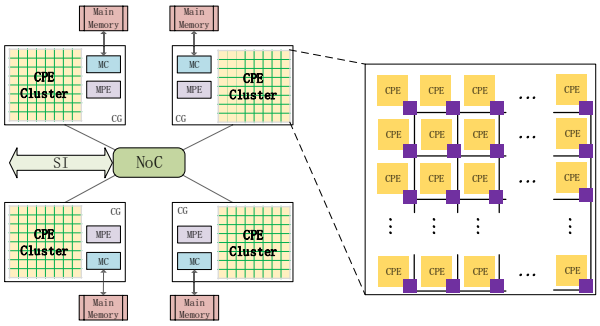

Fig. 1. SW26010 Architecture

The MPE and the CPE are both 64-bit RISC cores. MPE is a complete core which can run in both user and system modes. The feature of supporting interrupt functions, memory management, superscalar processing and out-of-order execution makes MPE an ideal core for handling management and communication. On the other side, the CPE is a reduced core that only has user mode. The design goal of CPE is to achieve the maximum aggregated computing power while minimizing the complexity of the micro-architecture. Both the MPE and CPE support 256-bit vector instructions. The whole processor has a peak performance of 3.06 TFlops in double precision. Since MPE has two float pipelines while CPE only has one, with a frequency of 1.45GHz, the 4 MPEs contribute 3.03% (2/66) to the whole float performance while the 256 CPEs contribute 96.97% (64/66).

As for memory hierarchy, each MPE has a 32 KB L1 instruction cache, a 32 KB L1 data cache and a 256 KB L2 cache for both data and instruction. Each CPE has a 16 KB L1 instruction cache and a 64 KB user-controlled scratch pad memory (SPM) (also known as Local Directive Memory, LDM). The four DDR3 memory controllers of a SW26010 processor provide a memory bandwidth of 136.51 GB/s. As mentioned above, all cores inside one CG share the memory space connected by the memory controller. On Sunway TaihuLight, every MC is connected to 8GB DDR3 memory. However, memory bandwidth between CPE and main memory is as low as 8 GB/s, which could noticeably impact performance of an application.

Inside the 8x8 CPEs mesh, there are: a control network, a data transfer network, 8 row communication buses and 8 column communication buses. The row and column communication buses enable fast register communication among CPEs, providing a very important data sharing capability inside the core group.

The basic software for SW26010 many-core processor includes basic compiler components (C/C++ and Fortran compilers, C++ is not supported on CPEs), an automatic vectorization tool and basic math libraries. For parallel programming, MPI is available across different computer nodes and different CGs. Other popular share-memory parallel programming frameworks include OpenMP, OpenACC and Pthreads are also supported. Additionally, on CPEs a pthread-like multi-thread library called acceleration thread library (athread) is provided.

III. swFLOW: Redesign and Optimize TensorFlow on Sunway TaihuLight

TensorFlow is a second generation machine learning system that operates at large scale and in heterogeneous environments. It is based on Google's first generation machine learning system DistBelief [11] and open sourced in the late of 2015 by Google. TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, GPGPUs, and custom-designed application specific integrated circuit (ASIC) known as Tensor Processing Unit (TPU) [12]. This architecture gives flexibility to the application developer: whereas in previous "parameter server" designs the management of shared state is built into the system, TensorFlow enables developers to experiment with novel optimizations and training algorithms. TensorFlow supports a variety of applications, with a focus on

training and inference of deep neural networks. TensorFlow has been widely used in production and research. It plays a very important role in the deep learning domain.

A. Software Architecture

Fig. 2 shows the architecture of TensorFlow: a C/C++ API separates user level code in different languages from the core runtime. The core TensorFlow library is implemented in C++ for performance and portability.

