A fuzzy neural network based dynamic data allocation model on heterogeneous multi-GPUs for large-scale computations

ZHANG, Chaolong, XU, Yuanping, XU, Zhijie, HE, Jia, WANG, Jing <http://orcid.org/0000-0002-5418-0217> and ADU, Jianhua

DOI: XXX

A Fuzzy Neural Network based Dynamic Data Allocation Model on Heterogeneous Multi-GPUs for Large-Scale Computations

Chaolong Zhang[1,3] Yuanping Xu[1] Zhijie Xu[2,3] Jia He[2] Jing Wang[4] Jianhua Adu[1]

[1]School of *Software Engineering*, Chengdu University of Information Technology, Chengdu 610225, PRC
[2]School of *Computer Science*, Chengdu University of Information Technology, Chengdu 610225, PRC
[3]School of *Computing & Engineering*, University of Huddersfield, Queensgate, HD1 3DH, Huddersfield, UK
[4]Department of *Computing*, Sheffield Hallam University, S1 2NT, Sheffield, UK

Abstract: The parallel computation capabilities of modern GPU (Graphics Processing Unit) processors have attracted increasing attention from researchers and engineers who have been conducting high computational throughput studies. However, current single GPU based engineering solutions are often struggle to fulfill their real-time requirements. Thus, the multi-GPU-based approach has become a popular and cost-effective choice for tackling the demands. In those cases, the computational load balancing over multiple GPU "nodes" is often the key and bottleneck that affect the quality and performance of the runtime system. The existing load balancing approaches are mainly based on the assumption that all GPU nodes in the same computer framework are of equal computational performance, which are often not the case due to cluster design and other legacy issues. This paper presents a novel dynamic load balancing (DLB) model for rapid data division and allocation on heterogeneous GPU nodes based on an innovative fuzzy neural network (FNN). In this research, a 5-state parameter feedback mechanism defining the overall cluster and node performances is proposed. The corresponding FNN-based DLB model will be capable of monitoring and predicting individual node performance under different workload scenarios. A real-time adaptive scheduler has been devised to reorganize the data inputs to each node when necessary to maintain their runtime computational performances. The devised model has been implemented on two dimensional (2D) discrete wavelet transform (DWT) tasks for evaluation. Experiment results show that this DLB model has enabled a high computational throughput while ensuring real-time and precision requirements from complex computational tasks.

Keywords: Heterogeneous GPU Cluster, Dynamic Load Balancing, Fuzzy Neural Network, Adaptive Scheduler, Discrete Wavelet Transform.

1 Introduction

In the last decade, the powerful parallel computing capabilities of graphics cards and GPUs, originally driven by the market demands for real-time and high-definition game displays, have been widely accepted by the research communities. Large scale and data intensive computational applications such as areal surface characterization filtration, visual recognition, and natural language processing (NLP), have been benefitted by this new-found and cost-effective computational powerhouse. It has also attracted increasing attentions from researchers across the globe in devising general hardware-based acceleration models for real world engineering challenges [1–3]. Leading the trend, in 2007, NVIDIA released the Compute Unified Device Architecture (CUDA) - a software framework that aimed at unifying the efforts in harnessing the GPU powers for general-purpose usages and serious applications. It has greatly simplified the GPU programming practices as well as embracing the inherent data parallelism from GPU architectures. The toolkit has significantly enhanced the performances of some of the most common data and signal processing functions such as Fast Fourier Transform (FFT), Gaussian filtering, and discrete wavelet transform (DWT) that are widely used in applications such as face detection, DNA sequencing, and more recently, machine learning systems such as convolutional neural networks [4–6].

CUDA provides a scalable and integrated programming model for allocating and organizing processing threads and mapping them onto the computer hardware infrastructure equipped with dynamical adaptation ability for all mainstream GPU architectures [7]. In addition CUDA has linked and embedded a series of interfaces and APIs to assist direct programming on GPUs instead of relying on various graphics APIs (e.g. OpenGL) like in the so-called "GPGPU" era. CUDA treats GPU as a standalone parallel computational device that can realize data processing algorithms by using C/C++-like programming routines and functions that are familiar to mainstream programmers and researchers.

Previous related works on parallelizing processes and data were mainly achieved through using a single GPU that had witnessed a moderate performance gain across board. However, due to the limitation of data storage format and space (memory), as well as the fixed number of data streams available on a single GPU, previous works are often struggling to fulfill the real-time requirements from many large scale computational applications, which is especially problematic for the latest deep learning applications that often require to process data sets with tens of gigabytes (GB) in size (e.g. ImageNet) [8–10], never mention the online areal surface metrological tasks for processing measured surface texture data engaging complex filtrations with a large amount of numerical parameters (e.g. processing a data

Regular paper
Manuscript received date; revised date
This work is supported by the National Natural Science Foundation of China (No.61203172), the STD of Sichuan (2017JY0011 and 2014GZ0007), and Shenzhen STPP (GJHZ20160301164521358).

set with multi-level DWT) [11, 12]. Thus, in comparison, Multi-GPUs based acceleration solutions can be flexible and to achieve higher performance with relatively low hardware costs. Numerous computational-intensive issues that cannot be resolved by using the single GPU model have been making steady progress in the context of multi-GPUs, e.g., multi-GPUs based FFT (Fast Fourier Transform) [13] and Gaussian Filtering [14]. In the meantime, several multi-GPUs based programming libraries (e.g. MGPU) [15] and MapReduce libraries (e.g. GPMR and HadoopCL) [16, 17] have been developed by researchers in the field.

It is a challenging task to fully utilize the parallel computational power of multiple and interconnected GPU nodes [18], which is especially true for the heterogeneous multi-GPU systems. Unbalancing load problem may cause low computational performance. To solve this problem, the load balancing models that can intelligently allocate tasks to individual GPU node becoming the key issue. Chen et al. [19] proposed a task-based DLB solution for multi-GPU systems that can achieve a near-linear speedup with the increasing number of GPU nodes. Acosta et al. [20] had developed a DLB functional library that aims to balancing the load on each node. However, these pilot studies are base according to the corresponding system runtime performance on the assumption that all GPU nodes equipped in a multi-GPU platform has equal computational capacity. In addition, task-based load balancing schedulers that these approaches have relied upon often fall short to support applications with huge data throughputs but limited processing function(s) - there are very few “task” to schedule, e.g. DWT. These applications need more attention in refining the task partition in each computational iteration taking into account of the data locality [18]. In terms of data parallelism based load balancing schedulers, Acosta et al. presented a DLB model that dynamically balances the workload using information established by the first iteration of the computation [21], which failed to respond to the information changes during the later computational iterations. In contrast, the strategies developed by Boyer et al. and Kaleem et al. collect system information during the system runtime [18, 22, 23], so they can support the dynamic load balancing scheduling demands according to the real-time feedbacks, which consolidates the foundation for this study.

To optimize the load balancing problem among multi-GPU nodes for large scale applications with highly repetitive computational procedures or iterations, this paper presents a novel DLB model based on fuzzy neural network (FNN) and data set division techniques for heterogeneous multi-GPU systems, and this study is extended from our previous publication [24]. In this study, five real-time state feedback parameters closely relating to the computational performance of every GPU node are defined. They are capable of predicting the relative computational performance of each GPU node during system runtime. Using the constructed FNN and the devised advanced data distribution method, a large data set can be adaptively divided to enhance the overall utilization of all hidden computing powers from a heterogeneous multi-GPU system.

The rest of this paper is organized as follows. Section 2 presents a brief review over the preliminaries and related works in the field. Based on the literatures, the rationales of this research are justified; then, the proposed FNN DLB model for multi-GPUs is explored and its features discussed in Section 3. Section 4 constructs a case study that demonstrates how to improve the computational performance of the lifting scheme of DWT by using the devised model. Section 5 provides the test results of the design and evaluations. Finally, the Section 6 concludes the research with future works.

