Scaling the Training of Recurrent Neural Networks on Sunway TaihuLight Supercomputer

Ouyi Li[1,3], Wenlai Zhao[2,3(✉)], Xuancheng Huang[2], Yushu Chen[3], Lin Gan[2,3], Hongkun Yu[2,3], Jiacheng Zhang[2], Yang Liu[2], Haohuan Fu[1,3], and Guangwen Yang[2,3]

[1] Ministry of Education Key Laboratory for Earth System Modeling, Department of Earth System Science, Tsinghua University, Beijing, China
loy16@mails.tsinghua.edu.cn
[2] Department of Computer Science and Technology, Tsinghua University, Beijing, China
zhaowenlai@tsinghua.edu.cn
[3] National Supercomputing Center in Wuxi, Wuxi, Jiangsu, China

Abstract. The recurrent neural network (RNN) models require longer training time with larger datasets and bigger number of parameters. Distributed training with large mini-batch size is a potential solution to accelerate the whole training process. This paper proposes a framework for large-scale training RNN/LSTM on the Sunway TaihuLight (SW) supercomputer. We take series of architecture-oriented optimizations for the memory-intensive kernels in RNN models to improve the computing performance. The lazy communication scheme with improved communication implementation and the distributed training and testing scheme are proposed to achieve high scalability for distributed training. Furthermore, we explore the training algorithm with large mini-batch size, in order to improve convergence speed without losing accuracy. The framework supports training RNN models with large size of parameters with at most 800 training nodes. The evaluation results show that, compared to training with single computing node, training based on proposed framework can achieve a 100-fold convergence rate with 8,000 mini-batch size.

Keywords: Neural machine translation · Recurrent neural networks · Large-scale training · Many-core architecture · Sunway TaihuLight supercomputer

1 Introduction

Deep learning has already proven its efficiency in many different tasks. Recurrent neural network (RNN) [13] plays an important role in deep learning. Neural machine translation (NMT) is one of the most successful application examples of RNN. Training RNN models in NMT task takes a lot of time.

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11536, pp. 427–440, 2019.
https://doi.org/10.1007/978-3-030-22734-0_31

It is proved that larger datasets and bigger models lead to improvements in accuracy for many deep learning tasks. In NMT, as the number of parameters increases, the memory space required becomes larger. Taking a specific parameter setting in [2] as an example, attention based long short-term memory (LSTM) [7] with 1000 hidden units, 620 dimensional word embeddings and 75 words in a sentence with mini-batch size 40 needs 30 GB memory.

With the rapid development of deep learning, the size of dataset in related tasks gradually becomes larger. It takes more time to traverse the whole training dataset as one epoch during training models. For example, the size of NMT dataset grows from 1,200,000 pairs of sentences in source language (English) and target language (Chinese) in dataset NIST to 5,800,000 pairs of sentences in dataset WMT. In order to cope with the increasing size of the dataset, we need to ensure that the overall training time is controllable by reducing the time of traversing the dataset during training process.

Scaling the mini-batch size by synchronizing data parallel is a good solution. In some previous works, a large-scale training on distributed deep learning frameworks have been explored by [5,15]. They mainly focus on training CNN models on ImageNet [11]. Correspondingly, we explore the solution of large-scale training RNN/LSTM in NMT in this paper.

SW provides large number of computing nodes, we can explore training RNN/LSTM with large size of parameters with large mini-batch size by synchronizing data parallel. If we train attention based LSTM model with 2000 hidden units, 1440 dimensional word embeddings on one computing node, the allowed mini-batch size is 10. To explore training large models with large mini-batch size, 800 training workers are needed to reach the mini-batch size 8,000.

