7-2023

# Distributed Deep Learning Optimization of Heat Equation Inverse Problem Solvers

Zhuowei Wang
*Guangdong University of Technology*

Le Yang
*Guangdong University of Technology*

Haoran Lin
*Guangdong University of Technology*

Genping Zhao
*Guangdong University of Technology*

Zixuan Liu
*Inner Mongolia University*

*See next page for additional authors*



## Citation Details

Wang, Z., Yang, L., Lin, H., Zhao, G., Liu, Z., & Song, X. (2023). Distributed Deep Learning Optimization of Heat Equation Inverse Problem Solvers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

# Distributed Deep Learning Optimization of Heat Equation Inverse Problem Solvers

Zhuowei Wang, Le Yang, Haoran Lin, Genping Zhao, Zixuan Liu, Xiaoyu Song

***Abstract*—The inversion problem of partial differential equation plays a crucial role in cyber-physical systems applications. This paper presents a novel deep learning optimization approach to constructing a solver of heat equation inversion. To improve the computational efficiency in large-scale industrial applications, data and model parallelisms are incorporated on a platform of multiple GPUs. The advanced Ring-AllReduce architecture is harnessed to achieve an acceleration ratio of 3.46. Then a new multi-GPUs distributed optimization method GradReduce is proposed based on Ring-AllReduce architecture. This method optimizes the original data communication mechanism based on mechanical time and frequency by introducing the gradient transmission scheme solved by linear programming. The experimental results show that the proposed method can achieve an acceleration ratio of 3.84 on a heterogeneous system platform with two CPUs and four GPUs.**

***Index Terms*—Distributed deep learning, concurrent computation, heat equation, gradient optimization.**

## I. Introduction

THE acquisition of data is a challenge task in the field of cyber-physical computing. . Before the development of data-driven deep learning, many types of industrial software and physical computing are driven by physical models. The physical models used to calculate predictions often contain some prior knowledge, such as Navier-stokes equations in electromagnetic field theory [1], Maxwell [2], Schrodinger equation in quantum mechanics [3]. However, the efficiency of these calculation methods is low in terms of physical calculation burden required.

Some researchers began to use deep learning methods to solve ordinary differential equations and partial differential equations [4]. Isaac Elias Lagaris et al. [5] employed intelligent neural networks to solve partial differential equations and studied the use of neural network methods to solve differential equations with complex boundary conditions [6]; however, due to the limitations of computing methods and hardware at that time, the research in this field does not gain sufficient attention. In recent years, with the development of deep learning and computing hardware technology, more and more researchers begin to pay attention to the research in this field. Regazzoni et al [7]. solve ordinary differential equations or time-varying partial differential equations based on data-driven neural network models. Maziar Raissi et al. [8]–[10] began to pay attention to the research in this field and proposed a new deep learning framework named Physics-informed Neural Networks(PINN) that can be applied to the solution of physical model of fluid mechanics and quantum mechanics. Then Mao et al. [11] used PINN to solve the Euler equation. Chen H et al. used an improved Cuckoo search algorithm (CSA) to solve the inverse problem of the heat conduction equation, which improved the accuracy of its solution calculation [12].

But its calculation efficiency is low, so it is difficult to apply to the industrial software applications that need largescale calculation. In order to meet the efficiency requirements of large-scale computing for industrial applications, more and more studies have applied distributed computing technology to large-scale computing for industrial applications. Priya Goyal [13] used multi-GPU distributed computing technology to train the deep learning model, makes full use of GPU computing resources, expands from 8 GPUs to 256 GPUs, and obtains a near-90% improvement in efficiency.

The main contributions of this article are as follows. 1) Our first contribution involves the development of HEInex, an image regression network specifically designed for identifying high transient heat flux density distributions in pool boiling problems. HEInex utilizes a multi-layer convolutional-deconvolutional network architecture as a dynamic solver for the inverse problem of heat conduction partial differential equations. By observing high-resolution temperature distributions on the heated surface, we are able to accurately estimate the local high transient heat flux density on the boiling surface, thus allowing for precise capture and reconstruction of real pool boiling phenomena. We performed an image quality comparison between the heat maps predicted by our deep learning model and the heat maps computed by using COMSOL, and the structural similarity index measure (SSIM). The SSIM value obtained was above 94%, indicating an excellent similarity between the two sets of images. This experimental result demonstrates the high accuracy of our deep learning model, which meets the requirements for industrial applications. 2) In the context of deep learning-based pool boiling problem solving, our second contribution is the pioneering use of the distributed training framework Horovod. This framework

This work was sponsored in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101190003, 2021B0101190004, 2021B0101190002, in part by the Innovation Fund for Industry-University-Research in Chinese Universities under Grant2021FNA02010. (Corresponding author: Genping Zhao)

Z. Wang is with the School of Computers, Guangdong University of Technology, Guangzhou 510006, China (e-mail: wangzhuowei0710@163.com).

L. Yang is with the School of Computers, Guangdong University of Technology, Guangzhou 510006, China (e-mail: 2112105002@mail2.gdut.edu.cn).

H. Lin is with the School of Computers, Guangdong University of Technology, Guangzhou 510006, China (e-mail: linhorizon@163.com).

G. Zhao is with the School of Computers, Guangdong University of Technology, Guangzhou 510006, China (e-mail: genping.zhao@gdut.edu.cn).

Z. Liu is with the College of Computer Science, Inner Mongolia University, Hohhot, Inner Mongolia 010021, China (e-mail: liu_zixuan163@163.com).

X. Song is with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207 USA (e-mail: songx@pdx.edu).

offers significant advantages, notably the adoption of the Ring-Allreduce algorithm. By connecting the communication patterns of GPUs in a circular manner, it effectively reduces resource consumption associated with increasing GPU numbers, leading to decreased training time and enhanced efficiency of distributed devices. Experimental results on a heterogeneous system with two CPUs and four GPUs demonstrate a speedup ratio of 3.46 achieved by the Ring-Allreduce algorithm. 3) To address the limitation of fixed clock frequency in the gradient data communication of the Ring-Allreduce parallel method, our third contribution introduces the GradReduce method. This method optimizes the mechanical gradient transmission scheme in Ring-Allreduce with a linear programming-based gradient transmission approach, which is better suited for efficient communication and execution of gradients with varying volume sizes for solvers. The GradReduce method optimizes the gradient transmission process of Ring-Allreduce method. Experimental results on a heterogeneous system with two CPUs and four GPUs show that the new implementation of Ring-Allreduce with GradReduce leads to a speedup ratio of 3.84 and an approximately 11% improvement in training time compared to original implementation of Ring-Allreduce.

This article is organized as follows. Section II summarizes the related research work of distributed algorithm. Section III introduces the distributed algorithm of the design and research of object solver models. Section IV summarizes and analyzes the experimental results. The conclusion of this research is presented in Section V.

## II. Related work

Data parallelism [14] is the most commonly used distributed method. For example, Alex Krizhevsky et al. [15], Dean et al. [16], Simonyan et al. [17], all adopted data parallelism in their research and experimental methods in the field of image recognition, which is the application for processing, analyzing, and understanding images to identify objects and objects with various patterns. There is a complete deep learning training model on each worker node which is a computing device node, such as GPU. Worker node can independently calculate forward and reverse gradients, and then the worker node sends these gradients to the parameter server.Many major deep learning frameworks, such as TensorFlow and PyTorch, support data parallelism. The parameter server calculates and updates all gradient parameters and resends the updated model parameters to each working node.

Model parallelism [18] is less common than data parallelism. This method divides the model into each working node and computes with batches of the same size. It is usually suitable for training of large model, which is neural network models with deep layers or a large number of parameters.For example, the parallelism of the models of Wu et al. [19] and Szegedy et al. [20] are limited on the premise of ensuring accuracy. Therefore, in the training of large-scale deep learning models, Saptadeep Pal et al. used the combination of model parallelism and data parallelism to overcome the communication overhead of data parallelism and optimize the computational efficiency of model parallelism . But all working nodes in a centralized architecture need to send gradients to the parameter server. Therefore, as the complexity of the deep learning model increases and the number of GPUs increases, the communication overhead of sending gradient data also increases. The combination of model and data parallelisms is adopted to improve efficiency in some research [21].

In order to speed up the training efficiency, some researchers optimized the gradient communication mechanism and proposed a decentralized Ring-AllReduce architecture with parameterless servers [22]. All working nodes form a ring structure. During data communication, all working nodes only receive data from one working node and send data to the other working node. Compared with the centralized architecture, the gradient communication and parameter updating speed are improved during model training. Tensorflow platform also optimizes gradient communication between layers through graph optimization. For deeper networks such as ResNet, each communication incurs additional communication overhead. Horovod proposes a gradient fusion approach to reduce the additional communication overhead [23]. A gradient fusion method is proposed to reduce the extra communication overhead. Although Ring-AllReduce architecture solves the communication overhead problem of centralized architecture, there is still room for optimization of gradient data fusion communication mechanism based on fixed time and frequency for large parameter network model.