2 Related Studies

2.1 GPU architectures and process model

Modern GPUs are not only powerful graphics engines, but also highly parallel arithmetic and programmable processors. More significantly, in 2007, NVIDIA introduced the Tesla architecture, which was the first unified graphics and computing architecture. After that, NVIDIA released series of GPU architectures, i.e. Fermi, Kepler, Maxwell, Pascal, and most recently, the Volta architecture. All GPU cards produced by NVIDIA in the last decade are based on these architectures. In the point of view of the hardware architectures, all these models are similar but with incremental improvements on memory sizes and their accessibility, number of data streams or CUDA cores, as well as the overall processing powers. For example, the number of and the so-called special-function units (SFU), and streaming multiprocessors (SM) that each contains multiple stream processors (SP) - the CUDA cores. Modern GPU architectures are based on a scalable processor array formed by SPs that provides a high performance parallel computing platform.

CUDA is a parallel programming framework that was designed especially for general purpose computing, and it greatly simplifies the GPU programming practices. CUDA adopts a SPMD (Single Program, Multiple Data) programming model and provides a sophisticated memory hierarchy (i.e. register, local memory, shared memory, global memory, texture memory and constant memory, etc.). Hence, a GPU can achieve high data parallel computation through elaborately designed CUDA codes empowered by the efficient usages of different memories according to the respective data features, including access mode, size and format.

The computational capacity of a single GPU can sometimes satisfy the computational demands of numerous applications, for example in the conventional image filtering and other transformation processes. However, it is still falling short of processing some complex tasks engaging massive data sets, for example in video indexing and visual recognition, due to its limited memory space, instruction length, and execution loops. One perceived solution is to deal with large volume data sets in distributed processing mode on multi-GPUs. At present, there are two representative categories of multi-GPU platforms, the standalone computer type (a single CPU node with multiple GPU processors); and the cluster type (multiple CPU nodes and each accompanied by one or more GPU processors). In general, the cluster computer systems require more complex communication and data transmission due to their commonly adopted PCI-E (Peripheral Component Interconnect Express) architecture and network connections. Thus, the standalone computers have been chosen for the study in this research.

2.2 Fuzzy neural network

Artificial neural network (ANN) is a branch of artificial intelligence (AI) that was first inspired by the "understanding" of how human brain works to process data and summarizes patterns. Contrast with traditional methods that have to extract features from input data in a rigid and almost mechanical manner, ANN based models can automatically find features from training data, which are called "learning from data". One of successful applications relating to ANN is Deep Learning (DL) based on a process model called Deep Neural Network, e.g., Krizhevsky presented the AlexNet to classify images in ILSVRC2012 (the ImageNet Large-scale Visual Recognition Challenge) and achieved a winning performance with the test error rate at 15.3% [8]. AlexNet is considered as the first successful DL model. Later, in 2015, He presented a new DL model, ResNet, that won the ILSVRC 2015 with an incredible error rate at 3.6% [25].

Generally speaking, traditional fuzzy systems are built on IF-THEN rules (i.e. fuzzy rules) which are acquired from experimental knowledge of domain experts. Fuzzy systems can solve complex decision-making issues when equipped with abundant fuzzy rules [26]. Li et al. designed a fuzzy keyword search engine based on a fuzzy system for searching encrypted data over cloud sources, and it solved the drawback of traditional techniques that struggled to match keywords on cloud [27]. Krinidis et al. had improved the fuzzy C-Means (FCM) algorithm and presented fuzzy local information C-Means (FLICM) algorithm based on fuzzy set theory for image clustering. Compared with FCM, FLICM is more effective and efficient, which provides robustness to noisy images clustering [28].

Both fuzzy theory and ANN have been widely used in decision-making applications. However, the main problem of traditional fuzzy systems is that it is very difficult to find experts who can extract and summarize knowledge from their experiences, and extracted IF-THEN rules are usually not objective, which means that traditional fuzzy models are lacking of flexibility and robustness. Furthermore, the ANN models are still inadequate in representing the expert experiences. To solve these shortcomings, fuzzy neural network (FNN) was developed to combine the fuzzy rule based fuzzy systems and ANNs. Thus, ANN models have been merged into fuzzy systems to improve their efficiency and accuracy, such that FNN was envisaged to be a promising model [29]. Kuo et al. proposed a FNN based decision support system of intelligent suppliers which is able to consider both the quantitative and qualitative factors [30]. Chen et al. used FNN to approximate unknown functions in stochastic systems, which not only reduced the online computation load, but also achieved significant performance enhancement for fuzzy control algorithm [31].

Fuzzy theory and ANN based load balancing approaches have been widely used in traditional multi-CPU systems, i.e., distribution systems, data centers, and cloud computing applications, etc. Saffar et al. presented a fuzzy optimal reconfiguration approach that combines fuzzy variables and ant colony search method to balance the workloads on distribution systems [32]. Susila et al. developed a fuzzy based firefly approach for dynamical load balancing purpose in cloud computing systems [33]. Toosi et al. proposed a fuzzy logic based DLB model for cost and energy efficient purpose [34]. These prior works have inspired the motivation of adopting FNN for multi-GPU load balancing applications investigated in this study.

In summary, based on the achievements of previous related works, it is anticipated a feasible way to solve the DLB issue by adopting the FNN model. This study explores and implements a novel data-oriented load balancing model by devising a FNN framework for large data sets with simple iterative tasks on heterogeneous multi-GPU systems.

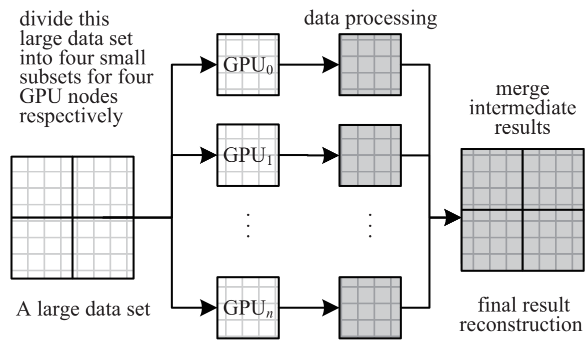


Fig. 1 A traditional load balancing model based on the pure data division method

2.3 Conventional Multi-GPU strategies

Fig.1 demonstrates a traditional load balancing model based on the pure data set division method [2], and it contains: 1) a large raw data set is divided into n small chunks (subsets) (n is equal to the number of GPU nodes in a targeted multi-GPU system), and each data chunk is distributed to a GPU node respectively; 2) each GPU node processes the corresponding subset; 3) the final results can be generated after merging the outputs of each GPU node. This approach is very simple and useful, however, it is likely to cause unbalancing load problem when the multi-GPU system contains different type of GPU hardware with unequal computational performances, known as heterogeneous multi-GPU platforms. As a result, the overall performance of a multi-GPU platform is restricted to the GPU node that has the lowest computational capability due to delayed merging process.

In a heterogeneous multi-GPU system, there are different types of GPUs having unequal computational performances, e.g., the multi-GPU workstation used in this study has two GPU cards C a middle-low-end (NVIDIA GTX 750 Ti) and a high-end GPU (NVIDIA GTX 1080). As the traditional data division method is still struggling to support heterogeneous multi-GPU systems, Acosta et al. developed a DLB library (named ULL_Calibrate_lib) for heterogeneous systems aiming to solve the task allocation problem [21]. ULL_Calibrate_lib can balance tasks dynamically to adapt system conditions during execution. This approach shows sound results for iterative operations, but performs less well when dealing with applications of large data throughputs with limited processing instructions – the too "few task" problem for the scheduler. For example, in surface metrology, metrologists often apply DWT functions to extract the surface texture characteristics from large volume of measured data [35]. Thus, in these cases, a data-oriented load balancing model is more suitable than the task-focused ones.