We propose architecture-oriented optimizations and communication optimizations customized for RNN/LSTM. We propose a distributed RNN/LSTM training framework based on SunwayCaffe [8,16]. We explore convergence acceleration ratio of training process and generalization of models under large-scale training scenarios. Our specific contributions include:

- We propose architecture-oriented optimizations for the memory-intensive kernels in RNN/LSTM, fully exploring the parallelism in SW26010 many-core architecture and improving the usage of memory bandwidth.
- We propose improved MPI_Allreduce implementation and lazy communication scheme customized for RNN/LSTM, so as to achieve high communication efficiency and high scalability for P2P-based distributed training.
- We propose a customized distributed framework design for NMT, which assigns the computing nodes with different tasks (training, testing and evaluating). The proposed framework provides environment for frequently validating models with little test interval. Training in the framework can overlap all testing time.
- We provide discussions and analysis of the convergence acceleration ratio of large-scale training RNN/LSTM and the generalization of models. Furthermore, we show an empirical training strategy, as well as the evaluation results of RNN/LSTM on a large dataset with up to 800 training nodes.

The rest of the paper is organized as follow. In Sect. 2, we describe the background of this paper. In Sect. 3.1, we propose our architecture-oriented optimizations for the memory-intensive kernels in RNN/LSTM on SW26010 processor. In Sect. 3.2, we propose our communication optimizations including the improved MPI_Allreduce implementation and lazy communication scheme. In Sect. 3.3, we describe our distributed framework designed for NMT. In Sect. 4, we explore large-scale training RNN/LSTM and we provide discussions and analysis of the convergence acceleration ratio of large-scale training RNN/LSTM and the generalization of the models. In Sect. 5, we show evaluation of the optimizations in our framework, and the evaluation results of large RNN/LSTM. In Sect. 6, we conclude the report and discuss about the future work.

2 Background

2.1 RNN Model and NMT Task

RNN is a class of artificial neural network and add sequential information to the artificial neural network model. RNN models can use their hidden units to process sequences of inputs. A finite RNN is a directed acyclic graph that can be unrolled. When it is unrolled, each layer deals with one element in the sequence. LSTM cell is a kind of RNN cell, and it adds gates to basic RNN cell to regulate the flow of information into and out of the cell.

The past several years have witnessed the rapid development of NMT, which aims to model the translation process using neural networks in an end-to-end manner. Most of the proposed NMT models belong to a family of encoder-decoder. The encoder-decoder system consists of an encoder and a decoder for a language pairs. They are jointly trained to maximize the probability of a correct translation given a source sentence. All neurons in encoders share their weights and all neurons in decoders share their weights.

There are three encoder-decoder models mentioned in this paper, the first model is encoder-decoder (RNNencdec) proposed in [14], the second one is attention-based encoder-decoder (RNNsearch) proposed in [2], the third one is attention-based encoder-decoder with twice number of hidden units and dimensional word embeddings of RNNsearch (RNNsearch-H).

We evaluate models on English-Chinese translation. We have two training datasets, one small dataset consisting of 1.25M pairs of sentences and one large dataset consisting of 5.8M pairs of sentences. We use the NIST 2002 dataset as validation dataset for the first training dataset and WMT dataset as validation dataset for the second one.

Bilingual evaluation understudy (BLEU) is an algorithm for evaluating the quality of text translated by machine from one natural language to another. BLEU considers the quality to be the correspondence between a machine's output and that of a human. It is now the normal standard for evaluating machine translation results. It is involved in our framework.

2.2 System Setup

One SW26010 many-core processor is composed of four core-groups (CGs), each CG consists of 65 cores: one management processor element (MPE) and 64 computing processor elements (CPEs). 64 CPEs are organized as a CPE cluster. Within a cluster, CPEs are connected in an 8 by 8 mesh. The MPE and CPEs are all based on 64-bit RISC architecture, but have different duties. The MPE supports the complete interrupt functions, memory management, superscalar and out-of-order issue/execution. It is good at handling management, task schedule, and data communications. CPE is designed to maximize the aggregated computing and minimize the complexity of the micro-architecture. Each of CGs owns 8 GB of DDR3 memory, shared by MPE and CPE cluster through the Memory Controller (MC). So one node has 32 GB memory. The on-chip network (NoC) connect the MPE/CPE chip with System Interface (SI).

2.3 Current Situation and Related Work

As for now, most of state-of-art RNN/LSTM in different tasks are trained with small mini-batch size performed in single-worker multi-GPU mode. In [14], they train RNNencdec model with mini-batch size 128. The bi-LSTM model has 4 layers with 1000 hidden units, 620 dimensional word embeddings. In [2], they proposed model RNNsearch, the encoder and decoder of RNNsearch both have 1000 hidden units, with mini-batch size 80, the length of the training sentences is 50.