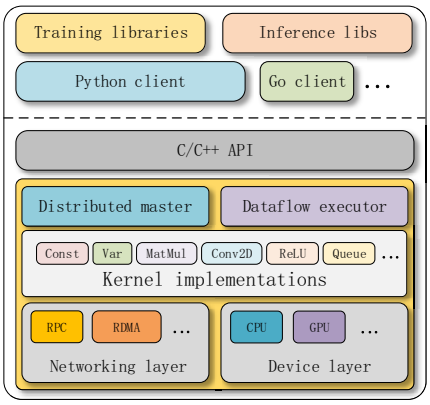

Fig. 2. TensorFlow Software Architecture [13]

The upper user level code includes different language clients and some high-level training and inference libraries. Users can conveniently build neural network models through these APIs. The core library of TensorFlow consists many modules such as kernel implementation, dataflow executors, local and distributed runtime implementations. The runtime contains over 200 standard operations, including mathematical, array manipulation, control flow and state management operations. Many of the operation kernels are implemented using Eigen::Tensor [14], which uses C++ templates to generate efficient parallel code for multicore CPUs and GPUs. On CPU, the calculation is based on Eigen inner kernels while on GPU cuDNN [15] and cuBLAS [16] are used for higher efficiency.

On Sunway TaihuLight, TensorFlow is deconstructed and reorganized in order to cooperate with the current software ecosystem. We separate the front end (the user level code) from TensorFlow and only keep the core library. Neural network models can be constructed through the provided C++ API or be imported from protobuf files exported from Python APIs. Through the latter way, all existing or pre-trained models can be loaded conveniently.

B. Optimization: Convolution Layer and Transpose

The VGG Network architecture [10] is used for uni-CG performance evaluation and profiling. VGG is a very popular CNN architecture which has been successfully applied to tasks like face recognition, semantic segmentation and visual tracking [17]–[20]. Compared with the AlexNet [21] framework, it replaces large kernel-sized filters (11×11 and 5×5 in the first and second convolutional layer respectively) with multiple 3×3 kernel-sized filters one after another. Such scheme with multiple non-linear layers deepens the network which enables it to learn more complex features at a lower cost. The thorough evaluation of VGG characterized by very simple convolution filters shows a significant improvement over AlexNet by pushing the depth to 16-19 weight layers.

In order to observe the behaviors of CNNs, we profiled VGG-16 on a single CG. Table I shows the performance analysis results of the VGG-16 architecture on a single CG on Sunway TaihuLight.

The profile result highlights that the hot spot is located in the convolution layer used during backward and forward propagation. This involves three functions: two for backward calculation and one for forward calculation. The backward functions are not separated in Table I because the two have partial common subfunctions calling which makes it difficult to separate them clearly. The running time of these hot spots accounts for 91.79% of the total execution time.

Due to lack of mature memory analysis tools, we did not analyze memory consumption on Sunway TaihuLight. Instead, we did memory analysis on an Intel platform using the valgrind [22], [23] toolkit. With a batch size of 64, the total memory heap consumption is over 5.7 GB. The value goes up to 9.4 GB when the batch size grows to 128. This estimate for memory usage is optimistic, since not all the memory operations are monitored.

swDNN [7] provides a convolution layer kernel implementation which can use all the computing resources of SW26010 processor efficiently. In order to get higher efficiency, it uses architecture-oriented optimization methods including double buffering, register communication, instruction reordering, etc. Such implementation is used in the customized swCaffe [9] framework (an adapted version of the Caffe [1] framework) showing distinguished performance improvements.

However, since the design and implementation of TensorFlow and Caffe are total different, merely integrating swDNN with TensorFlow can not achieve high efficiency out of the box. We redesigned swDNN according to TensorFlow data layout and real deep learning application practices.

The biggest limitation of swDNN comes from its strict boundary restrictions, especially regarding the allowed batch size which has to be no less than 128 when calling swDNN's APIs in the convolution layer. As previously mentioned, when running VGG-16 on intel platforms with a batch size of 64 the total allocated heap is 5.7 GB. However, when running on the Sunway TaihuLight system 5.7 GB would be overly optimistic. In addition to the memory used by VGG-16, extra memory has to be allocated to transpose the swFLOW data layout into the swDNN data layout before calling the swDNN API. Thus, the memory consumption of the convolution layer alone is doubled, resulting in over 1.5 GB of extra memory requirements for a batch size of 64. Therefore, a total of 7.2 GB for the heap would be required to run the VGG-16 model. As a result the 7000 MB available per CG are exceeded resulting in an out of memory (OOM) exception at runtime.