Boyer et al. explored a data-oriented DLB approach that supports GPU programs having a few kernels to process large volume data set iteratively. The main idea of Boyer's work is to predict the potential computational performance of each GPU node, and then divide the remaining data set according to the execution time of each GPU node for processing its previous assigned data set: 1) the host function sends initial small data chunks respectively for each GPU node and launches the corresponding kernels of each GPU. Assuming there are two GPU nodes in a multi-GPU system, a small data chunk is D, t_i indicates the processing time of the i^{th} GPU and C_i is the corresponding potential performance of the i^{th} GPU, then

$$C_i = \frac{D}{t_i} \tag{1}$$

2) the host function divides the remaining data for each GPU node. Let W be the remaining data set to be scheduled and W_i indicates the data set for i^{th} GPU, then

$$W = \sum_i W_i \tag{2}$$

In the balanced situation, all GPU nodes should finish their computations at the same time satisfying the following equation:

$$C_1 W_1 = C_2 W_2 \tag{3}$$

According to equations (1) and (3), W_i can be given as following:

$$W_1 = \frac{t_1 W_2}{t_2} \tag{4}$$

One of the drawbacks of this load balancing model is that it is disputable whether the initial execution time can accurately predict the real computational ability of a GPU. More specifically, a modern GPU card can have hundreds or even thousands of CUDA cores, e.g., NVIDIA GTX 750 Ti contains 640 cores, and NVIDIA GTX 1080 has 2560 cores. As a result, a small data set may cause a low GPU utilization rate, which causes the inaccurate performance prediction. For instance, in this study, we tested and evaluated the execution time for processing a small surface measurement data set by using DWT on these two GPUs respectively, experimental result shows that the processing time of these two GPUs are almost the same because both of them cannot fully use their hardware resources as there are not enough data to process. In this case, the data allocated on each GPU node will be in the same size by using equation (4), which is no difference with the pure data set division method (see Fig.1). In addition, these previous load balancing models failed to respond to the fluctuation of computational performance that is frequently occurred on multi-GPU systems in the real world.

The proposed DLB model in this paper aimed at predicting the computational performance according to the real hardware conditions rather than testing the processing time with a small data set, such that it improves the accuracy of performance prediction and supports real-time response to the fluctuation of computational performance.

3 Load Balancing on Heterogeneous Multi-GPU Systems

3.1 DLB idealism

To solve the load unbalancing problem and to respond to the fluctuation of computational performances from a heterogeneous multi-GPU system, this paper presents a novel DLB model for optimizing the overall parallel computational performance of large scale data computations on multi-GPU systems while ensuring the good price-performance ratio based on the FNN and dataset division method. In this model, the original data set is divided into several equal-sized data units and these data units are organized into n groups (n is equal to the number of GPU nodes in a specific multi-GPU platform) by using the scheduler, see Fig.2. The number of data units assigned to each GPU node is different, and it is determined by the real-time feedbacks (e.g. real-time computational performance and states of each GPU node) of a single GPU node. Thus, the purpose of data-oriented DLB model is to minimize the overall processing time by dynamically adjusting the number of data units in a group for each GPU node at runtime according to real-time state feedbacks of each GPU node.

3.2 Model and workflow

To describe the relationship between the real-time state feedback parameters and the number of data units assigned in a group to be "pushed" to a node, this model defines the relative computational ability CP_i^n to represent the n^{th} predication of real-time computational performance of i^{th} GPU node, and the scheduler and P_i^n is defined as following:

$$CP_i^n = f(\frac{D_{unit}}{T_i^{unit}}), CP_i^n \in [0,1], n = 0, 1, 2, ... \tag{5}$$

where D^{unit} is a data unit, T_i^{unit} is a feedback parameter denoting the actual processing time of D_{unit} by the i^{th} GPU node, and $f(x)$ is a normalization method.

In the ideal load balancing situation, all GPU nodes in a multi-GPU system would finish their respective work at the same time, this idea is the same as Boyer's model (see equation (3)), and it satisfies the following equation:

$$\begin{aligned} &T_1 = T_2 = ... = T_m \\ &\Rightarrow T_1^{unit} \times W_1 = T_2^{unit} \times W_2 = ... = T_m^{unit} \times W_m \end{aligned} \tag{6}$$

where T_i is the total processing time of i^{th} GPU node in a parallel computational task and W_i is the count of current workload (i.e. the current number of data units) for i^{th} GPU node. According to the equations (5) and (6), the number of data units can be calculated. Taking two GPU nodes as an example, $T_1 = T_2$, then:

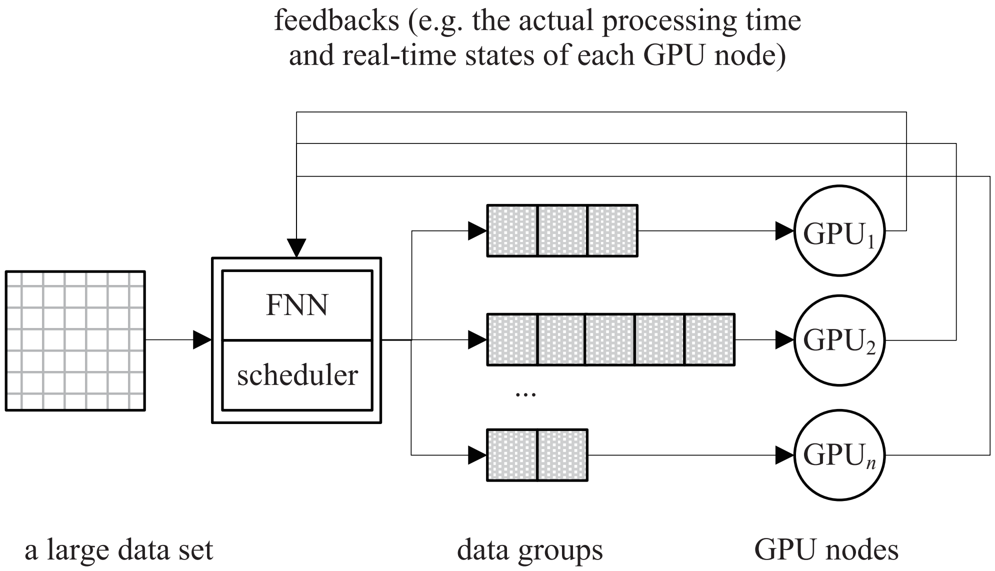


Fig. 2 The overall framework of the proposed data based DLB model

$$
\begin{aligned}
&T_1^{unit} \times W_1 = T_2^{unit} \times W_2 \\
&\Rightarrow W_1 = \frac{T_2^{unit} \times W_2}{T_1^{unit}} \Rightarrow W_1 = \frac{{P_2}^n \times W_2}{{P_1}^n}
\end{aligned} \tag{7}
$$

The same calculation method can be extended to multiple GPU nodes by using equation (7). Based on equations (5, 6, 7), the complete procedure for dynamically calculating the number of data units for every GPU node in any multi-GPU platform during runtime can be defined as: 1) This DLB model conducts the initial prediction to get P_i^0 for every GPU node by using the FNN structure defined in this model after acquiring the original data set (see Fig.2 and Fig.3); 2) The scheduler calculates the number of data units for each data group according to P_i^0 by using equation (7); 3) The multi-GPU platform begins the target parallel computational task when every GPU node gets the corresponding data group organized by the scheduler, and the FNN collects state feedbacks dynamically to prepare the next predication under certain state; 4) Once a GPU node has finished its data processing while others are not, the model estimates the remaining time (T_i^r) for each GPU node by using equation (8).

$$T_i^r = T_i^{unit} \times \left(W_i - W_i^{'}\right) \tag{8}$$

(where W_i' is the finished workload of the i^{th} GPU node.); 5) The data group reorganization is required when remaining time of any GPU node exceeds the threshold preset by this model, such that the next predication is required to get P_i^1; 6) The scheduler reorganizes the remaining data groups for all GPU nodes respectively according to P_i^1; 7) The step2-6 maintain a complete iteration that will be repeated until that all GPU nodes finish their workloads at the same time or the remaining time for every GPU is under the threshold (i.e. satisfying the equation (6)).