To explore training RNN/LSTM with larger number of parameters requiring more memory space, we choose to train models on SW. Architecture-oriented optimizations for the memory-intensive kernels in RNN/LSTM are needed to improve the usage of memory bandwidth and to improve the overall computing performance.

As the size of dataset increases, the time consumption of training RNN/LSTM in NMT increases. There are two training datasets mentioned in this paper. The time consumption of training on WMT dataset for one epoch is five times of that on NIST dataset. Multi-server distributed training is a solution. SW provides a large number of computing nodes, we can scale the mini-batch size in data parallelism method to meet our need.

The support for the existing distributed frameworks of parameter solver (PS) architecture [12] on SW is not satisfactory. In contrast, open-sourced Caffe [8] deep learning framework on a single computing node has been well supported on SW. Our previous work consists of optimized math library SWDNN and SunwayCaffe [4,16]. SWDNN mainly optimized the convolutional layer and pooling layer according to the architecture of SW. SunwayCaffe provides a basic distributed framework for training CNN models on SW. This paper proposes a framework based on SunwayCaffe. Because of recurrent structure of RNN/LSTM, the communication mechanism in training process are optimized to achieve high communication efficiency and scalability.

In terms of large-scale training neural networks, many researchers have explored ways to reduce the training time under the premise of ensuring the accuracy. [5] proposed a scheme of large-scale training CNN models. They scale mini-batch size to 8k in AlexNet and ResNet with decline in testing accuracy. [15] proposed a layer-wise adaptive optimization algorithm LARS. They scale the mini-batch size of training ResNet to 32k with no decline in accuracy. As aspect of RNN/LSTM in NMT task, few people explore large-scale training.

Therefore, in this paper, for large-scale training RNN/LSTM according to current situation, we propose architecture-oriented optimizations for the memory-intensive kernels in RNN/LSTM for improving the usage of memory bandwidth and improving the computing performance. We propose an efficient implementation of MPI_Allreduce and a lazy communication scheme for high communication efficiency and high scalability. We propose a customized distributed framework designed for overlapping the testing time. We explore large-scale training RNN/LSTM and provide discussions and analysis of convergence acceleration ratio and generalization of models. Finally we show the results of large-scale training RNNsearch-H with an empirical training strategy.

3 Optimizations

3.1 Architecture-Oriented Optimization

In training process of RNN/LSTM, GEMM (General Matrix Multiply) is the most computation intensive operation. GEMM performs in the computing of gates in LSTM neurons, logit layers and attention module. Optimization of GEMM on SW26010 many-core processor has been discussed in [9,16]. We apply the implementation in our framework directly.

Besides the computation intensive kernels, the optimizations of the memory intensive kernels are also very important from the perspective of the overall performance. Gradient-based optimization algorithms (e.g. square root operation) adopt element-wise vector operations, for which performance is limited by memory bound. These layers can be efficiently implemented by DMA for large continuous data blocks and perform computation in CPE cluster.

Particularly, two kernels are major considerations in this paper: the exponential layer and the softmax layer.

Exponential Layer Optimization. In neural networks, activation layers are used to perform a nonlinear transformation of data. In RNN/LSTM, activation layers are mainly sigmoid layers and tanH layers, which both utilize nonlinear property of exponential function. In addition to activation layers, exponential function is also in the implementation of softmax layer.

On SW, the underlying implementation of the exponential function is included in SW basic math library. the specific implementation of the exponential function is method of look-up table with interpolation. The look-up table of exponential function is stored in memory of MPE, both computing in MPE

and CPEs need to access data from main memory when exponential function is called. Data access from main memory takes more than 100 CPU cycles. Instead of SW basic math library, we implement the exponential function by Taylor Expansion using a small amount of LDM in CPEs. Our implementation avoids discrete data access from main memory. Efficiency of the operation is greatly improved while ensuring the precision.