This paper introduces the development of HEInex, an image regression network specifically designed for identifying high transient heat flux density distributions in pool boiling. The network utilizes a multi-layer convolution-deconvolution architecture as a dynamic solver for solving the inverse problem of thermal conduction partial differential equations. The use of solver conducted on the distributed platform with multiple GPUs under various distributed algorithms, including data parallelism and model parallelism, are also explored for efficient implementation of this solver. Notably, this study incorporates the Horovod distributed training framework for the first time and leverages the Ring-Allreduce algorithm to mitigate resource consumption as the number of GPU cards increases, thereby reducing training time. Additionally, to address the limitations brought by fixed-clock frequency in Ring-Allreduce's parallel gradient data communication, the paper also proposes the GradReduce approach for better communication of gradient data. This works for adaption to different gradient data sizes and solver operations and, further reduces training time costs and enhances computational efficiency on distributed devices.

## III. Methods

### A. The Inverse Problem of Heat Conduction Partial Differential Equation Based on Deep Learning

Based on the pool boiling problem, this paper is attempted to solving the inverse problem of the partial differential equation of heat conduction, that is, to inverse calculate the high transient local heat flux on the boiling surface by observing the high-resolution temperature distribution on the heating surface. From a mathematical point of view, this is

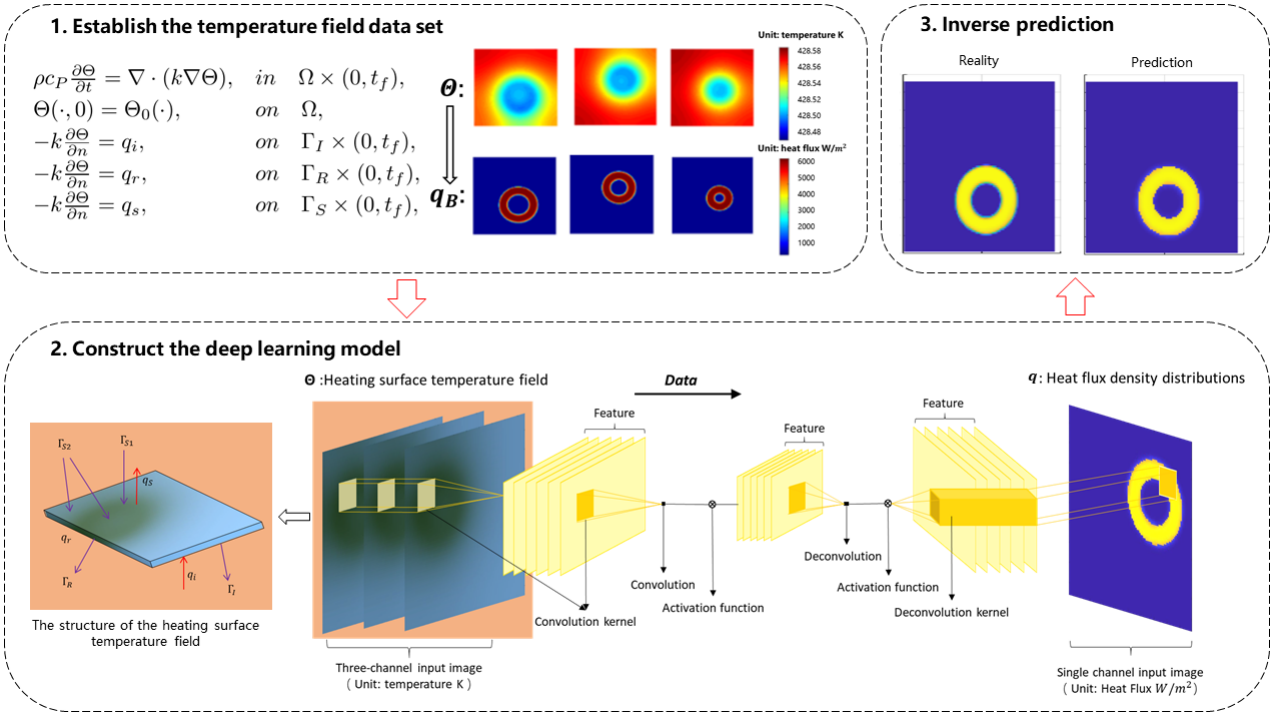



Fig. 1. Flow diagram of modelling of inverse problem solvers for partial differential equation of heat conduction. (1) Step1: Flow diagram of establishing the temperature field data set. (2) Step2: Structure diagram of the deep learning model of heat equation solvers. (3) Step3: Comparison of inverse predictions: real data and predicted data.

equivalent to a three-dimensional inverse problem of transient heat conduction identified by unknown Neumann boundary conditions [24]. In general, the inverse problem of Neumann boundary identification heat conduction is solved by using the method of finite element analysis combined with iterative optimization [25]. However, this method requires calculation of the optimization problem constrained by partial differential equations, and the amount of calculation data is large, resulting in high computational cost. In this paper, a proxy model of the abstract operator of the heat conduction equation is constructed by using the deep learning method, and the multi-layer convolution deconvolution network is established as the dynamic solver of the inverse problem calculation of the heat conduction partial differential equation. The overall process of using this method is shown in Fig. 1, and is divided into three steps: establishing a training set, constructing a deep learning model, and predicting the unknown flow of the inverse heat transfer problem.

$$F(\Theta)=\begin{cases}\rho c_P\frac{\partial\Theta}{\partial t}=\nabla\cdot(k\nabla\Theta), & in \quad \Omega\times(0,t_f),\\ \Theta(\cdot,0)=\Theta_0(\cdot), & on \quad \Omega,\\ -k\frac{\partial\Theta}{\partial n}=q_i, & on \quad \Gamma_I\times(0,t_f),\\ -k\frac{\partial\Theta}{\partial n}=q_r, & on \quad \Gamma_R\times(0,t_f),\\ -k\frac{\partial\Theta}{\partial n}=q_s, & on \quad \Gamma_S\times(0,t_f),\end{cases} \tag{1}$$

**Step 1:** Establishing the training set: since the deep learning method needs enough training data to obtain more accurate solutions, we first establish an accurate forward heat transfer model. The model is solved through the open source software COMSOL [26], to acquire sufficient training data. The forward heat transfer mathematical model can be expressed as Equation (1).

$\rho$, $c_p$, and $k$ are expressed as material density, specific heat capacity and thermal conductivity respectively. $t$ represents the unit time, $(0, t_f)$ denotes time threshold, and $\Omega$ is space threshold. $\Gamma_I$, $\Gamma_R$, and $\Gamma_S$ denote the temperature field of the lower surface, the side surface and the upper surface respectively. $n$ represents the length of the time series. $\Theta$ refers to the temperature field of the heating surface, which is a three-dimensional tensor. It can be described as a function $\Theta(t, x, y)$ that varies with time and space ($t$ indicates the temporal threshold component, and $x$ and $y$ stand for the spatial threshold components, respectively).

$q_i$ denotes the heat flow on the lower surface, which is a constant; $q_s$ is the hear flow on the upper surface; $q_r$ represents the side heat flow, and it is assumed to be 0. Similarly, $q_i$, $q_s$, and $q_r$ are all three-dimensional tensors, which can also be described as function $q_i(t, x, y)$, $q_s(t, x, y)$, $q_r(t, x, y)$ varying with time and space. The heat flux in different directions finally constitutes the heat flux parameter $q(t, x, y)$.

Therefore, many temperature field images and heat flux distribution images composed of $\Theta$ and $q$ can be obtained in the process of solving the forward heat conduction mathematical model. These images can be used as training data for our deep learning.

**Step 2:** Deep learning model: the abstract operator $F$ is defined as the implicit representation function of formula (1), so $F$ is described as the mapping from the temperature file on the heating surface ($\Theta$) to the heat flux distribution on the boiling surface ($q$), which can be expressed as

$$F(\Theta(t,x,y)) = q(t,x,y) \tag{2}$$

Three dimensional tensors $\Theta(t,x,y)$ and $q(t,x,y)$ have the components of the spatial threshold $\Omega$ in the $x$ and $y$ spatial directions and the components of the temporal threshold $(0,t_f)$. In a broad sense, tensor $\Theta$, $q$ can be regarded as the temperature field image and the heat flux distribution image which have changed dynamically on the time axis. Therefore, an image regression network HEInex is established based on convolutional neural network, which is suitable for pool-boiling, high transient heat-flux distribution recognition.

The network structure of HEInex is shown in Fig. 1 (2). It can be divided into output layer, convolution layer, activation layer deconvolution layer, and output layer. The input is the temperature field $\Theta(t^{n-2},x,y)$,$\Theta(t^{n-1},x,y)$,$\Theta(t^{n},x,y)$ when the time frame is $t^{n-2}$,$t^{n-1}$,$t^{n}$. The output is set as the heat flux distribution $q(t^n,x,y)$ when the time frame is $t^n$.

Convolution and image sampling in convolution layer and deconvolution layer can be implemented realize the mapping of $\Theta \rightarrow q$. So that the heat conduction abstract operator $F$ can be replaced.