According to equation (7), it is convenient to divide data units and organize data groups for each GPU node when P_i^n or T_i^{unit} are given. Unfortunately, P_i^n or T_i^{unit} can be given only when the whole data processing task is finished. Therefore, precise prediction of P_i^n is the key factor of the devised model.

3.3 A FNN-driven mechanism

To predict P_i^n for each GPU node, this research has explored in depth the fundamentals of fuzzy theory and defined a 5-state feedback (the fuzzy sets) parameters namely: the floating-point operations performance (F), global memory size (M), parallel ability (P), the occupancy rate of computing resources of a GPU (UF) and the occupancy rate of global memory of a GPU (UM). Each fuzzy set defines the "high" and "low" fuzzy subsets. Likewise, the n^{th} relative computational ability P_i^n is also fuzzed as "high" and "low". All fuzzy sets and subsets are listed in Table 1.

Table 1 The defined fuzzy sets and subsets

Sets	Descriptions	Fuzzy subsets	Descriptions of Fuzzy subsets
F	The floating-point operations performance	FL	Low
		FH	High
M	Memory size	ML	Low
		MH	High
P	Parallel ability (a positive correlation with the count of processor cores of a GPU node)	PL	Low
		PH	High
UF	The occupancy rate of computing resources	UFL	Low
		UFH	High
UM	The occupancy rate of global memory	UML	Low
		UMH	High
CP	The fuzzy relative computational ability	CPL	Low
		CPH	High

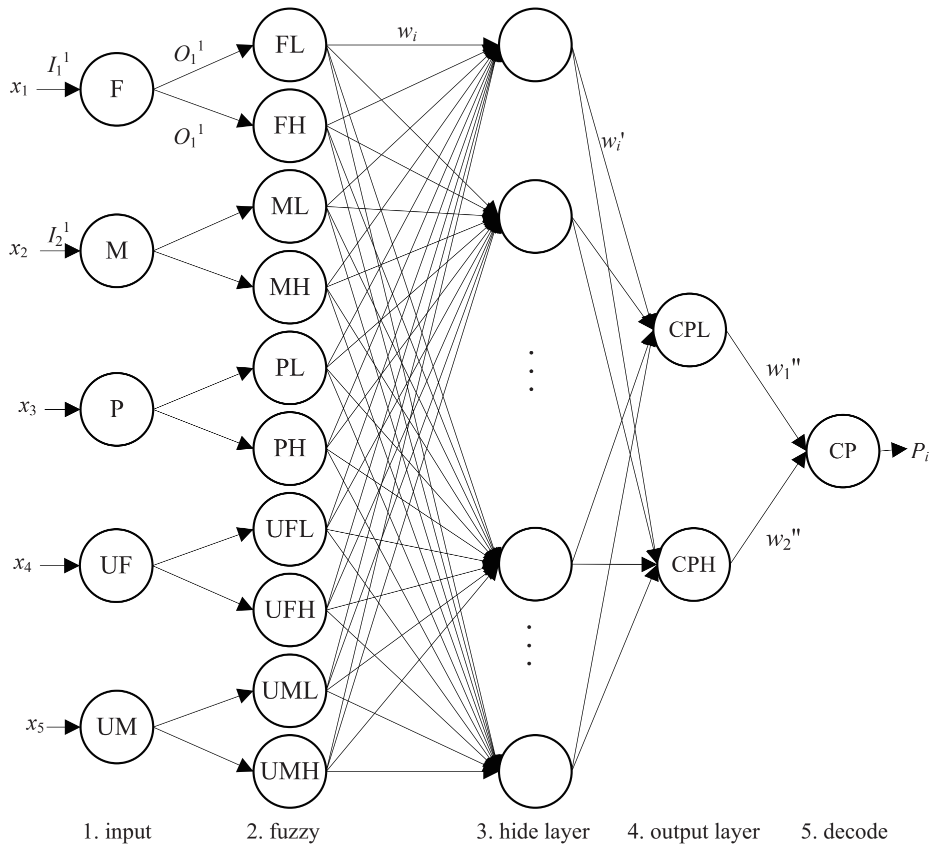


Fig. 3 The structure of FNN in the proposed data-oriented DLB model

After defining the fuzzy subsets, this research has designed a network structure of FNN that combines theories of the fuzzy mathematics and the back propagation mechanism from ANN to predict P_i^n of each GPU node before activating the scheduler to organize the data groups, see Fig.3. The first layer of the design is an input layer while the second layer, third layer and fourth layer are considered to be the fuzzy input layer, hide layer and output layer respectively in the classic structure of back propagation networks. This FNN structure has ten fuzzy truth values as inputs and two fuzzy truth result values as outputs. The final layer (i.e. fifth layer) decodes the fuzzy truth values to the correct value which is the actual P_i^n of i^{th} GPU's the n^{th} predication. The devised FNN uses I_i^j to denote the input of the i^{th} artificial neuron in the j^{th} level layer, O_i^j to denote the output of the i^{th} artificial neuron in the j^{th} level layer, w^i to denote weights of connections between the second and third layer, w_i' to denote weights of connections between the third and fourth layer, and w_i'' to denote weights of connections between the fourth and fifth layer (see Fig.3). The workflows of the corresponding inputs and outputs are illustrated in Fig.3.

Input layer: The input layer collects real-time states of a GPU node and generates values of the five state feedback parameters (see Table 1) as inputs when a predication of P_i^n is required. The input layer merely import real-time state feedback parameters into the FNN, and the input-output formula shows as the following:

$$O_i^1 = I_i^1 = x_i \tag{9}$$

where x_i is corresponding to the values of F, M, P, UF and UM in Table 1 respectively.

Fuzzy layer: The fuzzy layer transforms the correct values into fuzzy truth values by using a membership function. The input and output formulas are illustrated as the following:

$$\begin{gathered} I_i^2 = O_i^1 \\ O_i^2 = u_A\left(I_i^2\right), O_i^2 \subset [0,1] \end{gathered} \tag{10}$$

where $u_A(x)$ is the membership [31]. There are a lot of membership functions available, but this research chose the sigmoid function due to its "S" shaped curve can gracefully reflect the fluctuations of computational performance of GPU nodes [36]. The equation of sigmoid membership function is defined as the following:

$$f(x) = \frac{1}{1+e^{-a(x-c)}} \tag{11}$$

where a and c are constants having different values for different fuzzy subsets. Taking the occupancy rate of computing resources of a GPU node (UF, and $UF \in [0,1]$) as an example, this model takes $a = -15$ and $c = 0.5$, and $a = 15$ and $c = 0.5$ to transform a correct value of UF into its fuzzy truth values of UFL and UFH respectively, so the membership functions of UFL and UFH can be defined as:

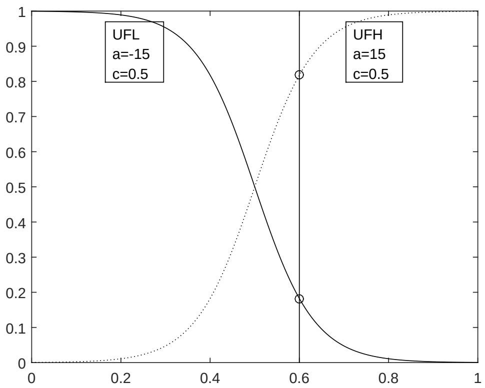


Fig. 4 The Membership Functions of UFL and UFH

$$\begin{cases} UFL : u_{UFL}\left(UF\right) = \frac{1}{1+e^{15\times(UF-0.5)}} \\ UFH : u_{UFH}(UF) = \frac{1}{1+e^{-15\times(UF-0.5)}} \end{cases} \tag{12}$$

According to equation (12), for instance, when a GPU's $UF = 0.6$, the membership value of UFL is 0.18, and the membership value of UFH is 0.82, see Fig.4.