Softmax Layer Optimization. In the decoding phase of NMT, models pass data through softmax layer at each time step, and the number of neurons in softmax layer is equal to the size of vocabulary, usually 50,000 or more. In ImageNet, the number of neurons in softmax layer is equal to the number of categories 1000. The categories of NMT is more than 50 times that of ImageNet, and the length of sentences in NMT is always 50 to 100. Therefore, in an training iteration, the amount of computation of softmax in NMT is 2500 to 5000 times that in ImageNet. The large number of neurons in softmax layer and the high occurrences make softmax layer a bottleneck in NMT when training on SW.

The softmax function is described as Eq. 1. K is the number of neurons in softmax layer. It maps the output value of j-th neuron to a new value in interval $(0, 1)$, which can be thought as the probability of the category represented by the output.

$$\alpha(x)_j = \frac{\exp(x_j)}{\sum_{k=1}^{K} \exp(x_k)} \qquad for\ j\ =\ 1\ to\ K \tag{1}$$

As the output of exponential function increases fast as x grows, the input of exponential function of each neuron in softmax layer must keep small while the

Algorithm 1. Implementation of softmax

Input

Vector<Data*> A; Batch size: B; Data count in one batch: N; Core group id: cg_id; CPE id: cpe_id;

output

Probability of each data of all batches: Vector<Data*> S

function PARALLEL_SOFTMAX(Vector<Data*> A, INT B, INT N)

    $Sync\_4cg()$;

    $start\_index = B/4 * cg\_id + B/(4 * 64) * cpe\_id$;

    $local\_count = B/(4 * 64)$;

    dma($local\_A, start\_index, local\_count$) $from\ A$;

    for each d in $local\_A_{cg\_id,cpe\_id}$ **do**

        $M = max(d)$;

        d = exp(d-M);

        $SUM = \sum d$;

        $local\_S = d \div SUM$;

    end for

return ;

end function

output of softmax layer unchanged. Max value of all neurons is subtracted from value of each neuron first. This trick can be applied to make sure that none of the exponentials overflows [8].

According to Algorithm 1, the implementation of softmax layer on SW26010 mainly contains four parts of computations. Two parts in Line 10 and Line 12 are both element-wise operation, and their implementations are mentioned previously. Other two parts of computations are MAX operation in Line 9 and SUM operation in Line 11. We do data parallelism in batch level, each CPE computes max (summation) of vector data in different mini-batches. For the same data, two operations in Line 10 and Line 11 can be completed after one DMA operation according to the index of CPE in CGs. Through the above implementations on SW, we can accelerate the training process on SW26010 processor.

3.2 Communication Optimization

Communication Architecture. Parameter Server (PS) and P2P communication are two mainstream communication architecture for distributed deep learning [12]. PS follows a client-server scheme and can be easily scaled up on a distributed cluster, tolerating the imbalanced performance, unstable network bandwidth and unexpectable faults of the workers. However, for a supercomputer system, the sustaining performance and the stability of the computing nodes can be guaranteed, as well as the network condition. Therefore, a P2P communication architecture is more suitable.

We adopt two optimization methods to further improve the communication efficiency, including an improved MPI_Allreduce design and a lazy communication strategy for distributed RNN/LSTM training.

Improved Allreduce. The network topology of SW is a two-level fat tree, which consists of an intra super-node level and an inter super-node level. At the intro super-node level, 256 computing nodes are fully connected via a customized super-node network switch. At the inter super-node level, a central switching network is designed for the communication between different super-nodes. Generally, the intro super-node communication has a higher bandwidth and a lower latency than the inter super-nodes communication.

To improve the overall communication efficiency, we implement a kind of hierarchical Allreduce. [6] An improved Allreduce design is proposed with four stages, which include: (1) an inter super-node reduce stage; (2) an intra super-node reduce stage; (3) an intra super-node broadcast stage; (4) and an inter super-node broadcast stage. Compared with the original MPI_Allreduce operation, the improved Allreduce contains as less inter super-node communication requests as possible.

Besides, in the improved Allreduce operation, the computation operations (usually SUM operation is used to aggregate gradients) is accelerated using the CPEs, while it is handled only by MPE in the standard MPI_Allreduce implementation. With the optimizations on both computation and communication,

the improved Allreduce operation can achieve about 20 times higher efficiency on average than the standard MPI_Allreduce operation.