Based on the above observations, the goal of redesigning

TABLE I
THE PROFILE RESULTS OF VGG-16 ON A CG

Function	Description	Time (%)	Time (s)
tensorflow::Conv2DCustomBackpropInputOp, tensorflow::Conv2DCustomBackpropFilterOp	Convolution: Backward	62.16	2141.66
tensorflow::Conv2DOp	Convolution: Forward	29.63	1020.92
tensorflow::MatMulOp	Fully Connected Layer	3.35	115.5
tensorflow::DirectSession::Run	The main running loop of session.	1.87	64.37
Eigen::ThreadPoolDevice::parallelFor		0.99	34
memset	Memory operation	0.52	17.78
tensorflow::ShapeRefiner::ShapeRefiner		0.45	15.37
memcpy	Memory copy	0.40	13.78
Total		99.37	3423.38

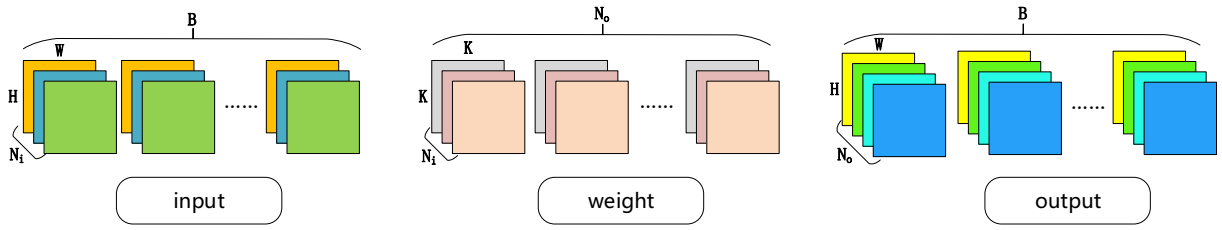

Fig. 3. A 2D Convolution Layer

the convolution layer of swDNN is to minimize the batch size limitation while ensuring computation efficiency.

The forward pass of a 2-dimension (2D) convolution layer with the SAME padding, as shown in Fig. 3, has an input tensor of a size $B \times N_i \times H \times W$, a filter tensor of a size $N_i \times N_o \times K \times K$ and an output tensor of a size $B \times N_o \times H \times W$ where B is the batch size, N_i is the input channels, N_o is the output channels, $H \times W$ is the size of every image and $K \times K$ is the filter size.

The convolution operation is implemented using a specialized general matrix multiplications (gemm) on every pixel of the $H \times W$ image. Overall, an input matrix of size $B \times Ni$ is multiplied by a filter matrix of size $Ni \times No$ to get an output matrix of size $B \times No$. In TaihuLight, the gemm operation is divided into 8×8 blocks and distributed across the 64 CPEs. Inside each CPE a block is subdivided into 4×4 elements, where each element is a vector containing 4 floating point numbers (only in the B dimension). Consequently, the batch size needs to be 128 elements ($8 \times 4 \times 4$) or more. In order to break this restriction, we reduced the small 4×4 blocks into 4×1 blocks, and rearranged the outer blocks by splicing 8 batches into a single big block. Through these measures the batch size limitation has been reduced from 128 to 4. For N_i and N_o, without the limitation of vector length, the limitation is 32.

The backward pass has a similar behavior except that in the input backpropagation the position of N_i and N_o are exchanged and in the filter backpropagation the position of B and N_i are exchanged. Hence, in the filter backpropagation the above optimization does not work. So we assign the work of all the 64 CPEs to a single column of CPEs, thus reducing the batch size restriction from 32 to 4.

As mentioned above, the data layout in swFLOW and convolution kernel are different, hence an extra transpose operation is required. We implement the transpose operations for both core types (i.e. CPE and MPE), and dynamically select the used version at runtime based on the amount of data required. The MPE transpose is faster for small data since CPEs transpose requires startup overhead and the overall faster memory bandwidth of CPEs ensures CPEs transpose faster for big data.

The evaluation results are shown in section V.