Hide layer: In principle, the more the hidden layers, the more complex functions can be fitted. However, it also may cause the disadvantages of a mass of computation and overfitting. Generally speaking, a single hidden layer can meet majority requirements for prediction purposes [36]. Thus, this load balancing model has only one hidden layer. The input and output formulas are defined as the following:

$$\begin{gathered} I_i^3 = \sum_{j=1}^{n} w_i O_j^2 - \theta_i \\ O_i^3 = \varphi(I_i^3) \end{gathered} \tag{13}$$

where O_j^2 (n=10) denotes outputs of 10 artificial neurons on the 2nd level layer, and θ_i is a threshold value while $\varphi(x)$ is the activation function used by the artificial neurons. This research has chosen a sigmoid function as the activation function:

$$\varphi(x) = \frac{1}{1+\exp(-ax)} \tag{14}$$

Output layer: The output layer generates fuzzy truth values of the "high" and "low" fuzzy subsets of P_i^n. The input and output formulas are defined as the following:

$$\begin{gathered} I_i^4 = \sum_{j=1}^{m} w_j{}' O_j^3 - \theta_i' \\ O_i^4 = \varphi\left(I_i^4\right) \end{gathered} \tag{15}$$

where m is the number of artificial neurons on the hide layer (i.e. 3^{rd} level layer), θ_i' is a threshold value while the definition of $\varphi(x)$ is the same as equation (14).

Decode layer: The decode layer is added in this network to transform the fuzzy truth values of the CPL and CPH into the correct value of P_i^n by using the fuzzy weighted average method. The input and output formulas are defined as the following:

$$\begin{gathered} I^5 = \sum_{i}^{2} w_i{}'' O_i^4 \\ P_i = O^5 = I^5 \Big/ \sum_{i=1}^{2} O_i^4 \end{gathered} \tag{16}$$

Based on the FNN structure illustrated in Fig.3, the proposed load balancing model can be learned by training data using the back propagation algorithm that is collected from historical data of real-time state feedbacks (e.g. data processing time and a GPU states at some point). After the model is trained, it can be used to predict P_i^n, and then the scheduler can organize the data groups dynamically according to equation (7).

4 A Case Study

This data-oriented DLB model supports a wide variety of large scale data computations. This research explores the LWT (Lifting Wavelet Transform - LWT) computation for huge metrological data sets of surface textures as a case study to evaluate the validity and efficiency of the proposed model. Rooted in DWT, which is one of the fundamental algorithms for filtration widely used in surface metrology, signal and image processing, biomedicine visualization, and machine vision, LWT aims to improve the computational efficiency through a lifting scheme, also referred as the second generation wavelet [37].

The 1D forward LWT contains four operative steps: split, predict, update, and scale [37].

Split: This step splits the original signal into two subsets of coefficients, i.e. *even* and *odd*, and the former one is corresponding to the even index values while the latter is corresponding to the odd index values. The split method is expressed as equation (17), and it is also called the lazy wavelet transform.

$$\begin{cases} even[i] = X[2i] \\ odd[i] = X[2i+1] \end{cases} \tag{17}$$

Predict: The *odd* coefficients can be predicted from the even by using prediction operator P, and then replace the old *odd* values by the prediction result as the next new *odd* coefficients recursively. This step can be expressed as equation (18).

$$odd = odd - P(even) \tag{18}$$

Update: Likewise, *even* coefficients can be updated from the update operator U, and then replace the old *even* values by the updated result as the next new *even* coefficients recursively. This step shows as equation (19).

$$even = even + U(odd) \tag{19}$$

Scale: Normalize *even* and *odd* coefficients with factor K respectively by using equation (20) to get the results of *evenApp* and *oddDet*, which are the final approximation coefficients and detail coefficients of forward LWT respectively.

$$\begin{cases} evenApp = even \times (1/K) \\ oddDet = odd \times K \end{cases} \tag{20}$$

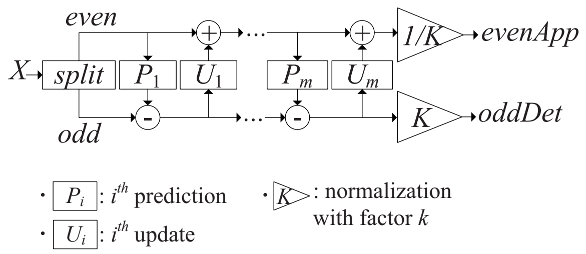


Fig. 5 Main computational procedure of single-level 1D forward LWT

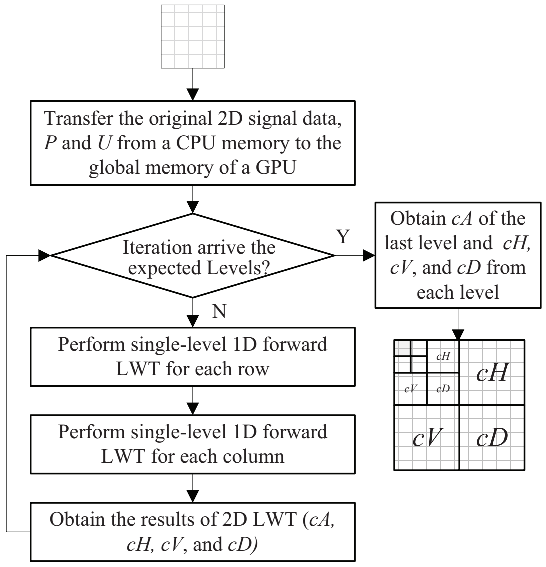


Fig. 6 Main computational procedure of multi-level 2D forward LWT

Table 2 The single-level forward LWT based on CDF (9, 7) wavelet

Split:	$\begin{cases} even[i] = X[2i] \\ odd[i] = X[2i+1] \end{cases}$
1^{th} Predict:	$odd[i]- = -\alpha \times (even[i] + even[i+1])$
1^{th} Update	$even[i]- = -\beta \times (odd[i] + odd[i-1])$
2^{th} Predict	$odd[i]- = -\gamma \times (even[i] + even[i+1])$
2^{th} Update:	$even[i]- = -\delta \times (odd[i] + odd[i-1])$
Scale:	$\begin{cases} even = even \times \varepsilon \\ odd = odd \times (1/\varepsilon) \end{cases}$