Lazy Communication. In a training iteration, each layer invokes an Allreduce communication for the gradients, so that the number of communication requests is usually large for deep neural networks. In modern RNN/LSTM, the number of parameters in each RNN/LSTM layer is related to the length of the word vector. Usually if the size of hidden states is 1000, which is relatively large, there are about 1 million (1000×1000) parameters in a layer, and then the data size of the gradients involved in one Allreduce operation is about 4 MB. Hence we can see that, there are numerous communication requests with small data size, which is the main feature of the communication pattern in a distributed NMT training framework, and is not efficient in large-scale training tasks.

To address the above issue, we propose a lazy communication scheme in our framework. The basic design idea is that instead of executing the communication requests immediately, we *remember* them temporarily until the unsent data size is greater than a given $MAXSIZE$, which is set to 100 MB empirically in our framework.

The lazy communication design can merge multiple small-data-size communication into a large-data-size communication, which can improve the overall efficiency by lowering the launch cost and increasing the utilization rate of network bandwidth.

3.3 Framework Optimization

Considering the load balance and the overall training efficiency, we propose a new distributed framework design for large-scale training RNN/LSTM.

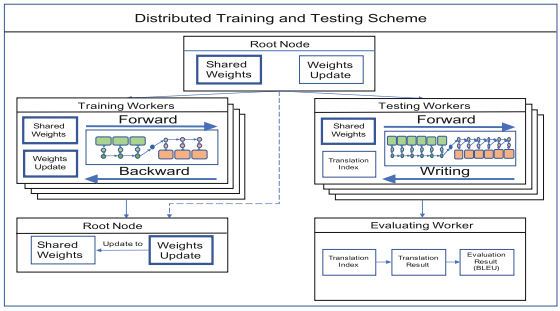

Fig. 1. The overview of distributed training and testing scheme

The training process in NMT contains three modules: training module, testing module and evaluating module. A training model and a testing model are involved in the training process. The model in training module and the model

in testing module share all parameters, but not absolutely the same. There is clear relationship between three modules. Evaluating module runs after testing module, training module and testing module run in parallel.

In fact, except sharing parameters at beginning of training iteration and testing iteration, training module and testing module don't rely on each other. We propose a distributed training and testing scheme as shown in Fig. 1. A testing interval is set that testing process is performed after every testing interval of training iterations.

In NMT, even after a few training iterations, BLEU value changes quite a lot. So frequent validation help find the highest BLEU value. The model with the highest BLEU value on validation dataset is always the model with the best generalization in theory. Distributed training and testing scheme leads to complete overlap of testing process and evaluating process, we can save extra testing time. The scheme greatly improves efficiency of entire training process and also provides an opportunity to find the model with the best generalization.

4 Convergence and Generalization

4.1 Model Convergence Optimization Algorithm

The stochastic gradient based optimization algorithm applied in our training experiments is Adaptive Moment Estimation (Adam). Adam applies momentum on a per-parameter basis and automatically adapts step size subject to a user-specified maximum learning rate [10]. Adam's convergence speed and generalization made it a popular choice for NMT [1,3].

Learning Rate. Under large-scale training scenarios, when RNNsearch model is trained with Adam with same mini-batch size (100 nodes, 8000 mini-batch size and 50 nodes, 4000 mini-batch size) on NIST dataset, the convergence speed at different learning rates are shown in Fig. 2(a) and (b). Under large-scale training scenarios, although the learning rate can be adaptively adjusted with the first moment and the second moment in Adam, setting a higher or lower learning rate will result in a slower convergence speed. When learning rate is set too high (0.003 or more), the model does not converge.

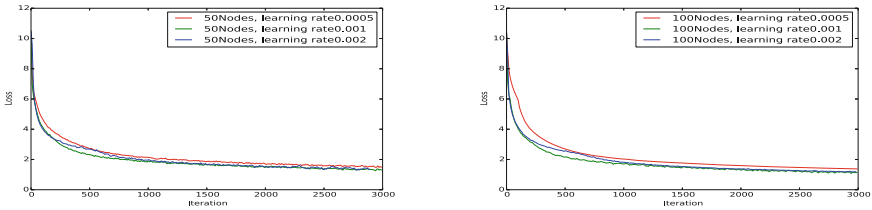

Fig. 2. Loss vs. learning rate

Parameter Setting. Under large-scale training scenarios, Adam is applied with momentum 0.9 and momentum2 0.999. The learning rate is 0.001. The length of sentences is set to 50 on dataset NIST and 75 on dataset WMT. When training models on single node, mini-batch size for RNNencdec model training on dataset NIST is 80, mini-batch size for RNNsearch model is 80 on NIST and 40 on WMT, mini-batch size for RNNsearch-H model training on dataset WMT is 10.