C. Distributed swFLOW: data parallelism and model parallelism

In distributed deep learning, there are roughly two parallel training methods: data parallelism and model parallelism. They distinguish by how the model is deployed on different machines. As Fig. 4 shows, in data parallelism every machine owns an independent, complete copy of the model while in model parallelism different parts of the model are distributed on different machines. Both of these methods are designed for better efficiency and maximizing the use of computing resources. However, model parallelism is mainly used when the model is too large to be deployed in a single machine. In addition, model parallelism requires more complex communication patterns between the different compute nodes as well as more available inter-node communication programming interfaces in the deep learning framework such as gRPC [24] in TensorFlow. Although ideally, each part of the model can be pipelined to avoid idling the machine in model parallelism, it is still difficult to fully utilize resources in practice due to high latency communication and the special neural network topology in gradients calculation. Although swFLOW can

support model parallelism through extended interfaces such as gRPC, our current work mainly focuses on data parallelism which is more popular.

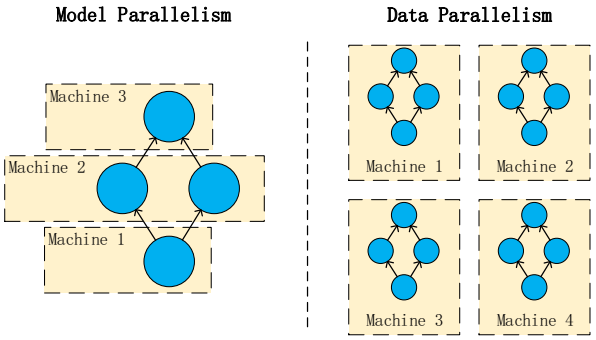

Fig. 4. Model Parallelism vs. Data Parallelism

In this paper, we use data parallelism for distributed training in order to get better parallelism efficiency. As shown in Fig. 5, there are 1 parameter server (PS) and n workers. Each worker has an independent model replica and works on its own data shard while the PS stores all the trainable variables (weights) in the model. In every training step, worker i get weights w from the PS, calculates gradients Δw and push Δw onto the PS to update w. However, this approach has the potential risk that high latency communication will become a bottleneck as the number of processes grows. To alleviate this problem, we use EASGD [25] to reduce the overall communication. The EASGD algorithm is described in Algorithm 1. Unlike the usual data parallelism implementation, weights are not updated in every training step. Instead, in EASGD each worker has a local copy of the training variables (x^i in Algorithm 1) that gets updated every cycle (line 7-8), while the PS stores another copy called the center variables ($\tilde{x}$ in Algorithm 1) that gets updated every τ cycles (line 4-5). Fluctuation in the values of the local variables hold by each worker is allowed, reducing the amount of communication and synchronization between workers and the PS. The frequency of communication is reduced to $1/\tau$, from every step to every τ steps. On Sunway TaihuLight, we use MPI communication API to achieve efficient communication between PS and local workers.

Data fetching is another problem in distributed training. The data sets for model training are usually very large. For example, the ImageNet [26] data set used in this paper contains over 1 million images with a size over 150 GB. Moreover, the DRAM memory size of a CG in Sunway TaihuLight is limited to 8 GB. This is the physical memory space of a process. So it is not practical to prefetch all the data into memory at once. The need of repeated reading further increases pressure in file system I/O. Therefore, a buffer scheme is used for data prefetching as shown in Fig. 6. In every worker process, a thread called producer is created for data prefetching and managing the data buffers. The producer runs asynchronously with the model training logic called consumer. A buffer has 3

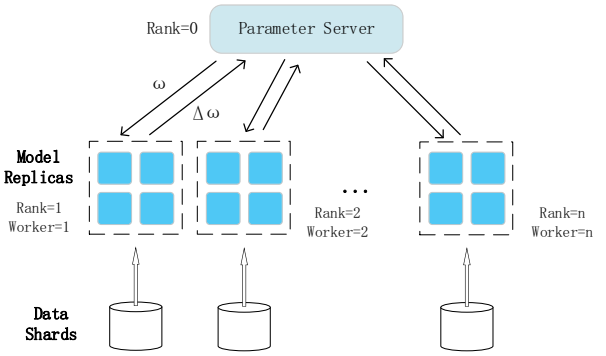

Fig. 5. Data Parallelism in Detail

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $t^i = 0$
1: **repeat**
2: $x \leftarrow x^i$
3: **if** τ divides t^i **then**
4: **a)** $x^i \leftarrow x^i - \alpha(x - \tilde{x})$
5: **b)** $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$
6: **end if**
7: $x^i \leftarrow x^i - \eta g^i_{t^i}(x)$
8: $t^i \leftarrow t^i + 1$
9: **until** forever