Here, we use the mean square loss function as shown in formula (3). By selecting appropriate gradient descent techniques, this convolutional neural network can be effectively trained and the desired solution is obtained as.

$$MSE(q,\widetilde{q}) = \frac{\sum_{i=1}^{n}(q_i - \widetilde{q}_i)^2}{n} \tag{3}$$

$n$ is the number of samples. $q$ represents the real heat flux distribution value calculated by the abstract operator $F$ of heat conduction. $\widetilde{q}$ is the predicted heat flux distribution value obtained by the convolutional neural network.

**Step 3:** Inverse prediction: finally, the actual test data $q$ of heat flux distribution are compared with the predicted data $\widetilde{q}$ through the structural similarity index measure (SSIM) [27], as shown in Equation (4).

$$SSIM(q,\widetilde{q}) = \frac{(2\mu_q\mu_{\widetilde{q}} + c_1)(2\sigma_{q\widetilde{q}} + c_2)}{(\mu_q^2 + \mu_{\widetilde{q}}^2 + c_1)(\sigma_q^2 + \sigma_{\widetilde{q}}^2 + c_2)} \tag{4}$$

$\mu_q$ and $\mu_{\widetilde{q}}$ represent the average of $q$ and $\widetilde{q}$, respectively. $\sigma_q$ and $\sigma_{\widetilde{q}}$ denote the standard deviation of $q$ and $\widetilde{q}$, respectively. $\sigma_{q\widetilde{q}}$ refers to the covariance of $q$ and $\widetilde{q}$. $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ are constants used to maintain stability. $L$ is the dynamic range of the pixel value. $k_1 = 0.01$, $k_2 = 0.03$. The value of SSIM ranges from $-1$ to 1.

In real-world industrial applications, a higher level of precision is often demanded compared to other domains. However, there is no established standard for SSIM values within the context of industrial applications. As a result, we have adopted an SSIM of 90% as the benchmark for satisfactory performance in industrial settings, based on the references [28]–[32]. Hence, for the time being, this study posits that if the $SSIM > 0.9$, the similarity between the predicted heat flux image and the actual heat flux image is sufficient to meet the demands of industrial computing applications. Nonetheless, the suitable SSIM value should be determined according to the specific requirements of individual applications.

### *B. Model parallelism and data parallelism*

Model parallelism and data parallelism are used in this research to improve the efficiency of the solvers of the heat equation. These methods are based on a centralized architecture. The parameter server function of the centralized architecture is performed with the CPU, and the computing function of the working node is implemented with the GPU.

Model parallelism [18] assigns the model and its associated solvers to each GPU for computation. This is often used to train large network models. If the neural network model involved in the deep learning task has deep layers and many parameters, exceeding the upper limit of the local memory storage of a single GPU, the model needs to be decomposed. Different GPUs store different parts of a neural network model, and each GPU is only responsible for updating the parameters of the local storage part of the network and computing this part of the network. This distributed machine learning approach is called model parallelism. In a multi-GPU distributed scenario, model parallelism requires manual model partitioning of GPU resources. Artificial model partitioning may fail to achieve optimal partitioning, leading to the invalid utilization of GPU hardware resources and load-balancing problems on different GPUs. It will lead to a high time-cost of solver operation and poor optimization effect.

When the solvers use data parallelism [14] to solve the heat equation, the dataset of heating surface temperature field $\Theta$ which size is $Data$ is divided into subsets. Then it distributes the subsets of $\Theta$ to different GPUs. Each GPU has a complete neural network model, which independently performs forward and reverse calculations according to the distributed data subset to obtain the local gradient subset of $\Theta$. These gradients are driven by the GPU to the parameter server CPU, which then aggregates and updates them. Finally, the updated model weight parameters are returned to each GPU to complete an iteration.

### *C. Ring-AllReduce*

The Ring-AllReduce architecture [22] was originally proposed by Andrew Gibiansky for the Baidu Research Technology blog. We assume that there are $N$ GPUs in our system. $B$ denotes the bandwidth of GPU communication. $Data$ stands for the amount of the data to calculate. $D_{cal}$ describes the amount of the data to calculate in single GPU, and $D_{com}$ describes the amount of the data to communicate in single GPU. $T$ is the theoretical time required for data communicate.

When the solvers apply the Ring-AllReduce, the computing device will receive data of $Data$ size. Each GPU in the Ring-AllReduce architecture will send and receive values $N-1$ times for Scatter-Reduce and $N-1$ times for all-Gather. Each GPU will send gradient data with size of $Data/N$. The amount of data to communicate in single GPU and the theoretical time of data communication is expressed as follows:

$$\begin{aligned} D_{cal} &= Data/N \\ D_{com} &= 2(N-1)/D_{cal} \\ T &= D_{com}/B \end{aligned} \tag{5}$$

The communication overhead of ring-AllReduce communication eliminates most of the influence of the number of GPUs of working nodes and does not increase with the increasing number of GPUs. Overall communication speed is limited only to adjacent GPUs in the slowest connected ring and this can average the gradient in neural network, ensure convergence and accuracy as far as possible, and reduce communication overhead.

### *D. GradReduce*

The GradReduce method proposed in this article is based on the Ring-AllReduce method. It uses the ring architecture of multi-GPU distributed computing, and optimizes the original gradient data communication mode of Ring-AllReduce. Each gradient data communication of Ring-AllReduce is based on a fixed clock frequency. The total communication time of this gradient data communication mode is limited by the time-consuming gradient data calculation.

The GradReduce method gradient sending scheme is based on the combination of the environmental parameters of a distributed computing platform and iterative neural network parameters. GradReduce method can obtain the optimal data communication scheme through linear programming to minimize the total time. The specific variable parameters are listed in Table I.

TABLE I
PARAMETERS OF THE LINEAR PROGRAMMING

| Parameters | Definition |
|---|---|
| $Layer$ | Network layer |
| $i$ | Network layer rank |
| $B$ | The bandwidth constrained by GPU hardware parameters |
| $start$ | The first layer rank |
| $end$ | The last layer rank |
| $\nu$ | GPU computing power |
| $D_{cal}(i)$ | The amount of gradient data of layer $i$ to calculate |
| $D_{com}(i)$ | The amount of gradient data of layer $i$ to communicate |
| $Ex$ | The delay caused by insufficient device bandwidth during GPU communication |
| $Cost$ | The fixed cost time of communication |
| $State(i)$ | Gradient data communication state at layer $i$ |
| $Ts_{cal}(i)$ | The start time of gradient data calculation for layer $i$ |
| $Te_{cal}(i)$ | The end time of gradient data calculation for layer $i$ |
| $T(i)$ | The theoretical time of gradient data communication at layer $i$ |
| $T_{tal}(i)$ | The total time of gradient data communication at layer $i$ |
| $Ts_{com}(i)$ | The start time of gradient data communication at layer $i$ |
| $Te_{com}(i)$ | The end time of gradient data communication at layer $i$ |
| $Bf(i)$ | The time for gradient data communication buffering in layer $i$ |
| $Buffer$ | The time allocated for gradient data communication buffering in GPU communication |

Given that the gradient data of each layer are the data of the temperature field of the heating surface, so the summation equations are shown in Equations (6) and (7). The GradReduce method gradient data calculation and communication process run independently. It does not affect the execution of other routines in the calculation process.

$$D_{cal} = \sum_{i=1}^{L} D_{cal}(i) \tag{6}$$

$$D_{com} = \sum_{i=1}^{L} D_{com}(i) \tag{7}$$

Layer $i$ gradient data calculation end time is obtained from the computing amount of gradient data of this layer and the computing capacity of hardware parameter GPU; the start time of gradient data calculation is obtained based on the end time of upper gradient data calculation. The calculation in Equations (8) and (9) can be expressed as follows:

$$Te_{cal}(i) = Ts_{cal}(i) + \frac{D_{cal}(i)}{\nu} \tag{8}$$

$$Ts_{cal}(i) \geq Te_{cal}(i-1) \tag{9}$$

Theoretical time of gradient data communication at layer $i$ is determined from the gradient data traffic of this layer and hardware parameter GPU communication, as calculated by Equation (10).

$$T(i) = \frac{D_{com}(i)}{B} \tag{10}$$

Layer $i$ network communication starts at $Ts_{com}(i)$. It needs to wait for the end of the communication of the upper layer network. In addition, communication can be conducted only after the gradient calculation of the layer $i$ is completed, and the time when the gradient calculation ends is $Te_{cal}(i)$, so Equation (11) can be acquired as follows:

$$Ts_{com}(i) \geq max(Te_{cal}(i), Te_{com}(i-1)) \tag{11}$$

The end time of data communication at layer $i$ should be calculated according to the start time of communication plus the communication cost. The communication cost is usually composed of the total time and the fixed cost time of communication. Data communication incurs a time-cost, multiplied by its extra cost coefficient, and Equation (12) can be obtained as follows:

$$Te_{com}(i) \geq (Ts_{com}(i) + T_{tal}(i) \cdot (1 + Ex) + Cost) \tag{12}$$

When the data communication of layer $i$ does not start, the theoretical sending time of the data that has not been communicated is $T(i)$. It can be calculated and added to the data buffer of layer $i$. Whether or not to join the cache is determined by the input parameter $State(i)$, which is a Boolean array representing the initial way in which the data are sent. This indicates that if the data of the layer $i$ are sent immediately, then $State(i) = 1$. If the data of this layer are not sent and stored in the buffer area, then $State(i) = 0$. Since the last data must be sent, the layer $i$ corresponds to $State(i) = 1$.