Exploring Large-Scale Training. The evaluation of models contains two parts. The first one is loss function, used to measure the degree of fit on training dataset. The second one is BLEU value of the models on the testing dataset in the training process, which represents the generalization of the trained models. We explore tradeoff between mini-batch size, accuracy and training convergence time.

We evaluate RNNsearch on dataset NIST. The convergence speed on training dataset is shown in Fig. 3(a) and (b). In Fig. 3(a), models trained with different mini-batch size fit the dataset very close after enough epochs. The increase of mini-batch size would not lead to a no-converge situation. As shown in Fig. 3(b), assessing on iterations, the model converge faster with the increase of mini-batch size. Large-scale training has a great advantage on the convergence speed for our framework provides a good scalability.

As shown in Fig. 3(c), training under large-scale training scenarios on WMT dataset, the trend of convergence speed is consistent with that on NIST dataset. Figure 3(d) shows that as mini-batch size increases, BLEU value increases faster. Under large-scale scenarios, the generalization of models increases faster when fitting faster on training dataset.

In terms of generalization of models trained with different mini-batch size, BLEU of RNNsearch on NIST assessing on epochs is shown in Fig. 3(e), with increase of mini-batch size, BLEU value increases slower assessing on epochs. From the perspective of training time, the convergence speed assessing on iterations is important. As shown in Fig. 3(f), assessing on iterations, when mini-batch size is above 32k (400 training nodes), the convergence speed of BLEU value does not increases. The convergence speed even decreases when mini-batch size reaches 64k (800 training nodes).

The highest BLEU results in Fig. 3(e) are shown in Table 1. Scaling mini-batch size to 4k leads no generalization gap. Scaling mini-batch size to 16k, leads 0.77 generalization gap, which is about 2.5% lose of BLEU value. As mini-batch size reaching 64k, we get 2.1 generalization gap, about 10% lose of BLEU value.

Training RNNsearch with 100 training nodes has an acceptable 0.3% lose of BLEU compared with the baseline of training with 10 training nodes. Training with 100 training nodes has a near convergence speed to the baseline assessing on epochs and converge quite faster than baseline assessing on iterations. So we select the mini-batch size of 8k as the scheme for training RNN/LSTM with large mini-batch size. **Training with mini-batch size 8k can bring a 100-fold convergence rate of training with a single node.**

