Article

RGCA: a Reliable GPU Cluster Architecture for Large-Scale Internet of Things Computing Based on Effective Performance-Energy Optimization

Yuling Fang, Qingkui Chen *, Neal N. Xiong, Deyu Zhao and Jingjuan Wang

University of Shanghai for Science and Technology, Shanghai 200093, China; forwardfyl@163.com (Y.L.F.); xiongnaixue@gmail.com (N.N.X); zhao781201@163.com (D.Y.Z); wjj9209@163.com (J.J.W)

* Correspondence: chenqingkui@usst.edu.cn; Tel.: +86-131-2238-1881

Abstract: This paper aims to develop a low-cost, high-performance and high-reliability computing system to process large-scale data using common data mining algorithms in the Internet of Things computing. Considering the characteristics of IoT data processing, similar to mainstream high performance computing, we use a GPU cluster to achieve better IoT services. Firstly, we present an energy consumption calculation method (ECCM) based on WSN. Then, using the CUDA Programming model, we propose a Two-level Parallel Optimization Model (TLPOM) which exploits reasonable resource planning and common compiler optimization techniques to obtain the best blocks and threads configuration considering the resource constraints of each node. The key to this part is dynamic coupling Thread-Level Parallelism (TLP) and Instruction-Level Parallelism (ILP) to improve the performance of the algorithms without additional energy consumption. Finally, combining the ECCM and the TLPOM, we use the Reliable GPU Cluster Architecture (RGCA) to obtain a high-reliability computing system considering the nodes' diversity, algorithm characteristics, etc. The results show that the performance of the algorithms significantly increased by 34.1%, 33.96% and 24.07% for Fermi, Kepler and Maxwell on average with TLPOM and the RGCA ensures that our IoT computing system provides low-cost and high-reliability services.

Keywords: Internet of Things; data mining algorithms; GPU cluster; performance; energy consumption; reliability

1. Introduction

Since the first day it was conceived, the Internet of Things (IoT) [1, 2] has been considered the technology for seamlessly integrating classical networks and networked objects [3]. The basic idea of IoT is to connect the Internet to all things used by us in the world [4]. Nowadays, researchers from different industries, research groups, academics, and government departments focus on revolutionizing the Internet [5] by constructing a more convenient environment that is composed of various intelligent systems, such as smart homes, intelligent transportation, global supply chain logistics, and health care [6]. The application of domestic and international IoT mainly includes smart homes, intelligent environmental protection, intelligent transportation, intelligent security and industrial monitoring, etc. In this paper, we focus on the application of IoT for smart homes.

With the development of modern technology, people are no longer satisfied with the fundamental function of their houses, and seek more social functions, such as home offices, online education and information of all the places [7]. Smart homes are built through the use of wireless sensors, image recognition, RFID, positioning and other technical means to comprehensively perceive the family environment and changes to people and things [8]. The establishment of monitoring and early warning systems in the family home through Wireless Sensor Networks (WSNs) based on the IoTs build up informatization, intellectualization remote control network. So that users of the system can clearly and in a timely fashion understand the current state of the environment [9]. However, it is necessary to establish an efficient and high-reliability model to guarantee that the system can continuously execute without malfunction. In the work in [10-12], an energy conserving mechanism and a high-energy efficient scheme for the IoTs were studied.

Our large-scale smart home cloud platform lab can access and store 48M data per second in real-time from 420,000 households, enabling millions of bytes of data throughout per second. In IoT, the massive amount of data generated by a user's family is collected by the sensor network deployed in each home, and the collected data packets are decoded and then stored into the Data Access System for the Internet of Things (DASIoT) [13]. After DASIoT's transformation and control, this information is exchanged and communicated for the intelligent identification, location, tracking, supervision and management of the objects [14] and processes of the application system so that an intelligent network is created between man, the Internet and things. In terms of the framework of the IoT application system, DASIoT mainly works at the application level of the upper level of the IoT, as shown in Figure 1.

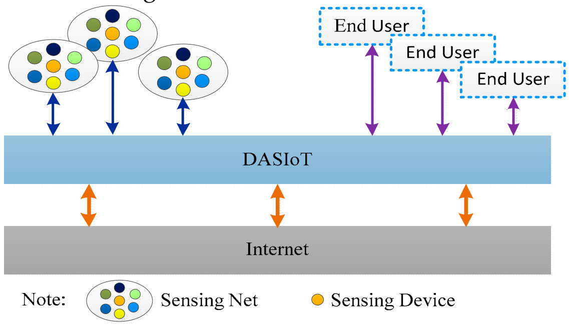


Figure 1. Structure of the integrated IoT system.