If the data pertaining to the layer $i$ are not sent, the buffer area $Bf(i)$ of this layer is at least composed of the value of the buffer area of the previous layer $Bf(i-1)$ plus the data

communication duration of this layer. If $State(i) = 1$ of the layer $i$, it means that the gradient data calculated by this layer and the accumulated data in the buffer area $Bf(i)$ will be sent. Therefore, Equation (13) can be obtained as follows:

$$\begin{cases} Bf(i) \geq (Bf(i-1) + T(i)) & State(i) = 0, \\ Bf(i) \geq 0 & State(i) = 1, \end{cases} \tag{13}$$

Combining Equation (13), the value of gradient data cache can be calculated as per Equation (14).

$$Bf(i) \geq (Bf(i-1) + T(i)) \cdot (1 - State(i)) \tag{14}$$

Since the theoretical total duration of data communication $T_{tal}(i)$ denotes the sum of the theoretical communication duration after the fusion of all gradient data of the layer $i$. When $State(i) = 0$, the data of this layer are not sent and stored in the buffer area. $T_{tal}(i)$ is recorded as 0.

If $State(i) = 1$ of the layer $i$, it means that the gradient data calculated by this layer and the accumulated gradient data in the buffer area will be blended, that is, the gradient data will be sent together after gradient fusion. The total time of communication of this layer, $T_{tal}(i)$, is composed of the theoretical data communication duration $T(i)$ of this layer and the accumulated value $Bf(i-1)$ in the buffer area of the previous layer. Equation (15) is derived as follows:

$$\begin{cases} T_{tal}(i) \geq 0 & State(i) = 0, \\ T_{tal}(i) \geq (Bf(i-1) + T(i)) & State(i) = 1, \end{cases} \tag{15}$$

Combining Equation (15), the total theoretical duration of gradient data communication can be calculated as per Equation (16)

$$T_{tal}(i) \geq (Bf(i-1) + T(i)) \cdot State(i) \tag{16}$$

According to the Equation (17), $State(i)$ is decided by $Buffer$, $T(i)$ and $Bf(i)$.

$$State(i) = \begin{cases} 0, & Buffer > T(i) + Bf(i) \\ 1, & Buffer \leq T(i) + Bf(i) \end{cases} \tag{17}$$

The process of obtaining $State(i)$ with the above equation constraints is one of linear programming; this is a way to solve the optimal solution of the objective function in the feasible region, which can be implemented using the CPLEX technique [33].

There are two cases about the solvers with GradReduce using the linear programming to optimize data communication of Ring-AllReduce (Fig. 2).

To better understand the communication mode advantages of the GradReduce distributed algorithm, we combined the Theoretical time and extra cost into communication cost.

Communication cost encompasses both the time required for theoretical data transmission (Theoretical time) and the delay caused by insufficient device bandwidth during GPU communication ($Ex$). Theoretical time can be calculated as a mathematical function of the gradient data traffic of a layer and the GPU communication hardware parameters, as shown in Equation (10). Extra cost specifically refers to the delay caused by insufficient device bandwidth in GPU communication, which is generally proportional to the size of data transmission. On the other hand, fixed cost ($Cost$) is independent of data size and is typically related to communication preparation between GPUs. This includes data transmission, memory allocation, thread synchronization, and other costs that do not change with data size but increase with an increase in communication frequency. In distributed training, fixed cost is a crucial factor that must be considered.

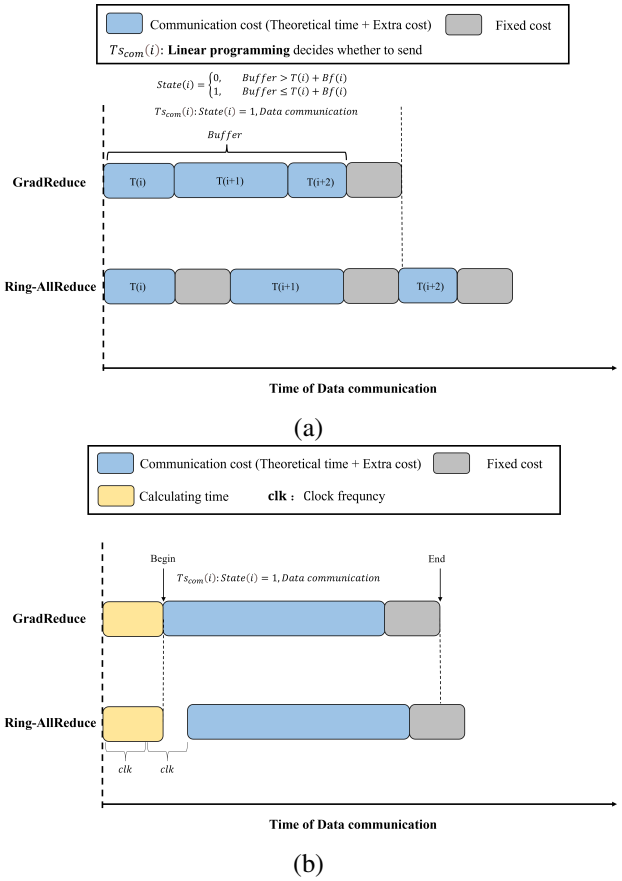



Fig. 2. Communication-calculating pipeline and the spatio-temporal diagram of the solvers using GradReduce and Ring-AllReduce. (a) Case 1: Multiple small-sized gradient data communication-calculating pipeline spatio-temporal diagram.(b) Case 2: Single large-sized gradient data communication-calculating pipeline spatio-temporal diagram.

**Case 1:** The spatio-temporal diagram of Case 1 is shown in Fig. 2(a). The solvers will send three small-sized calculated data for communication. If the solvers use GradReduce, the three calculated data will merge in $Buffer$ and be sent simultaneously, which generates one fixed cost. If the solvers using Ring-AllReduce send the same three calculated data, they will generate one fixed cost for each item of data sent. In this case, the solvers calculate and communicate with some small-sized gradient data items. The solvers using GradReduce or Ring-AllReduce have a same start time of data communication $Ts_{com}(i)$. According to Equation (9), if a calculated data is communicated to another GPU, the time cost of this data communication is composed of the

Communication cost, and the fixed cost. Therefore, each data communication of the solvers consumes the fixed cost $Cost$. According to Equation (14), if the solvers use GradReduce, the small-sized gradient data can be deposited in the buffer area when $State(i) = 0$. In addition, $State(i) = 1$ means that the size of buffer area $Buffer$ is too small to deposit the calculated data of this layer. The data in buffer area and the calculated data are merged for data communication. The solvers using GradReduce communicate with some small data to further generate less fixed cost. If the solvers use Ring-AllReduce, it will generate more time consumption cost from the fixed cost.

**Case 2:** The spatio-temporal diagram of Case 2 is shown in Fig. 2(b). In this case, the solvers calculate, and prepare to communicate with, a large-sized gradient dataset. The solvers using GradReduce or Ring-AllReduce have a same end-time for their data calculation $Te_{cal}(i)$. In layer $i$, $State(i) = 1$ refers to that the solvers using GradReduce will communicate. In addition, the solvers using Ring-AllReduce calculate and communicate with large-sized gradient data, which have a fixed clock frequency. The clock frequency is the frequency of the request calculated data for GPU communication. If the clock frequency of the solvers is set to some large value, it generates excessive waiting time after the calculated data are prepared. If the clock frequency of the solvers using Ring-AllReduce is set to some small value, the problem of Case 2 which has an excessive waiting time will not be obvious, but the problem of Case 1 happens, which generates more fixed costs. Since GradReduce uses a linear programming method to optimize communication, the waiting time problem arising in Case 2 will not occur in GradReduce. The linear programming model can be constructed through the above equation constraints, in which the decision dependent variable is set to $Te_{com}(end)$, that is the time at which the gradient data of the $end$ layer are sent to the end. The decision objective of the linear programming is to minimize the dependent variable $Te_{com}(end)$. If the value of $Te_{com}(end)$ is smaller, the communication scheme of GradReduce is also better, that is, the training efficiency of the model is higher. The execution of GradReduce is shown in Algorithm 1.

In the GradReduce algorithm, $FluxPara$ refers to the parameters of the neural network. $EnvPara$ denote the parameters of training environment. These parameter sets are all used to construct and solve the linear programming model, and then obtain the optimal gradient data sending scheme of this layer, from which the gradient sending start and end times follow.

GradReduce not only follows the Ring-Allreduce communication architecture to solve the problem of high cost of cross-machine implementation of data and model parallelisms, but also uses the linear programming to achieve greater effect than Ring-AllReduce.

**Algorithm 1:** GradReduce

**Input:** Number of training network layers:$\boldsymbol{Layer}$
Training environment parameters: $\boldsymbol{EnvPara}$;
Flux network Parameters: $\boldsymbol{FluxPara}$.
**Output:** End time of gradient data calculation for the last layer: $\boldsymbol{Te_{com}(end)}$.