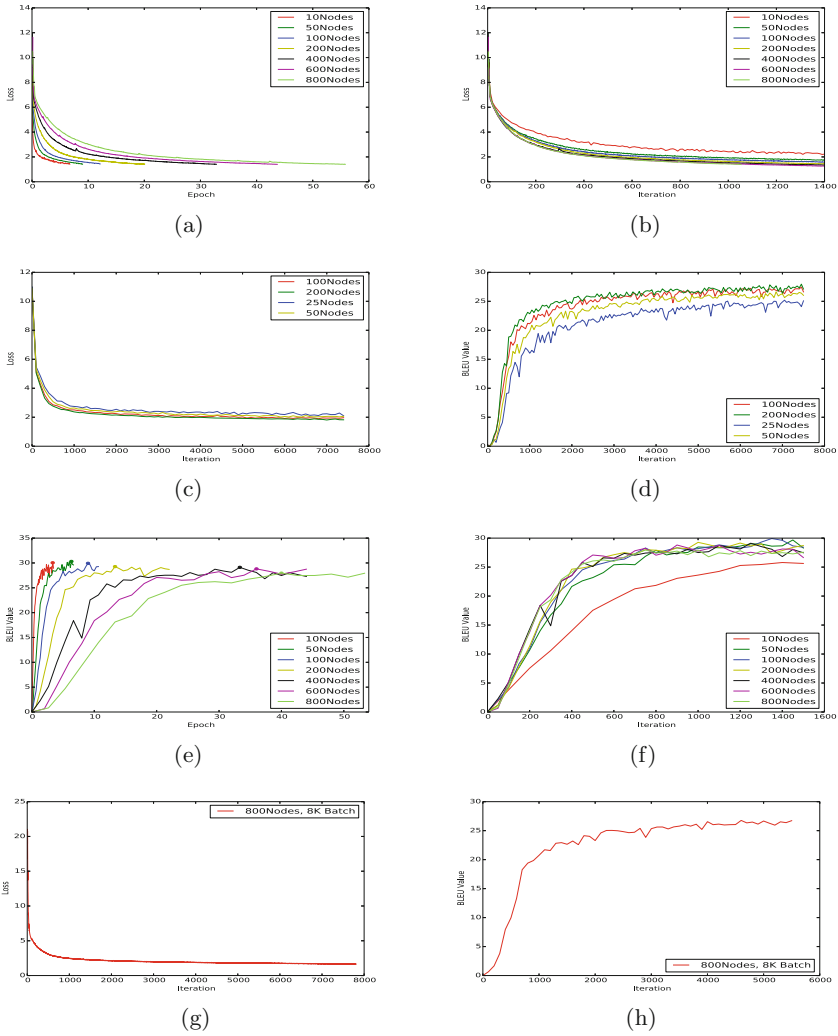

Fig. 3. (a) **Loss vs. Node Size** on **Epochs** of RNNsearch on NIST (b) **Loss vs. Node Size** on **Iterations** of RNNsearch on NIST (c) **Loss vs. Node Size** on **Iterations** of RNNsearch on WMT (d) **BLEU vs. Node Size** on **Iterations** of RNNsearch on WMT (e) **BLEU vs. Node Size** on **Epochs** of RNNsearch on NIST (f) **BLEU vs. Node Size** on **Iterations** of RNNsearch on NIST (g) Loss of RNNsearch-H with 800 Nodes (h) BLEU of RNNsearch-H with 800 Nodes

Table 1. Results of top BLEU of RNNsearch on NIST

Node size	10	50	100	200	400	600	800
Batch size	800	4000	8000	16000	32000	48000	64000
BLEU	30.04	30.31	29.93	29.27	29.12	28.81	27.95

5 Evaluation

In this section, we evaluate the performance of our framework in two ways firstly. One is to evaluate the computing performance of training on single worker and to compare the computing performance versus GPU. The other one is to evaluate the scalability of the framework. Secondly, we show the loss function and BLEU value both assessing on iterations of training model RNNsearch-H.

5.1 Performance

SW26010 vs. GPU. In terms of computing performance of single-worker training, we compare the performance of SW26010 with the performance of NVIDIA TITAN X. The single-precision computing capability of NVIDIA TITAN X is 11 TFlops, and the memory bandwidth of NVIDIA TITAN X is 505 GB/s. The double-precision computing capability of SW26010 is 3 TFlops. Compared with the double-precision computing capability, single-precision computing capability can reach 60% of double-precision computing capability [16], the memory bandwidth of SW26010 is 128 GB/s. As a result of the gap between NVIDIA TITAN X and SW26010, for one iteration training of RNNencdec, the average computing time on NVIDIA TITAN X is 0.71 s, the average computing time on SW26010 is 8.32 s. Actually, SW has a computing power of only 1/6 of NVIDIA TITAN X and 1/4 memory bandwidth of NVIDIA TITAN X, we can achieve 1/12 computing performance of NVIDIA TITAN X.

Table 2. Module time before and after architecture-oriented optimizations

Module	Time before	Percentage before	Time now	Percentage now
Total	66.765 s	100.0%	8.326 s	100.0%
Softmax	26.317 s	39.4%	0.583 s	7.00%
Activation function	34.363 s	51.4%	1.658 s	19.9%

The performance of training on SW is shown in Table 2. Data parallel optimization are operated on exponential layer. Optimizations described in Sect. 3.1 is operated on exponential layer and softmax layer. Training time in one iteration has decreases from 66.765 s to 8.326 s in RNNencdec after architecture-Oriented optimizations.