Algorithm 1. The EASGD Algorithm

mutable states: EMPTY, READY and EXCLUSIVE. Once the producer fills a buffer or a consumer consumes a buffer, the status is changed accordingly. Since there is only one producer and one consumer, no lock is required to avoid conflicts. This design minimizes the latency of file system I/O with limited memory consumption.

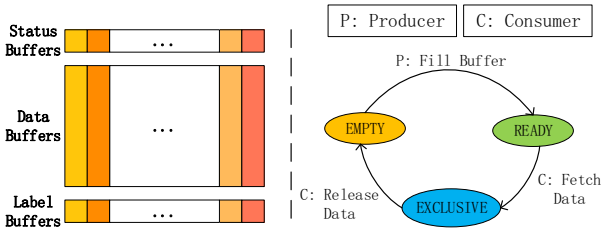

Fig. 6. The Buffer Scheme

IV. Related Work

A. Deep Learning and its Software Framework

Modern deep neural networks are built on top of the work by McCullogh and Pitt [27], and Rosenblatt (perceptron) [28]. However, these early models struggled as the network size increases, limiting their utility in the analysis of complex systems. More recently, work by Krizhevsky [21] opened the

flood gates for modern day Deep Learning, showing impressive performance on hard vision tasks using large supervised deep networks. This breakthrough was made possible in part by the rapid increase in computational power of modern computing systems. Since then, the complexity of tasks and the size of the networks have been growing steadily over time.

A number of software frameworks have been proposed for deep learning and related high performance calculations, such as Caffe [1], MXNet [2] and TensorFlow [3].

Yangqing et al. released Caffe [1] in 2014. This framework provides a simple and extensible deep learning toolbox for researchers in AI and computer vision (CV) fields. The good performance and the stable model architecture based on Google protobuf made it be quickly applied to deep learning applications in many fields such as speech recognition and robotics. Tianqi et al. proposed MXNet [2] in 2015. Automatic gradient calculation and computational graph construction based on symbolic calculation was implemented in MXNet. It also improved the efficiency of memory utilization by a series of optimization methods. TensorFlow [3] was first open sourced in late 2015. It constructs network architecture on the base of dataflow graph. The friendly network construction, data visualization and the integration with cloud computing systems made it a very popular framework in a short time in academia and industry.

B. Deep Learning on Sunway TaihuLight

As mentioned in section III-B, swDNN [7] is a library supporting efficient CNN implementation on Sunway TaihuLight. It was designed with the guide of a performance model which combined computing and memory resources together. With a series of optimization schemes including LDM-oriented algorithmic transformations, customized register communication schemes, as well as reordering of the instruction sequence, swDNN was capable of providing double-precision convolution performance around 1.6 TFlops on an SW26010 processor. swCaffe [9] combined swDNN with Caffe and got 4x speedup for the complete training process of the VGG-16 network compared to the unoptimized algorithm and framework. swCaffe on SW26010 has nearly half the performance of K40m in single precision and have 1.8x speedup over K40m in double precision. However, the strict limitation especially on batch size reduces the practicality of swDNN and swCaffe. swFLOW also use swDNN for batch size no less than 128, thus swFLOW and swCaffe have very close theoretical performance with the same batch size.

Jiacheng et al. [29] discussed how to port TensorFlow to new hardware including FPGA and Sunway TaihuLight. Like the way we present in section III-A, they also deconstruct TensorFlow and ported it as an static library which supports only C++ APIs. However, since they mainly focused on TensorFlow's behaviors on different types of hardware, the TensorFlow implementation in [29] was still very basic. They did not include swDNN in their package for acceleration, nor did they consider distributed training on Sunway TaihuLight. Fatal issues such as eliminating the overall communication conflicts and repeatedly fetching big training data sets have to be solved in large scale distributed training.