The inverse LWT with a lifting scheme is achieved by inverting the complete sequence of operation steps of forward LWT and switching the corresponding addition and subtraction operators. With the lifting scheme, the computational results of both forward and inverse LWT for arbitrary wavelet can be obtained through applying several steps of prediction and update operations and the final normalization with factor K, where P_i and U_i represent the i^{th} prediction and update coefficients respectively (see Fig.5). For a multi-level DWT, the computational process is repeatedly applied to the approximation coefficients until a desired number of decomposition levels are reached.

```
void lwt(raw[][]) {

 // allocating GPU memory
 // transfer data to the global memory of a GPU
 cudaMemcpy(d_raw, raw, size,
      cudaMemcpyHostToDevice);

 // Split data on GPU
 gpu_split(d_even, d_odd, d_raw);
 gpu_lwt_predict(d_even, d_odd, [-alpha, -alpha
      ]); // 1th Predict
 gpu_lwt_update(d_even, d_odd, [-beta, -bate]);
         // 1th Update
 gpu_lwt_predict(d_even, d_odd, [-gamma, -gamma
      ]); // 2th Predict
 gpu_lwt_update(d_even, d_odd, [-deta, -deta]);
         // 2th Update
 gpu_scale(d_even, d_odd, phi); // Scale

   // transfer the LWT result from the global
        memory of a GPU
   // to CPU memory
   cudaMemcpy(evenApp, d_even, size,
        deviceToHost);
   cudaMemcpy(oddDet, d_odd, size, deviceToHost
        );
}
```

Algorithm. 1 The scheduling software routine on a GPU node

In the case of a 2D DWT, it simply needs to perform the horizontal 1D LWT for each row of a 2D input data set and the vertical 1D LWT for each corresponding column in sequence separately due to a 2D LWT can be realized through the 1D wavelet transform along its x- and y-axis, such that we can obtain the 2D LWT results: cA, cH, cV and cD; cA is approximation coefficients while cH, cV and cD indicate detail coefficients along horizontal, vertical and diagonal orientations respectively. Fig.6 illustrates the main computational procedure of a multi-level 2D forward LWT.

Lifting scheme supports variety types of wavelets, and in this case, the research has adopted the CDF (9, 7) wavelet as an example. Table 2 illustrates equations for a single level forward LWT based on the CDF (9, 7) wavelet, and its scheduling software routine on a GPU is illustrated in Algorithm.1. The basic idea is that every step of the lifting scheme is performed by different functions, and the CPU program schedules and launches these functions with respect to all data dependencies.

In the context of CUDA and multi-GPU architectures, the overall workflow of LWT computation by using the devised DLB model conforms to Fig.2, and the scheduler allocates initial data groups of a raw data set to each GPU node, and then GPU nodes process the corresponding data groups with the LWT functions listed in Algorithm.1.

5 Test and Performance Evaluation

5.1 Hardware and test environment

This section analyses the tests and evaluation results of the developed data-oriented DLB model. Table 3 specifies the computer system constructed for the tests which contains two different type GPU nodes - a middle-low range GPU (NVIDIA GTX 750 Ti) and a high-end GPU (NVIDIA GTX 1080). The proposed model and LWT are realized by using CUDA C/C++ and CUDA Toolkit 8.

Table 3 The specifications of a computer system for the case study for close-up retrieval

	Description
CPU	Intel Core i7-4790 3.6GHZ
GPU1	GeForce GTX 750 Ti, 2G
GPU2	GeForce GTX 1080, 8G
OS	Windows 10 64 bit
CUDA	Version 8.0

5.2 FNN Training

The FNN can be trained end-to-end by the back propagation and the stochastic gradient descent (SGD) methods. Since there are limited open benchmarks or datasets for multi-GPUs based load balancing models, this study has devised a customizable dataset containing 5-state feedback parameters (see Table 1), the processing data size D and the corresponding actual processing time T. The relative computational ability CP can be given by equation (21):

$$CP = f(\frac{D}{T}) \tag{21}$$

We randomly initialized the weights for all layers (four layers) from a zero-mean Gaussian distribution algorithm, and trained the FNN in two steps to complete the supervised pre-training and fine-tuning. The first step trained the FNN on 300 data items with the SGD on a learning rate of 0.01. The fine-tuning step then continued the SGD on the learning rate of 0.001 with 200 data items. With this two-step training strategy, the FNN based DLB model has achieved a reliable prediction performance. The comprehensive evaluation of the devised DLB model is further discussed in the following subsections.

5.3 Computation without the DLB model

This section reports test results and evaluates the computational performance of multi-level 2D LWT without applying any DLB models on both a single GPU and a multi-GPU platform.

To begin with, this study tested and compared the processing time of a 2D LWT between a single GPU (using either GPU1 or GPU2 respectively) and two GPUs (using both GPU1 and GPU2) environment without employing any DLB models but the traditional division method to divide the data set for each GPU (see Section 2.3). This test performed 4 levels of forward 2D LWT with CDF (9, 7) wavelet on three data sizes 10240×10240, 11264×11264, to 12288×12288. The processing times of each test on three different data sizes had been recorded in Fig.7. It can be seen from Fig.7 that the GPU2 setting needs less processing time than the GPU1 setting, and the main reason for this is that the hardware performance of GPU2 is higher than GPU1 - the GPU2 has 2560 CUDA cores while GPU1 contains only 640 CUDA cores, and the memory storage of GPU2 is also larger than GPU1. According to Fig.7, the two GPUs (GPU1 & GPU2) setting merely gains limited speedup of about 1.6 times compared with GPU1, and at around 1.3 times compared with GPU2. The detail processing times for processing different data sizes by using the two GPUs setting are shown in Table 4 which also indicates the processing time of GPU1 and GPU2 respectively. It can be clearly seen from Table 4 that the overall processing time of the two GPUs setting in the context of the load unbalancing situation is equal to the GPU1-alone situation because the overall computational performance of a multi-GPU platform is ultimately determined by the GPU node with the lowest performance, in this case, the GPU1.

Table 4 The processing times of Two GPUs Setting with unbalancing implementations (ms) for close-up retrieval

data size	GPU1	GPU2	overall
1024010240	2758	1500	2758
1126411264	3876	2806	3876
1228812288	4500	3645	4500

5.4 Computation with the DLB model

Then, this study has tested and compared the computational performances of the 2D LWT operation between the unbalanced implementation (i.e., each GPU node processes a half of a large data set without consideration of their performance variations) and the data-oriented DLB implementation by using the FNN structure in the target multi-GPU system. The processing times of each implementation with different data sizes have been listed in Fig.8. The processing times of two single GPU settings (i.e. using GPU1 only and GPU2 respectively) also tested and recorded for comparison. It can be seen from Fig.8 that the computational performance of the unbalanced implementation has no significant difference comparing with the two single GPU settings. In contrast, the FNN based DLB implementation has gained improvement on computational performances steadily, i.e., it processed a very large data set (e.g. 16384×16384) in less than one second. Compared with the unbalanced implementation, the peak performance gain (speedup) can reach 12 times which is truly significant. The experimental results show that the proposed data-oriented DLB model can satisfy performance requirements from real-time and large-scale data intensive applications.s

5.5 Benchmarking

Lastly, this study carried out a benchmarking test. There are several data oriented DLB models on multi-GPUs, and the Boyer's model mentioned in section 2.3 is still considered as the most significant and mainstream strategy for data based DLB according to the latest review papers [22, 38], so the computational performance of the FNN based data-oriented DLB model has been compared with the representative DLB model by Boyer et al. [18]. In order to simulate the node performance fluctuations, tasks were assigned to the GPU nodes randomly. The experiment results are shown in Fig.9 where "stabilization" indicates the steady conditions and "fluctuation" represents the fluctuating conditions. It can be seen from Fig.9 that both the devised model and Boyer's model can keep the load (i.e., data allocations) in the balanced situations, and have consistent computational performance when the hardware performance of a multi-GPU platform can keep stable. However, once the hardware performance is perturbed, Boyer's model struggles to keep up the performance and the processing time increases dramatically (e.g. 12288×12288 in Fig.9). In contrast, the FNN based DLB model can resolve