**EnvPara** is determined by the training environment, including: $\nu$, $B$, $Ex$, $Cost$
**FluxPara** is generated by the torchstat tool, including: $D_{cal}$, $D_{com}$, $Ts_{cal}$.
**for** $i = \boldsymbol{start}$; $i \leq \boldsymbol{end}$; $i+ = 1$ **do**
  **if** $i \leq \boldsymbol{Layer}$ **then**
    Executing torchstat $\rightarrow$
    $D_{cal}(i)$, $D_{com}(i)$, $State(i)$, $Ts_{cal}(i)$.
  **end**
  Executing linear programming model:
  **while** $Te_{com}(i)$ *is the minimum value* **do**
    $State$ from equation(14)
    $T(i)$ from equation(7)
    $Bf(i)$ from equation(10)
    $T_{tal}(i)$ from equation(12)
    $Te_{com}(i)$ from equation(9)
  **end**
  Update $Te_{com}(i)$ in GPU memory;
**end**
return $\boldsymbol{Te_{com}(end)}$;

## IV. Experiments

### A. Experiment Setup

Since this research uses a deep learning method to establish a multi-layer convolution-deconvolution neural network, sufficient training data are required by the COMSOL tool. These data are all derived from the model for solving the heat conduction problem in the single-bubble boiling process in plate-type scenarios. The parameters of the model in Case 1 are shown in Table II. The specific calculation of the data features is displayed as follows:

$$c_P = 462 + 023 * (\Theta - 273.15) + 0.24 * (\Theta - 273.15)^2 \quad (18)$$

$$k = 62 + 0.0378 * (\Theta - 273.15) - 0.01 * (\Theta - 273.15)^2 \quad (19)$$

In this case, the initial temperature is set to $1550\,^{\circ}\mathrm{C}$. Lower surface heat flow: $q_i = 5000W/m^2$. Side heat flow: $q_r = 0$. According to the heat flow [34], upper surface heat flow calculation is governed by Equations (20) to (22).

$$q_s(t, x, y) = a(t) \cdot b(x, y) \quad (20)$$

$$a(t) = \begin{cases} 0.1\sin(100\pi(t - 0.02)) + 0.1 & 0.015 \leq t \leq 0.035, \\ 0 & otherwise, \end{cases} \quad (21)$$

$$b(x, y) = \begin{cases} 1 & (x, y) \in \Gamma_{S_1}, \\ 0 & (x, y) \in \Gamma_S \setminus \Gamma_{S_1}, \end{cases} \quad (22)$$

The upper surface space domain boundary conditions are shown in Equation (23).

TABLE II
PARAMETERS OF THE MODEL IN CASE 1

| Parameter | | Values |
|---|---|---|
| Geometric properties | Length,$L(m)$ | $10^{-3}$ |
| | Width,$W(m)$ | $10^{-3}$ |
| | thickness,$d(m)$ | $2.5 \times 10^{-5}$ |
| Physical properties | Density,$\rho(kg/m^3)$ | 7850 |
| | Specific heat capacity,$c_P(J/(kg \cdot K))$ | Equation(18) |
| | Heat conductivity coefficient,$k(W/(m \cdot K))$ | Equation(19) |
| Grid discretization | Grid type | hexahedron (Structured network) |
| | Number of grids | 50000 |
| | Size($x.y-$Direction,m) | $10^{-5}$ |
| | Size($z-$Direction,m) | $5 \times 10^{-6}$ |
| | Time discrete$Dt(s)$ | 0.001(50 step) |
| Amount of computation | Degrees of freedom | 624276 |
| | Calculating time$(s)$ | 714 |

$$\Gamma_{S_1} : r - 0.1 \leq \sqrt{(x-m)^2 + (y-n)^2} \leq rmm; \quad z = 0.025mm; m, n \in [0.5, 0.7]mm \tag{23}$$

The running memory of this data case is 2-3 G, the calculation precision is set to $1 \times 10^{-4}$, and the size of the entire model after the calculation is completed is 531 MB.

The experimental platform used was a heterogeneous system consisting of an Intel i9-10850K CPU with 10 CPU cores and multiple NVIDIA GeForce RTX 3090 GPUs. The dataset used for this study included 1040 training samples, 130 testing samples, and 130 validation samples. The model's hyperparameters included using the Adam optimizer with a learning rate of $10^{-4}$, training for 50 epochs, and using a batch size of 32. The neural network model was trained on up to 4 GPUs and 2 CPUs, using the Deep Learning of PyTorch. The solver model's parameters and computational information can be found in Table III.

TABLE III
PARAMETERS AND TRAINING INFORMATION OF HEINEX

| Parameters | | Values |
|---|---|---|
| Data size | Training | 1040 |
| | Validation | 130 |
| | Test | 130 |
| Model Parameters | Optimization | Adaptive Moment Estimation (Adam) |
| | Learning rate | $10^{-4}$ |
| | Epochs | 50 |
| | Batch size | 32 |
| Traning modes with multiple GPUs (NIVIDA GeForce RTX 3090 GPU) | | Single CPU (Intel i9-10850K CPU 10 cores) |
| | | Multiple CPUs (Intel i9-10850K CPU 10 cores) |

### *B. Impact of spatio temporal resolution on results*

In this experiment, the settings of the learning rate and batch can be adjusted according to the specific experimental platform and data characteristics. For the experimental platform with strong multi-GPU distributed cluster computing capability, the batch of training data can be appropriately increased the better to estimate the computing performance of the solvers under different distributed methods.

A data case of the constructed heat conduction partial differential equation inverse problem solvers trained on a single-machine single-GPU platform is displayed in Fig.3. It is found that the SSIM of the image exceeds 90% which meets the requirements of industrial calculating applications.

In the previous section, we presented our experimental setup and results. Building on our earlier analysis of the impact of temporal resolution on image quality, we then examined the impact of four major hyperparameters (learning rate, epochs, batch size, optimizer) on SSIM. To complement our earlier findings on the impact of temporal resolution, we analyzed the changes in SSIM with respect to temporal resolution for different hyperparameter values, and provided a figure (Fig.4) to illustrate the results.

We were able to identify the following patterns through our analysis using Fig.4:

Batch size has a significant impact on training time. Firstly, reducing the batch size will result in longer training time for the deep learning model. Secondly, as shown in Figure 4(a), increasing the batch size can lead to increased fluctuations in SSIM. Therefore, in our testing of six different batch sizes (512, 256, 128, 64, 32, 16), we found that a batch size of 32 is the most stable and achieved the best results.

The number of epochs is positively correlated with the training time of the deep learning model, but increasing the number of epochs can also improve the SSIM performance. As shown in Figure 4(b), increasing the epochs will lead to longer training time, but we found that achieving good results is possible with 200 epochs.

A learning rate that is too small can lead to poor performance of the deep learning model, while a larger learning rate may achieve better SSIM performance (as shown in Figure 4(c)). Although the Adam optimizer has the function of adapting to adjust the learning rate, too large a learning rate can increase instability, while too small a learning rate may

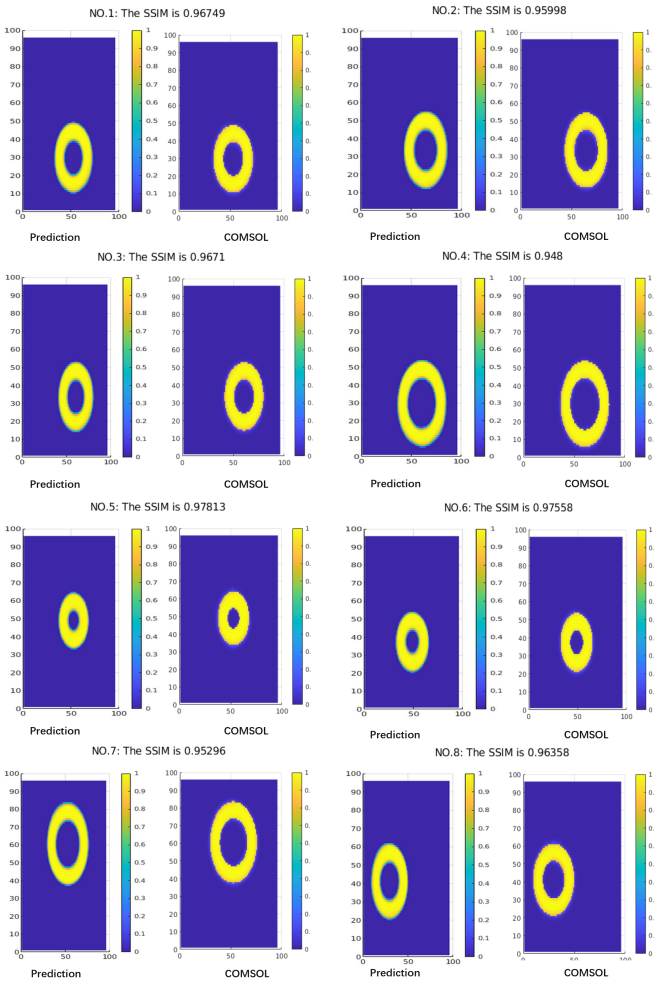



Fig. 3. Deep learning-based prediction of heat flux density distribution images (left figure). COMSOL simulation of actual heat flux density distribution (right figure).

result in poor optimization of SSIM. Therefore, when using the Adam optimizer, we selected a relatively stable learning rate of 0.001.