C. Large-Scale Deep Learning

There are more recent work on scaling deep learning up to large cluster and performance. Preferred Networks, Inc. demonstrated ResNet-50 [30] converging to 75% accuracy in 15 minutes using the ChainerMN [31] framework on 1024 NVIDIA Tesla P100 GPUs at a total global batch size of 32K for 90 epochs [32]. Jia et al. [33], concurrent with this work, demonstrated scaling to 2048 NVIDIA Tesla P40 GPUs at 64K batch size, achieving convergence in 6.6 minutes using TensorFlow.

Torsten et al. [5] realized a much larger scale deep learning training on Summit supercomputer. Motivated by the segmenting extreme weather patterns problem, they applied Tiramisu [34] and DeepLabv3+ [35] architecture to high resolution, multi-variate climate datasets. The Deeplabv3+ network was scaled up to 27360 V100 GPUs with a sustained throughput of 325.8 PF/s and a parallelism efficiency of 90.7% in single precision.

swFLOW has potential in large scale distributed deep neural network training. This is very important on supercomputers as large as Sunway TaihuLight. But swFLOW is not mature enough at present for distributed training at such a large scale. This is one of swFLOW's future work directions. The peak performance of SW26010 processor in double precision is 0.43 times that of NVIDIA V100 GPU (3.06 TFlops vs. 7 TFlops). With 40960 processors in total, Sunway TaihuLight has a peak performance of more than 17600 V100 GPUs. As for single precision, this number goes down to 8800. Once equipped with proper DNN libraries as well as mature software frameworks, Sunway TaihuLight has great potential in deep learning.

V. Evaluation

This section evaluates the performance of convolution layers and models training speed on swFLOW. Based on these results, further analysis has been conducted and presented.

A. Convolution Layer

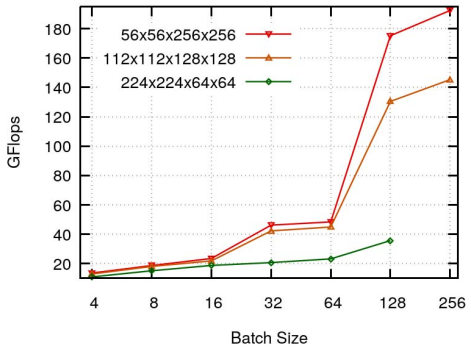

Fig. 7. Convolution Layer Performance, Label can be read as ($H \times W \times Ni \times No$).

Fig. 7 shows the performance of convolution layers in swFLOW with the change of batch size. Both the forward and backward routines are presented. Each curve represent a different layer. The four different parameters of each convolution layer are: height (H), width (W), input channels (N_i) and output channels (N_o) presented in that order in the label. All the 3 curves show a very normal tendency: the process speed grows as the batch size grows. The leap on 128 is because swFLOW calls swDNN directly with these parameters. The optimization introduced in section III-B results in additional overhead due to changes in the iteration process of the convolution layer, thus, resulting in a higher number of iteration steps. The convolution layer with parameters of $224 \times 224 \times 64 \times 64$ requires more memory and caused an out of memory error with batch size of 256.

B. Single Process

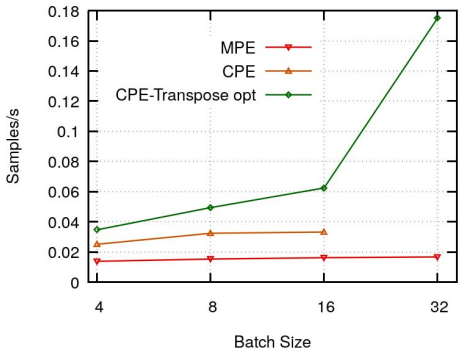

Fig. 8. Performance of Single Process

Fig. 8 presents the overall training performance of VGG-16 on a CG. Results labeled "MPE" used a single MPE. Compared to results labeled "CPE" that used the CPE cluster to accelerate the convolution layer, and the "CPE-Transpose opt" labeled results that remove the redundant transpose operation. The CPE-Transpose opt version is up to 10.42 times faster than MPE version. This demonstrates the acceleration improvements brought by the redesign of the convolution layer, as explained in section III-B. For batch size between 4 and 16, due to the limitation of SPM size, not all the convolution layers are offloaded onto CPE Cluster. The performance changes significantly on batch size 32 because all the convolution layers are accelerated by CPE Cluster in this case. The transpose optimization results in 59% performance improvement on average. Moreover, reducing memory consumption makes it possible to process larger batch size.