Scalability. After communication optimizations mentioned in Sect. 3.2, our framework provides high communication efficiency. When training RNNencdec under large-scale scenarios, scaling the size of training workers to 800 achieves 580x speedup, and parallel efficiency is 72.5%. When training RNNsearch under large-scale scenarios, scaling the size of training workers to 800 achieves 692x speedup, with parallel efficiency of 86.5%.

5.2 Experimental Results

As mentioned in Sect. 2, RNNsearch-H is with twice the size of hidden states and dimensional word embeddings of RNNsearch, the size of parameters is four times of RNNsearch, which needs about 30 GB memory space with mini-batch size 10. As mentioned in Sect. 4, we need 800 training nodes to reach the suitable mini-batch size 8k on SW. Training RNNsearch-H model under large-scale training scenarios on GPUs is unbearable for the memory bound and the limitation of the size of server cluster.

The loss function and BLEU value are shown in Figs. 3(g) and (h). As shown in Fig. 3(g), RNNsearch-H can converge to about the same level as model RNNsearch. RNNsearch-H can fit well on the training dataset. As shown in Fig. 3(g), BLEU value of RNNsearch-H can get to the same level as RNNsearch, after training enough time, the model can have a good generalization.

6 Conclusion

In this paper, we propose architecture-oriented optimizations for memory-intensive kernels in RNN/LSTM, exploring the parallelism in SW26010 many-core architecture. We propose a lazy communication scheme with improved MPI_Allreduce to achieve high communication efficiency and high scalability. We provide a distributed framework for large-scale NMT training to overlap all of testing time for frequently validation. At last, we provide discussions and analysis on convergence speed and generalization quality under different training mini-batch size and get a 100-fold convergence rate with 100 training nodes, 8k mini-batch size. We show an empirically training strategy, as well as the convergence and evaluation results, of training RNNsearch-H on a large dataset with 800 training nodes, 8k mini-batch size.

Acknowledgement. This work is supported in part by the National Key R&D Program of China (Grant No. 2017YFB0202204, 2017YFA0604500, 2016YFA0602200), by National Natural Science Foundation of China (Grant No. 91530323, 5171101179), and by the China Postdoctoral Science Foundation (Grant No. 2018M641359).

References

1. Arthur, P., Neubig, G., Nakamura, S.: Incorporating discrete translation lexicons into neural machine translation. arXiv preprint arXiv:1606.02006 (2016)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
3. Britz, D., Goldie, A., Luong, T., Le, Q.: Massive exploration of neural machine translation architectures. arXiv preprint arXiv:1703.03906 (2017)
4. Fang, J., Fu, H., Zhao, W., Chen, B., Zheng, W., Yang, G.: swDNN: a library for accelerating deep learning applications on Sunway TaihuLight. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 615–624. IEEE (2017)

5. Goyal, P., et al.: Accurate, large minibatch SGD: training imageNet in 1 hour. arXiv preprint arXiv:1706.02677 (2017)
6. Hasanov, K., Lastovetsky, A.: Hierarchical optimization of MPI reduce algorithms. In: Malyshkin, V. (ed.) PaCT 2015. LNCS, vol. 9251, pp. 21–34. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21909-7_3
7. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
8. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 675–678. ACM (2014)
9. Jiang, L., et al.: Towards highly efficient DGEMM on the emerging SW26010 many-core processor. In: 2017 46th International Conference on Parallel Processing (ICPP), pp. 422–431. IEEE (2017)
10. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: International Conference on Neural Information Processing Systems, pp. 1097–1105 (2012)
12. Li, M., et al.: Scaling distributed machine learning with the parameter server. In: OSDI, vol. 14, pp. 583–598 (2014)
13. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533 (1986)
14. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
15. You, Y., Gitman, I., Ginsburg, B.: Scaling SGD batch size to 32K for imagenet training. arXiv preprint arXiv:1708.03888 (2017)
16. Zhao, W., Fu, H., Fang, J., Zheng, W., Gan, L., Yang, G.: Optimizing convolutional neural networks on the sunway taihulight supercomputer. ACM Trans. Arch. Code Optim. (TACO) **15**(1), 13 (2018)