Among the four optimizers that were tested, the Adam optimizer yielded the best performance (as shown in Figure 4(d)). We compared the SGD, Adam, Adagrad, and RMSprop optimizers. While SGD could steadily reduce the model's loss, its improvement in SSIM was relatively slow, requiring a large number of iterations to achieve better results. Adagrad adapts the learning rate according to different parameter gradients, but its performance showed significant fluctuations despite finding a relatively good SSIM value. RMSprop is similar to Adagrad, but it introduces exponential weighted averages to smooth the gradient's historical information and adapt to different gradient changes. On the other hand, Adam combines the advantages of momentum and the RMSprop algorithm. It can adaptively adjust the learning rate and momentum coefficient and can adapt to different gradient changes, which ultimately results in the best performance.

### C. Training Time for Each Experiment

In this research, the solvers are trialed by applying the traditional data parallelism on a single computer. The experimental results are summarized in Table IV.

In the control experiments of heterogeneous systems with single computer with multiple GPUs, the data parallelism improves the speedup ratio of the solver model training by 1.61. Data parallelism has a centralized architecture that uses a single CPU as a parameter server to schedule gradient data. Data parallelism requires assigning solver models of the same structure to different GPUs, sending gradient data to different GPUs to update the gradient data. The updated gradient data are merged in a single CPU that is transferred to the parameter server. Therefore, the data parallelism cannot be applied on multiple CPU heterogeneous system, rendering data parallelism unsuitable for solvers required to run on a multiple-CPU, multiple-GPU distributed platform.

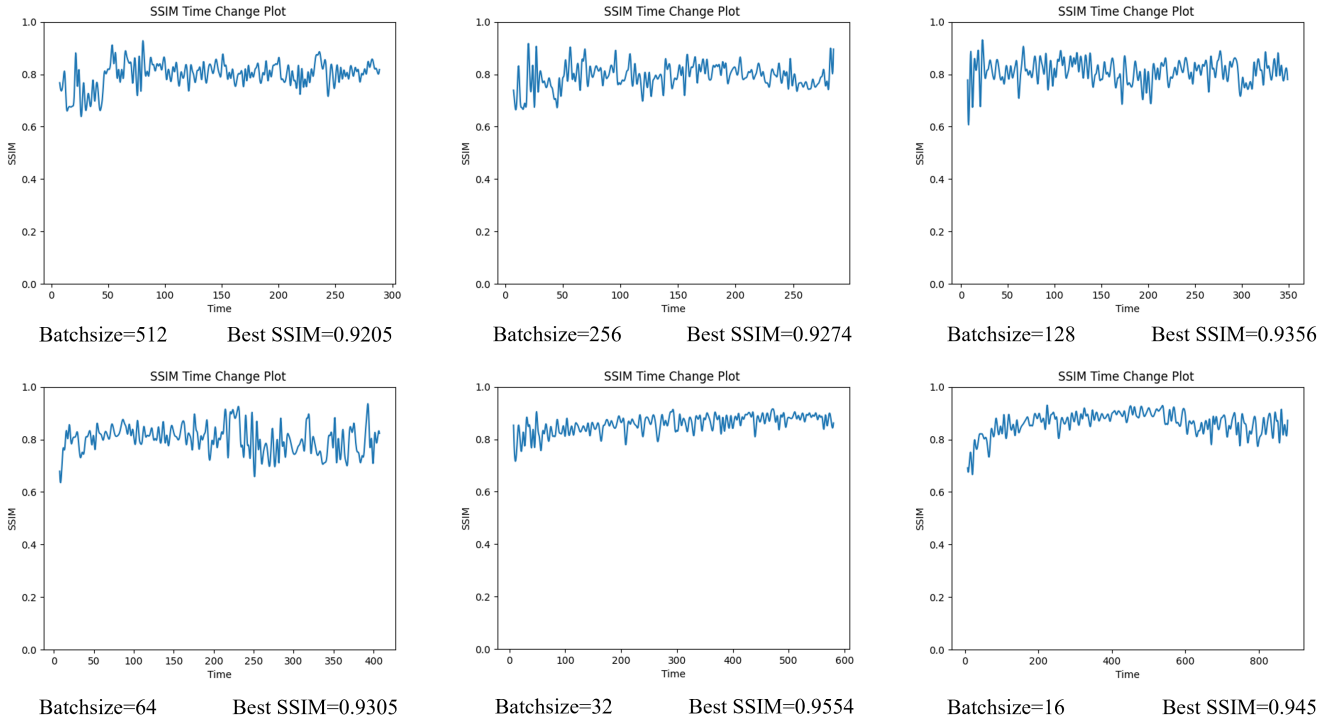
Model parallelism is applied in the same experimental environment. The model parallelism improves the computational efficiency of the solvers by a factor of 1.37, and the increase in efficiency is not as good as the effects of data parallelism. The solvers of the model parallelism can be executed in the systems with multiple computers with multiple GPUs, but only achieves the desired accelerating factor of 1.42, the model training efficiency is improved very little, and the computing performance of multiple GPUs is not fully utilized. The experimental results show that the training efficiency is very small. So, model parallelism is not the best distributed strategy for the solvers in the course of the execution.

The results show that the solver models use the Ring-Allreduce method to obtain accelerations of 1.94 and 3.46 times on a system running on a single computer with multiple GPUs as well as those using multiple computers with multiple GPUs. This indicates that the solvers are distributed on the platform for which the accelerating effect is significantly better than that arising from data and model parallelisms.

In addition, the solvers using the GradReduce method obtain an acceleration ratio of 2.03 and 3.84 over the system with a single computer with multiple GPUs as well as multiple computers running with multiple GPUs. The performance improvement effect is better than the Ring-AllReduce. It indicates that the GradReduce method is a distributed strategy that is suitable for the solvers of heat conduction partial differential equation inverse problems. It can be applied on heterogeneous systems with multiple CPUs and GPUs. The GradReduce technique invokes a linear programming routine, which can determine the optimal data communication scheme according to the prior data communication time-cost and is more suited to handling the training features associated with the large amount of data calculation needed in problems involving heat conduction and partial differential equation inverse problem solvers.

## V. Conclusion

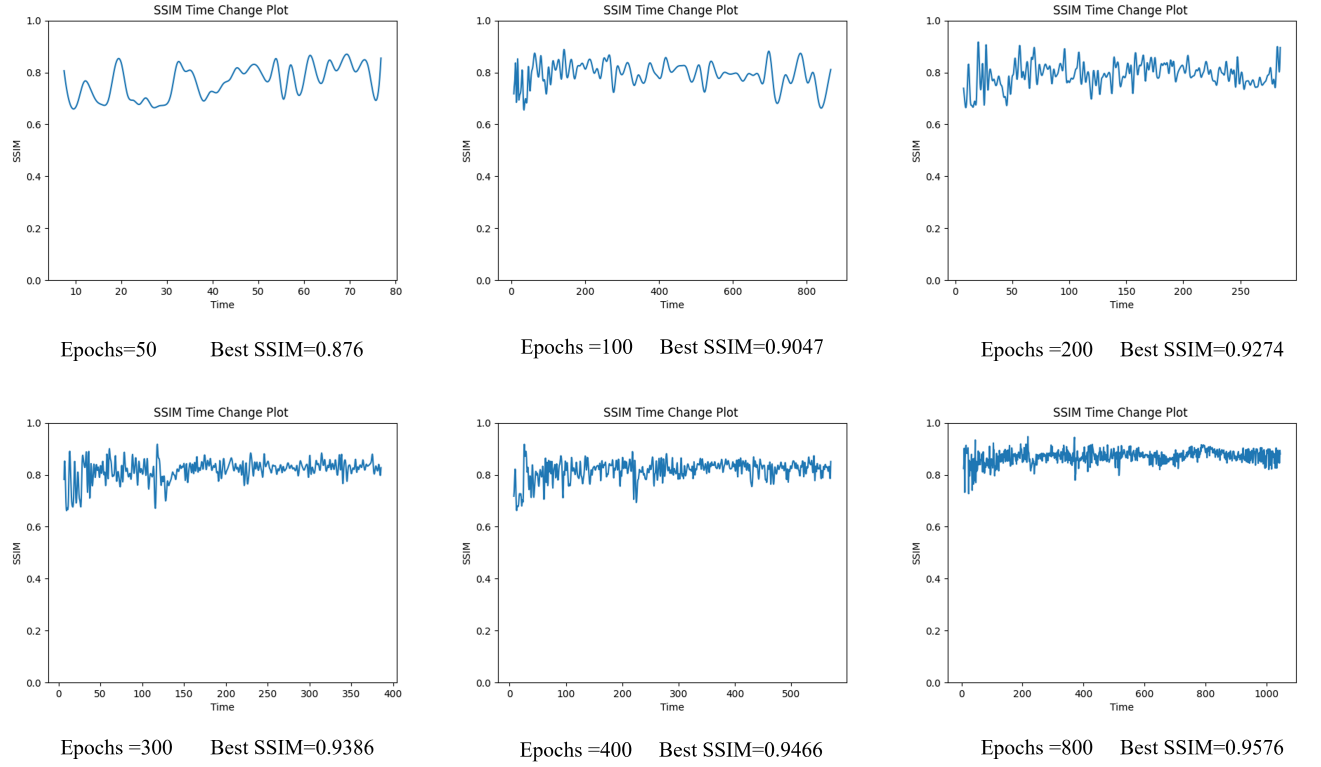
In this paper, a solver for the inverse problem of heat conduction partial differential equation is constructed based on the distributed deep learning method. The experiments show