The results in section V-A shows that increasing batch size further can improve performance of convolution layers further. Apart from increasing physical memory size on Sunway TaihuLight, model parallelism may be a good way to accelerate calculation even further: by splitting DNNs and putting different parts on different processors, the upper bound of batch size can be increased.

C. Distributed Training

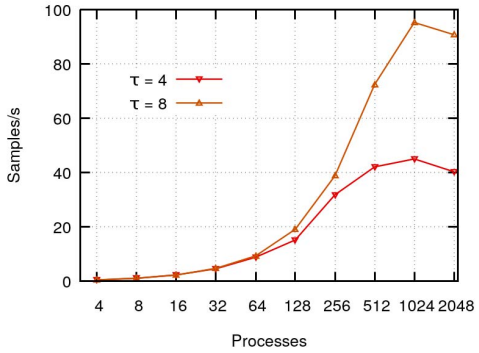

Fig. 9. Performance of EASGD

Fig. 9 shows the performance of distributed VGG-16 as the number of processes grows with different τ. The overall training speed increases steady as the number of processes grows. By using the EASGD algorithm, it is possible to reduce communication overhead, hence improve performance for swFLOW distributed model training. The parallel efficiency of 512 processes reaches 47.09% for $\tau = 4$ and 81.01% for $\tau = 8$. For 1024 processes the parallel efficiency is 25.15% and 53.21% respectively — communication plays a more important role as the scale grows. Both curves have a nearly linear range (parallel efficiency >80%). For $\tau = 4$, it is [4, 64] and for $\tau = 8$ it is [4, 512]. Beyond those intervals, the overall performance improvement rate has dropped sharply. With 2048 processes, both curves even show worse performance.

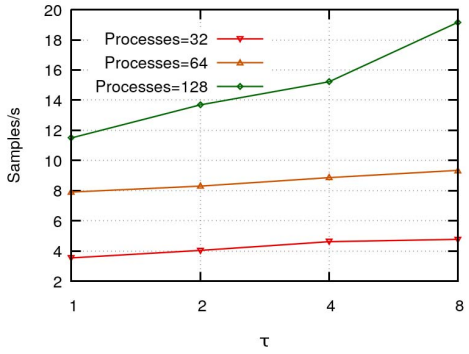

Fig. 10. Performance on Different τ

Fig. 10 displays the effect of τ on VGG-16's performance with batch size 32. When $\tau = 1$, the average training speed of every process is 48.26% slower than the serial performance shown in section V-B with 128 processes. With the grow of τ, the overhead caused by global communication decreases and the training speed increases. The results also reveal a trend: the larger the number of processes, the more sensitive the performance is to the change of τ. In short, when the

number of processes is small, the change of τ has little impact on overall performance. As the number of processes grows, communication becomes an increasingly important factor to performance and therefore the importance of τ is becoming more and more prominent. The value of τ should be selected accordingly in relation to the size of the training model. If the value of τ is too small, communication is not effectively reduced. On the other hand, if the value of τ is too large, the overall convergence speed of the model will be slower. In larger scale distributed training, a larger τ is preferred.

VI. Conclusion and Future Work

In this paper we present swFLOW: a redesigned TensorFlow for Sunway TaihuLight supercomputer. After optimizing the convolution layer and the data layout transpose operation, swFLOW achieves a speedup of up to 10.42x in comparison with the single MPE version. The distributed training on 512 processes reaches a parallel efficiency of 81.01%.

On Sunway TaihuLight, swFLOW is one of the earliest available software frameworks supporting distributed deep learning. This is critical for Sunway TaihuLight to develop a mature deep learning ecosystem. Moreover, the development of deep learning software can also motivate the future development of Sunway hardware architectures. Even though TensorFlow itself provides a portable approach, the specialized many-core architecture of the SW26010 processor requires a lot of analysis and optimization to achieve an acceptable performance. The design and implementation of swFLOW reveals a general methodology of porting deep learning frameworks onto supercomputers with specific architectures.