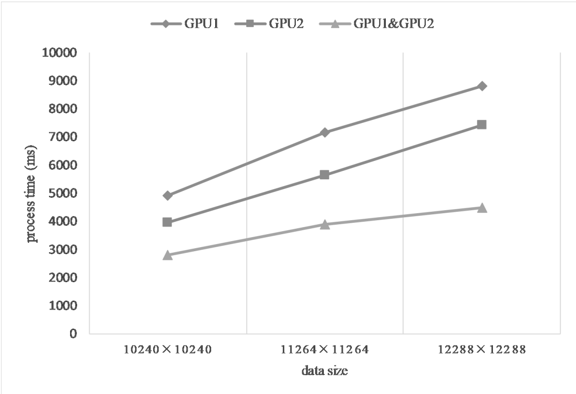


Fig. 7 The computational performance of three test settings (ms)

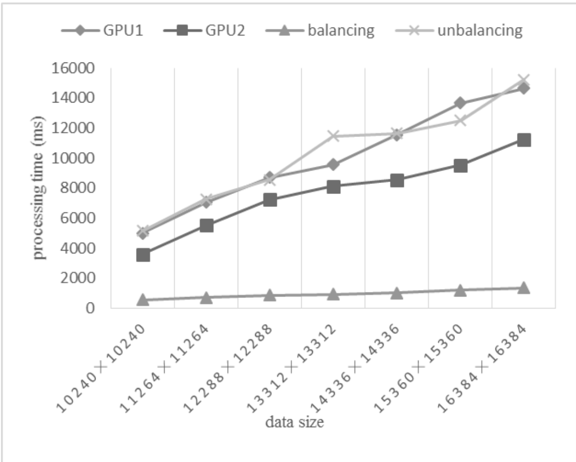


Fig. 8 The comparison of processing times between unbalancing and balancing implementations

the fluctuation problem readily and steadily due to its key feature of predicting and adjusting the current computational performance (P_i^n) dynamically according to the real hardware condition and feedbacks.

6 Conclusions and Future Work

To fully utilize the parallel computational power of modern GPUs, this paper presents a novel data-oriented DLB model for multi-GPU systems based on an innovative FNN structure and the corresponding dataset division methods. The research started with a comprehensive investigation and analysis of the traditional load balancing models, and concluded with the main drawbacks of them, for example, the rigidity when dealing with heterogeneous node specifications and configurations. To alleviate the load balancing issues and to effectively respond to the runtime fluctuation of cluster performance, this research has proposed a novel data-oriented DLB model for balancing and optimizing the overall parallel computational performance across multi-GPU nodes. In this model, five state feedback parameters have been identified, and the FNN structure has been implemented to predict the relative computational performance in an adaptive manner. An improved scheduler can then be activated to automating the data allocation tasks according to the relative computational performances across all nodes in a cluster. Experiment results show that the proposed model can achieve substantial computational performance gain when compared with conventional techniques, and the FNN based dynamic model can address the runtime fluctuation issues effectively. The innovative model and its corresponding techniques have addressed the key challenges from large scale computational applications that are often featured by extremely large input volume and highly repetitive operational procedures. Further work will be focused on bridging the flexible FNN idealism across the GPU and CPU boundary, especially when facing the new computing device paradigm of Cell CPUs, so as to progressing towards a truly hybrid and efficient task-data distribution scheme for engineering applications.

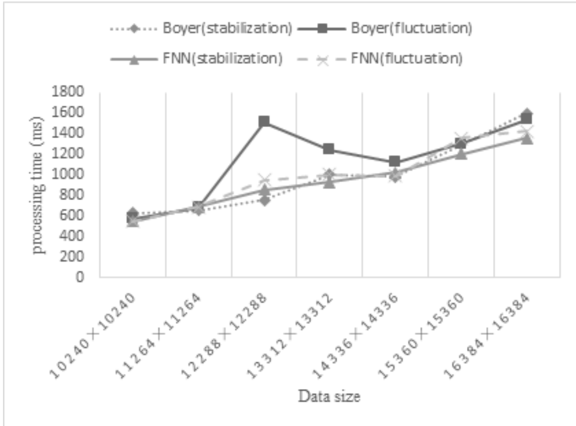


Fig. 9 The comparison of processing times between FNN based DLB model and the Beyer's model

References

[1] D. B. Kirk and W. H. Wen-mei, Programming massively parallel processors: a hands-on approach. Morgan kaufmann, 2016.

[2] R. Couturier, Designing Scientific Applications on GPUs. CRC Press, 2013.

[3] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, GPUs and the future of parallel computing. IEEE Micro, vol. 31, no. 5, pp. 7–17, 2011.

[4] C. W. Lee, J. Ko, and T.-Y. Choe, Two-way partitioning of a recursive Gaussian filter in CUDA. EURASIP Journal on Image and Video Processing, vol. 2014, no. 1, pp. 33–42, 2014.

[5] J. A. Belloch, A. Gonzalez, F. J. Martnez-Zaldvar, and A. M. Vidal, Real-time massive convolution for audio applications on GPU. The Journal of Supercomputing, vol. 58, no. 3, pp. 449–457, 2011.

[6] F. Nasse, C. Thurau, and G. A. Fink, Face Detection Using GPU-Based Convolutional Neural Networks. International Conference on Computer Analysis of Images and Patterns, pp. 83–90, 2009.

[7] NVIDIA, CUDA C Programming Guide v8.0. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.htm. 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, pp. 1–9, 2012.

[9] C. Szegedy et al., Going Deeper with Convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[10] K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. Information and Software Technology, vol. 51, no. 4, pp. 769–784, 2014.

[11] X. J. Jiang and D. J. Whitehouse, Technological shifts in surface metrology. CIRP Annals, vol. 61, no. 2, 2012.

[12] W. Jianjun, L. Wenlong, L. Xiaojun, and J. Xiangqian, High-speed parallel wavelet algorithm based on CUDA and its application in three-dimensional surface texture analysis. International Conference on Electric Information and Control Engineering, pp. 2249–2252, 2011.

[13] S. Chen and X. Li, A hybrid GPU/CPU FFT library for large FFT problems. Performance Computing and Communications Conference (IPCCC), 2013 IEEE 32nd International, pp. 1–10, 2013.

[14] C. Zhang, Y. Xu, J. He, J. Lu, L. Lu, and Z. Xu, Multi-GPUs Gaussian filtering for real-time big data processing. 2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA), 2016.

[15] S. Schaetz and M. Uecker, A multi-GPU programming library for real-time applications. International Conference on Algorithms and Architectures for Parallel Processing, pp. 231–236, 2012.

[16] J. A. Stuart and J. D. Owens, Multi-GPU MapReduce on GPU clusters. Parallel & Distributed Processing Symposium (IPDPS), pp. 1068–1079, 2011.

[17] M. Grossman, M. Breternitz, and V. Sarkar, HadoopCL: Mapreduce on distributed heterogeneous platforms through seamless integration of hadoop and opencl. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 1918–1927, 2013.

[18] M. Boyer, K. Skadron, S. Che, and N. Jayasena, Load balancing in a changing world. Proceedings of the ACM International Conference on Computing Frontiers, pp. 1–10, 2013.

[19] L. Chen, O. Villa, S. Krishnamoorthy, and G. R. Gao, Dynamic load balancing on single- and multi-GPU systems. 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1–12, 2010.

[20] A. Acosta, R. Corujo, V. Blanco, and F. Almeida, Dynamic load balancing on heterogeneous multicore/multiGPU systems. 2010 International Conference on High Performance Computing & Simulation, pp. 467–476, 2010.