Batchsize=512 Best SSIM=0.9205 Batchsize=256 Best SSIM=0.9274 Batchsize=128 Best SSIM=0.9356

Batchsize=64 Best SSIM=0.9305 Batchsize=32 Best SSIM=0.9554 Batchsize=16 Best SSIM=0.945

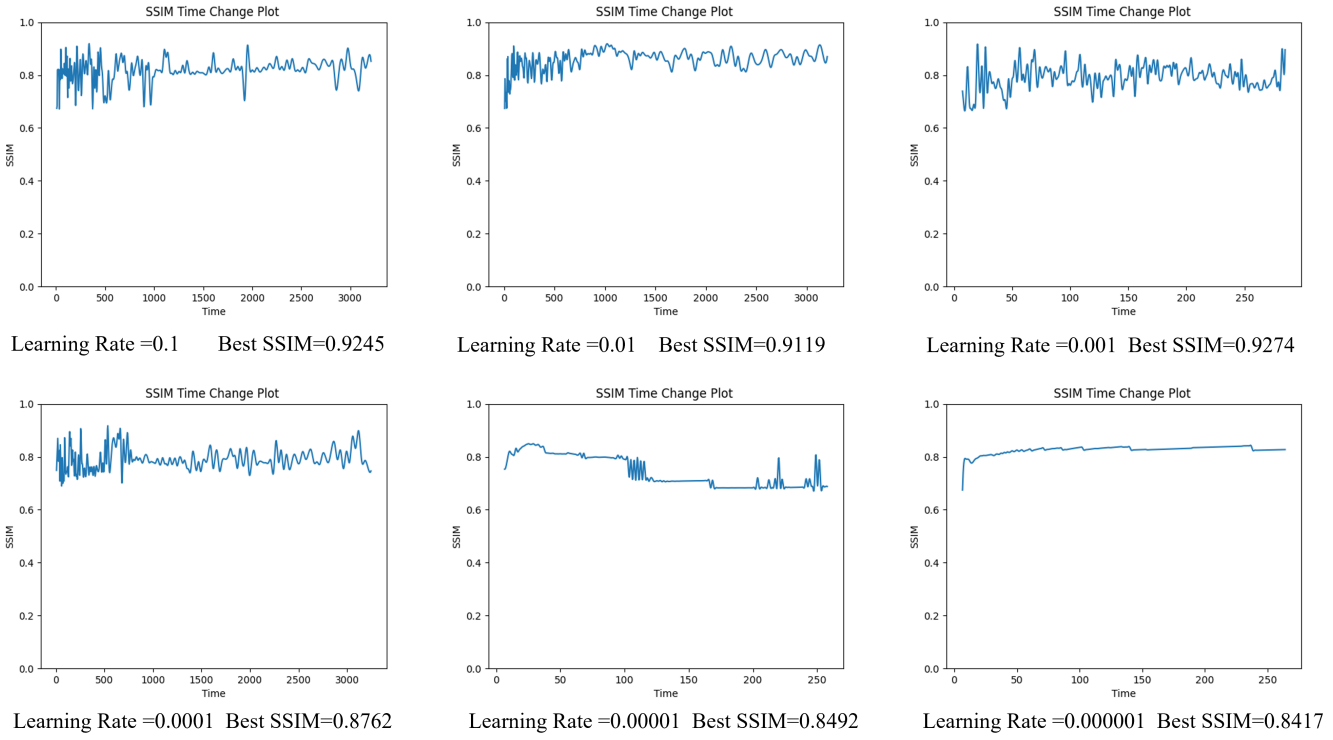
(a)The spatio-temporal variation of SSIM under different batch sizes.




Epochs=50 Best SSIM=0.876 Epochs =100 Best SSIM=0.9047 Epochs =200 Best SSIM=0.9274

Epochs =300 Best SSIM=0.9386 Epochs =400 Best SSIM=0.9466 Epochs =800 Best SSIM=0.9576

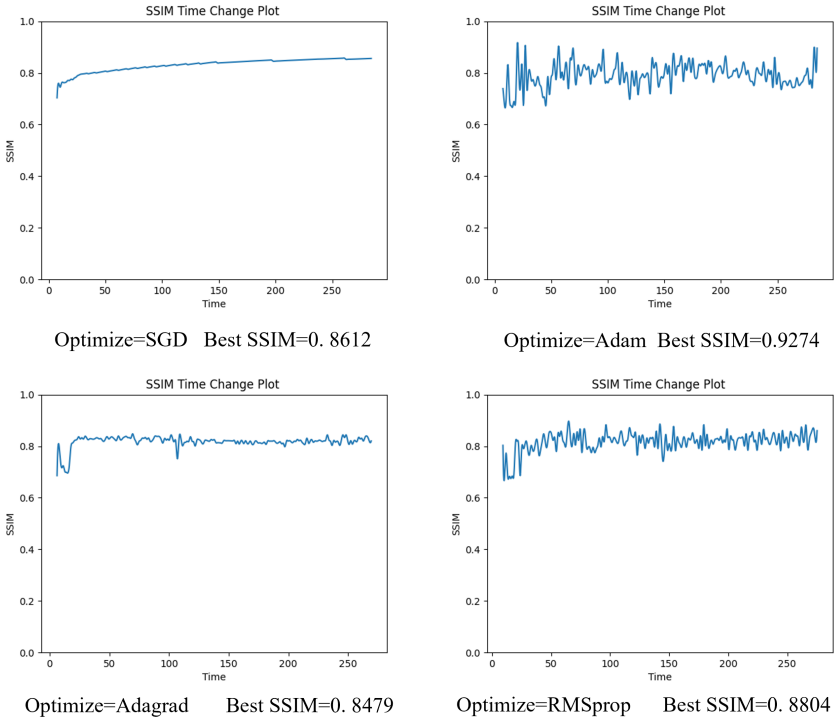
(b)The spatio-temporal variation of SSIM under different Epochs.




Learning Rate =0.1 Best SSIM=0.9245 Learning Rate =0.01 Best SSIM=0.9119 Learning Rate =0.001 Best SSIM=0.9274

Learning Rate =0.0001 Best SSIM=0.8762 Learning Rate =0.00001 Best SSIM=0.8492 Learning Rate =0.000001 Best SSIM=0.8417

(c)The spatio-temporal variation of SSIM under different Learning Rate.




Optimize=SGD Best SSIM=0. 8612 Optimize=Adam Best SSIM=0.9274

Optimize=Adagrad Best SSIM=0. 8479 Optimize=RMSprop Best SSIM=0. 8804

(d)The spatio-temporal variation of SSIM under different optimizer.

Fig. 4. Visualization of the impact of hyperparameter settings on SSIM with respect to spatiotemporal resolution.

TABLE IV
COMPARISON OF SSIM AND TRAINING TIME ON DIFFERENT HETEROGENEOUS SYSTEMS

| Heterogeneous system | Model parallelism | | Data parallelism | | Ring-Allreduce | | GradReduce | |
|---|---|---|---|---|---|---|---|---|
| | SSIM | Time(s) | SSIM | Time(s) | SSIM | Time(s) | SSIM | Time(s) |
| Single computer with single GPU | 0.938 | 532.849 | 0.9426 | 521.500 | 0.9417 | 532.917 | 0.9551 | 531.849 |
| Single computer with multi-GPUs | 0.9455 | 389.110 | 0.9374 | 323.198 | 0.9488 | 274.073 | 0.9519 | 261.350 |
| Multi-computers with multi-GPUs | 0.9424 | 373.048 | 0.9413 | 292.389 | 0.9416 | **153.9103** | 0.9583 | **138.328** |

that the SSIM of the image predicted is more than 94%, which can meet the needs of industrial application. Then, data parallelism and model parallelism methods are used to optimize the performance of the solver. The data parallelism and model parallelism can achieve an acceleration ratio of 1.61 and 1.36 in a single computer with multiple GPUs, respectively. Due to updated gradient data being merged in a single parameter server, the data parallelism cannot be invoked on multiple computers. The performance optimization of the model parallelism running on multiple computers with multiple GPUs is insignificant. Furthermore, we propose a GradReduce method that follows its ring data communication architecture, which optimizes its gradient transmission method and replaces the mechanical clock-frequency gradient transmission scheme with a linear programming gradient transmission scheme. The experimental results show that GradReduce method achieves an acceleration ratio of 3.84 on a heterogeneous system platform with two CPUs and four GPUs.