Performance of swFLOW shown in this paper is limited by the size of physical memory in the Sunway architecture, which restrict the maximum executable batch size. Apart from increasing physical memory size, model parallelism may be a good way to increase the upper bound of batch size. In distributed training, communication across multiple nodes limits performance scaling in addition to the aforementioned single core performance. Further optimizations such as using remote direct memory access (RDMA) is critical for improving overall performance.

Our future work is focused on two different aspects: 1) exploring further optimizations and implementations of our model such as model parallelism, instruction reordering, use of RDMA engines and improvements on the implementation on specific batch sizes. And 2) target other applications and neural networks. In particular large scale distributed training on problems that combine scientific computation and deep learning.

Acknowledgment

The work is supported by the National Key Research and Development Program of China (Grant No. 2016YFB1000403) and the Fundamental Research Funds for the Central Universities of China.

References

[1] 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 *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[2] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[4] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao *et al.*, "The sunway taihulight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, p. 072001, 2016.

[5] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica *et al.*, "Exascale deep learning for climate analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 2018, p. 51.

[6] R. M. Patton, J. T. Johnston, S. R. Young, C. D. Schuman, D. D. March, T. E. Potok, D. C. Rose, S.-H. Lim, T. P. Karnowski, M. A. Ziatdinov *et al.*, "167-pflops deep learning for electron microscopy: from learning physics to atomic manipulation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 2018, p. 50.

[7] J. Fang, H. Fu, W. Zhao, B. Chen, W. Zheng, and G. Yang, "swdnn: A library for accelerating deep learning applications on sunway taihulight," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 615–624.

[8] W. Zhao, H. Fu, J. Fang, W. Zheng, L. Gan, and G. Yang, "Optimizing convolutional neural networks on the sunway taihulight supercomputer," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 1, p. 13, 2018.

[9] L. Li, J. Fang, H. Fu, J. Jiang, W. Zhao, C. He, X. You, and G. Yang, "swcaffe: A parallel framework for accelerating deep learning applications on sunway taihulight," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 413–422.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 1–12.

[13] T. Developers, "Tensorflow architecture," *Web page at https://www.tensorflow.org/guide/extend/architecture*, 2019.

[14] E. Developers, "Eigen," *Web page at http://eigen.tuxfamily.org*, 2019.

[15] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[16] C. Nvidia, "Cublas library," *NVIDIA Corporation, Santa Clara, California*, vol. 15, no. 27, p. 31, 2008.

[17] J. Dai, K. He, and J. Sun, "Convolutional feature masking for joint object and stuff segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3992–4000.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[19] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Hierarchical convolutional features for visual tracking," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3074–3082.

[20] Y. Sun, D. Liang, X. Wang, and X. Tang, "DeepID3: Face recognition with very deep neural networks," *arXiv preprint arXiv:1502.00873*, 2015.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
[22] N. Nethercote and J. Seward, "How to shadow every byte of memory used by a program," in *Proceedings of the 3rd international conference on Virtual execution environments*. ACM, 2007, pp. 65–74.
[23] V. Developers, "Valgrind," *Web page at http://valgrind. org*, 2019.
[24] g. Developers, "grpc," *Web page at https://grpc.io*, 2019.
[25] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems*, 2015, pp. 685–693.
[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.
[27] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
[28] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.
[29] J. Zhao, Y. Chang, D. Li, C. Xia, H. Cui, K. Zhang, and X. Feng, "On retargeting the ai programming framework to new hardwares," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2018, pp. 39–51.
[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[31] T. Akiba, K. Fukuda, and S. Suzuki, "Chainermn: scalable distributed deep learning framework," *arXiv preprint arXiv:1710.11351*, 2017.
[32] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes," *arXiv preprint arXiv:1711.04325*, 2017.
[33] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *arXiv preprint arXiv:1807.11205*, 2018.
[34] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE, 2017, pp. 1175–1183.
[35] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *arXiv preprint arXiv:1802.02611*, 2018.