[21] A. Acosta, V. Blanco, and F. Almeida, Towards the Dynamic Load Balancing on Heterogeneous Multi-GPU Systems. 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pp. 646–653, 2012.

[22] B. Prez, E. Stafford, J. L. Bosque, and R. Beivide, Energy efficiency of load balancing for data-parallel applications in heterogeneous systems. The Journal of Supercomputing, vol. 73, no. 1, pp. 330–342, 2017.

[23] R. Kaleem, R. Barik, T. Shpeisman, B. T. Lewis, C. Hu, and K. Pingali, Adaptive heterogeneous scheduling for integrated GPUs. Proceedings of the 23rd international conference on Parallel architectures and compilation, pp. 151–162, 2014.

[24] C. Zhang, Y. Xu, J. Zhou, Z. Xu, L. Lu, and J. Lu, Dynamic load balancing on multi-GPUs system for big data processing. 2017 23rd International Conference on Automation and Computing (ICAC), pp. 1–6, 2017.

[25] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition. arXiv Prepr, 2015.

[26] H. Zermane and H. Mouss, Development of an internet and fuzzy based control system of manufacturing process. International Journal of Automation and Computing, vol. 14, no. 6, pp. 706–718, 2017.

[27] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, Fuzzy Keyword Search over Encrypted Data in Cloud Computing. 2010 Proceedings IEEE INFOCOM, pp. 1–5, 2010.

[28] S. Krinidis and V. Chatzis, A Robust Fuzzy Local Information C-Means Clustering Algorithm. IEEE Transactions on Image Processing, vol. 19, no. 5, pp. 1328–1337, 2010.

[29] M. Algabri, H. Mathkour, and H. Ramdane, Mobile Robot Navigation and Obstacle-avoidance using ANFIS in Unknown Environment. International Journal of Computer Applications, vol. 91, no. 14, pp. 36–41, 2014.

[30] R. J. Kuo, S. Y. Hong, and Y. C. Huang, Integration of particle swarm optimization-based fuzzy neural network and artificial neural network for supplier selection. Applied Mathematical Modelling, vol. 34, no. 12, pp. 3976–3990, 2010.

[31] C. L. P. Chen, Yan-Jun Liu, and Guo-Xing Wen, Fuzzy Neural Network-Based Adaptive Control for a Class of Uncertain Nonlinear Stochastic Systems. IEEE Transactions on Cybernetics, vol. 44, no. 5, pp. 583–593, 2014.

[32] A. Saffar, R. Hooshmand, and A. Khodabakhshian, A new fuzzy optimal reconfiguration of distribution systems for loss reduction and load balancing using ant colony search-based algorithm. Applied Soft Computing, vol. 11, no. 5, pp. 4021–4028, 2011.

[33] N. Susila, S. Chandramathi, and R. Kishore, A Fuzzy-based Firefly Algorithm for Dynamic Load Balancing in Cloud Computing Environment. Journal of Emerging Technologies in Web Intelligence, vol. 6, no. 4, pp. 435–440, 2014.

[34] A. N. Toosi and R. Buyya, A Fuzzy Logic-Based Controller for Cost and Energy Efficient Load Balancing in Geo-distributed Data Centers. 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 186–194, 2015.

[35] H. Muhamedsalih, X. Jiang, and F. Gao, Accelerated surface measurement using wavelength scanning interferometer with compensation of environmental noise. Procedia Engineering: 12th CIRP Conference on Computer Aided Tolerancing, vol. 10, pp. 70–76, 2013.

[36] S.-H. Lee and J. S. Lim, Forecasting KOSPI based on a neural network with weighted fuzzy membership functions. Expert Systems with Applications, vol. 38, no. 4, pp. 4259–4263, 2011.

[37] W. Sweldens, The lifting scheme: A construction of second generation wavelets. SIAM Journal on Mathematical Analysis, vol. 29, no. 2, pp. 511–546, 1998.

[38] S. Mittal and J. S. Vetter, A Survey of CPU-GPU Heterogeneous Computing Techniques. ACM Computing Surveys, vol. 47, no. 4, pp. 1–35, 2015.

Chaolong Zhang received the BEng and MSc degrees in software engineering from Chengdu University of Information Technology, China, in 2014 and 2017, respectively. He is currently a PhD student with School of Computing and Engineering at University of Huddersfield, UK.

His research interests include High-performance computing (HPC), Computer Vision, and deep learning network applications.

E-mail: chaolong.zhang@hud.ac.uk
ORCID iD: 0000-0003-4990-4636

Yuanping Xu received the BEng degree in computer science and technology from Southwest JiaoTong University, China, in 2003, and MSc and PhD degrees in software engineering from University of Huddersfield, UK in 2004 and 2009, respectively. From February 2009 to November 2010, he worked as a research fellow in the Centre of Precision Technologies at University of Huddersfield, UK. He is currently a professor with School of Software Engineering at Chengdu University of Information Technology, China.

His research interests include knowledge-based systems, expert systems, big data analysis and deep learning network applications.

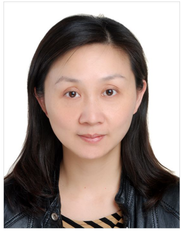
E-mail: ypxu@cuit.edu.cn (Corresponding author)

ORCID iD: 0000-0002-4536-6220

Zhijie Xu received the B.Eng degree in communication engineering from the Xi'an University of Science and Technology at China in 1991. After graduation, he first started as an electronics engineer before moving to the United Kingdom and worked as a research scientist in the Robotics Lab at the University of Derby. He received the Ph.D degree in 2000 from the University of Derby based on his research work in Virtual Reality-based Manufacturing Simulation and Robotics Systems. He has been employed as a full time academic member of staff since April 1999 serving the roles of Lecturer, Senior Lecturer, Reader and Professor respectively at the University of Huddersfield in UK.

Professor Xu's research in the last twenty years has mainly been focused on the areas of visualization, HCI, vision systems, and machine learning. He has published over one hundred peer-reviewed journal and conference papers as well as edited 5 books in the relevant fields. He has successfully supervised 8 PhD students to completion while securing substantial research and industrial grants. He is a member of the IEEE, IET, BCS, and a fellow of HEA, and editors for multiple prestigious academic journals and conferences. He is the current President of the Chinese Automation and Computing Society in the United Kingdom.

E-mail: z.xu@hud.ac.uk



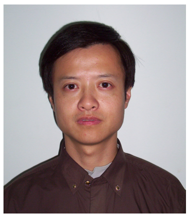
Jia He received BEng and Msc degrees in Computer Science and Technology from Southwest Normal University of China, in 1989 and 1996, respectively, and received PhD degree in Computer Science from University of Electronic Science and Technology of China, in 2012. She is currently a professor and the Dean with School of Computer Science at Chengdu University of Information Technology, China.

Her current research interests include computer vision, artificial intelligence, and pattern recognition.

E-mail: hejia@cuit.edu.cn

Jing Wang gained his PhD degree in 2012 from University of Huddersfield. He worked as Research Fellow and carried out independent research work on Image Processing, Analysing and Understanding in University of Huddersfield before 2017. He is now working at Sheffield Hallam University as a Lecturer in software engineering and computer science. His research interests are focused on real-world applications of Computer Vision Systems and have published more than 20 refereed papers in the related fields.

E-mail: jing.wang@shu.ac.uk



Jianhua Adu received BSc degree in Applied Physics from Minzu University of China, Beijing, China, in 1999, received MSc in Computer Science from ShanDong University, China, in 2006, and received PhD degree in Computer Science from SiChuan University, China, in 2012. He is currently an associate professor with School of Software Engineering at Chengdu University of Information Technology, China.

His current research interests include image fusion and segmentation, medical image processing and analysis, and pattern recognition.

E-mail: adujh@126.com