## REFERENCES

[1] Z. Brzeniak, G. Dhariwal, and Q. T. L. Gia, "Stochastic navier-stokes equations on a thin spherical domain," *Applied Mathematics Optimization*, vol. 84, DOI 10.1007/s00245-020-09702-2, no. 2, pp. 1971–2035, Jul. 2020. [Online]. Available: https://doi.org/10.1007%2Fs00245-020-09702-2

[2] S. Deng, Z. Li, and K. Pan, "An adi-yee's scheme for maxwell's equations with discontinuous coefficients," *J. Comput. Phys.*, vol. 438, DOI 10.1016/j.jcp.2021.110356, no. C, Aug. 2021. [Online]. Available: https://doi.org/10.1016/j.jcp.2021.110356

[3] I. J. Njoku, C. P. Onyenegecha, C. J. Okereke, E. Omugbe, and E. Onyeocha, "Solutions of schrodinger equation and thermodynamic properties of iodine and scandium fluoride molecules based on formula method," *Physica Scripta*, vol. 97, DOI 10.1088/1402-4896/ac4717, no. 1, p. 015201, Jan. 2022. [Online]. Available: https://dx.doi.org/10.1088/1402-4896/ac4717

[4] J. H. Gu, M. Hong, Q. Q. Yang, and Y. Heng, "A fast inversion approach for the identification of highly transient surface heat flux based on the generative adversarial network," *Applied Thermal Engineering*, vol. 220, p. 119765, 2023.

[5] I. E. Lagaris and A. Likas, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[6] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–9, 2000.

[7] F. Regazzoni, L. DedÃ¨, and A. Quarteroni, "Machine learning for fast and reliable solution of time-dependent differential equations," *Journal of Computational Physics*, vol. 397, DOI https://doi.org/10.1016/j.jcp.2019.07.050, p. 108852, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999119305364

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations," *CoRR*, vol. abs/1711.10561, 2017. [Online]. Available: http://arxiv.org/abs/1711.10561

[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations," *CoRR*, vol. abs/1711.10566, 2017. [Online]. Available: http://arxiv.org/abs/1711.10566

[10] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, 2018.

[11] A. D. Jagtap, Z. Mao, N. Adams, and G. E. Karniadakis, "Physics-informed neural networks for inverse problems in supersonic flows," *Journal of Computational Physics*, vol. 466, DOI 10.1016/j.jcp.2022.111402, p. 111402, Oct. 2022. [Online]. Available: https://doi.org/10.1016%2Fj.jcp.2022.111402

[12] H. Z. Z. M. Haolong Chen, Bo Yu, "Improved cuckoo search algorithm for solving inverse geometry heat conduction problems," *Heat Transfer Engineering*, vol. 40(3-4), pp. 362–374, 2019.

[13] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[14] H. Su and H. Chen, "Experiments on parallel training of deep neural network using model averaging," *CoRR*, vol. abs/1507.01239, 2015. [Online]. Available: http://arxiv.org/abs/1507.01239

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, DOI 10.1145/3065386, no. 6, May. 2017. [Online]. Available: https://doi.org/10.1145/3065386

[16] J. Dean, G. S. Corrado, R. Monga, K. Chen, and A. Y. Ng, "Large scale distributed deep networks," *Advances in neural information processing systems(NIPS)*, 2012.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[18] C. Kai and H. Qiang, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[19] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, DOI 10.1109/CVPR.2016.308, pp. 2818–2826, 2016.

[21] S. Pal, E. Ebrahimi, A. Zulfiqar, Y. Fu, V. Zhang, S. Migacz, D. Nellans, and P. Gupta, "Optimizing multi-gpu parallelization strategies for deep learning training," *IEEE Micro*, vol. 39, no. 5, pp. 91–101, 2019.

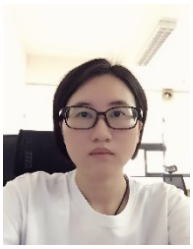
[22] A. Gibiansky, "Bringing hpc techniques to deep learning," https://pytorch.org/https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/, 2017.

[23] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *CoRR*, vol. abs/1802.05799, 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[24] K. N. Habib, "Convergence analysis for wave equation by explicit finite difference equation with drichlet and neumann boundary condition," *Science Publishing Group*, vol. 7, pp. 19–24, 2021.

[25] B. MacNeal and R. MacNeal, "Minimum constraints for finite element vector potential problems with neumann boundary conditions," *IEEE Transactions on Magnetics*, vol. 27, DOI 10.1109/20.105006, no. 5, pp. 4114–4117, 1991.

[26] R. W. Pryor, *Multiphysics Modeling Using COMSOL: A First Principles Approach*, 1st ed. USA: Jones and Bartlett Publishers, Inc., 2009.
[27] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, DOI 10.1109/TIP.2003.819861, no. 4, pp. 600–612, 2004.
[28] Z. Zhang, Q. Chen, J. Shen, X. Liao, and J. Zhou, "A sparse-view ct reconstruction method based on combination of densenet and deconvolution," *IEEE Transactions on Medical Imaging*, vol. 38, DOI 10.1109/TMI.2019.2898287, no. 7, pp. 1580–1591, 2019.
[29] Y. Chen, W. Yang, and W. Wang, "Deep convolutional neural networks for accelerated magnetic resonance imaging: Reconstruction and synthesis," *IEEE Transactions on Medical Imaging*, vol. 35, DOI 10.1109/TMI.2015.2466992, no. 1, pp. 207–219, 2016.
[30] S. Nah, T. H. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," *IEEE Transactions on Image Processing*, vol. 26, DOI 10.1109/TIP.2017.2723870, no. 9, pp. 4509–4522, 2017.
[31] C. Wang, C. Xu, C. Wang, and D. Tao, "Perceptual adversarial networks for image-to-image transformation," in *IEEE Conference on Computer Vision and Pattern Recognition*, DOI 10.1109/CVPR.2018.00109, pp. 979–987, 2018.
[32] W. Wang, X. Wu, X. Yuan, and Z. Gao, "An experiment-based review of low-light image enhancement methods," *IEEE Access*, vol. 8, DOI 10.1109/ACCESS.2020.2992749, pp. 87 884–87 917, 2020.
[33] M. Barros and M. Casquilho, "Linear programming with cplex: An illustrative application over the internet cplex in fortran 90," in *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, DOI 10.23919/CISTI.2019.8760632, pp. 1–6, 2019.
[34] S. T. H. H. W. T. S. Y. Rokoni A., Zhang L., "Learning new physical descriptors from reduced-order analysis of bubble dynamics in boiling heat transfer," *International Journal of Heat and Mass Transfer*, vol. 186, 2022.



**Zhuowei Wang** received the B.S. degree in computer science and technology from the China University of Geosciences, Wuhan, China, in 2007, and the M.S. and Ph.D. degrees in computer architecture from Wuhan University, Wuhan, in 2009 and 2012, respectively.

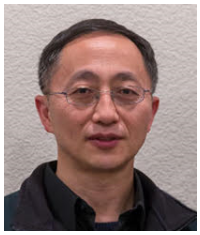
From 2019 to 2020, she worked as a Visiting Scholar with the Norwegian University of Science and Technology, Gjøvik, Norway. She is currently an Associate Professor with the School of Computers, Guangdong University of Technology, Guangzhou,China. Her research interests focus on high-performance computing, low-power optimization, and distributed systems.

**Le Yang** received the B.S. degree in network engineering from Guangdong University of Technology, Guangzhou, China in 2021. He is currently pursuing a Master's degree in Computer Technology at Guangdong University of Technology in Guangzhou, China. His research interests include parallel computing.

**Haoran Lin** received the B.S. degree in network engineering from Guangdong University of Technology, Guangzhou, China in 2020. He is currently pursuing the M.S degree in computer Technology with Guangdong University of technology, Guangzhou, China. His research interests include the parallel computing.

**Genping Zhao** received the Ph.D. degrees in Information and Telecommunications Engineering from Harbin Engineering University, Harbin, China, in 2017. From Dec. 2013 to Jan. 2015, she worked as a visiting PhD student in the University of Western Australia, Perth, Australia. From Nov. 2018 to Nov. 2019, she worked as a Post Doc in the University of Alberta, Edmonton, Canada. She is currently a lecturer with the Institute of Computers, Guangdong University of Technology, Guangzhou, China. Her research interests focus on multi-source remote sensing data analysis and machine learning.

**Zixuan Liu** received the B.E. Degree in computer science and technology from the university of Dalian Nationalities University, Dalian, China in 2015. She received the M.S. degree in computer science and technology from the university of Inner Mongolia University, Hohhot, China in 2019. She is currently pursuing the Ph.D. degree in computer science and technology with Inner Mongolia University, Hohhot, China. Her research interests include the theory and design of the SRT division.



**Xiaoyu Song** received the Ph.D. degree from the University of Pisa, Italy, in 1991. From 1992 to 1998, he was on the faculty at the University of Montreal, Canada. He joined the Department of Electrical and Computer Engineering at Portland State University in 1998, where he is now a Professor. He was an editor of IEEE Transactions on VLSI Systems and IEEE Transactions on Circuits and Systems. He was awarded an Intel Faculty Fellowship from 2000 to 2005. His research interests include formal methods, design automation, embedded systems and emerging technologies.