In Figure 1, the main function of the DASIoT is to provide network access services for end users (including PCs, mobiles, etc.) and coordinate the large amount of requests from end users by transmitting them onto the sensing devices. For data processing, one of the most important questions that arise now is, how do we convert the data generated or captured by DASIoT into knowledge to provide a more convenient environment for people? This is where useful information discovery in database and data mining technologies comes into play, for these technologies provide possible solutions to find information hidden in the data of IoT which can be used to enhance the performance of the system or to improve the quality of services [15]. Previous researchers focused on using or developing effective data mining technologies for the DASIoT, and their results in [16, 17] prove that some data mining algorithms are able to make IoT more useful and intelligent, thus providing faster and smarter services.

In this paper, we not only give a related description of common data mining algorithms in IoT computing but also propose a detailed performance-energy optimization scheme based on a GPU cluster in order to acquire better services. In this study, we answer the following important questions: Is the CUDA programming model suitable for common data mining algorithms in IoT computing? Can the GPU cluster help IoT achieve a low-overhead, high real-time and high-reliability system? In addition to adjusting Thread-Level Parallelism (TLP), what can be done to improve the performance of parallel programs? To achieve these goals, we propose a new optimization architecture for IoT computing which is applicable, accurate and straightforward for different data mining algorithms. It is worth mentioning that, unlike previous performance-energy models, our model applies to different types of applications, including compute-bound, memory-bound and balanced type kernels. By exploiting the appropriate TLP and instruction-level parallelism (ILP) configuration, the Two-level Parallel Optimization Model (TLPOM) improves its average performance by 34.1%, 33.96% and 24.07% for Fermi, Kepler and Maxwell architecture, respectively. Then, on the basis of considering the type of kernels, we use different computing nodes in our cluster. Finally, we obtain a low-cost and high-reliability computing system.

The rest of this paper is organized as follows: in Section 2, we overview the related work. Section 3 presents the energy consumption calculation model and our current sampling method. The TLPOM is presented in Section 4 and Section 5 gives our RGCA for the high-reliability computing system. Our experiment results and analysis are shown in Section 6. Finally, we conclude this paper in Section 7.

2. Related work

With the development of IoT, an increasing amount of large-scale data needs to be processed in the DASIoT. However, technical issues and challenges on how to handle these data and how to extract useful information have emerged in recent years. The issue we need to take into account is that the data from IoT is generally too big and it is too difficult to process using the tools available today [18, 19]. As paper [20] proposed, the bottleneck of IoT services will shift from sensor to data processing, communication, etc. Since the issue of processing massive data has been studied for years, it is not surprising that some traditional but practical methods have been applied in IoT, such as random sampling [21], data condensation [22], and incremental learning [23]. However, these methods only handle interesting data instead of all the data, so all of these studies need a data preprocess in DASIoT, as shown in Figure 2. In addition to preprocessing, how to find information hidden in these data has become a crucial issue for better IoT services. According to paper [15], the data mining technique is responsible for extracting information from the output of the data processing step and then feeding this into the decision-making step, which transforms its input into useful knowledge. Therefore, in relation to big data, the application of data mining techniques has become increasingly extensive [16, 24].

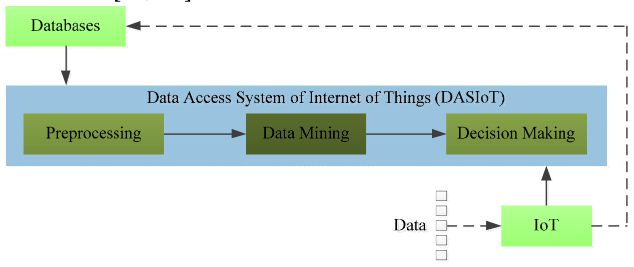


Figure 2. Basic structure of the DASIoT.

2.1. Introduction to Common Data Mining Algorithms

It is much easier to create data than to analyze data. To develop a high-performance and high-reliability data mining module for IoT computing, there are several key considerations when choosing applicable mining technologies: the objective, input data, output data, the characteristics of data, application scenario and mining algorithm. According to papers [15, 16], the authors divide the data mining algorithms into many categories based on their application, including classification, clustering, association analysis, time series, outlier analysis, etc. Here we briefly introduce several common data mining algorithms and their features:

- K-means: As one of the most well-known and simple clustering algorithms, K-means [25] is widely used in multiple domains, such as data mining and pattern recognition. It generally analyzes data objects without consulting a known class model. Firstly, a user-specified number of clusters, k, is created randomly to represent how the input patterns are divided into different groups. Then, the assignment operator computes the minimization of intra-cluster distance and the maximization of the inter-cluster distance, and this idea complies with parallel reduction. So a high-efficiency parallel k-means algorithm should be considered for better performance.
- KNN: This is one of the most traditional and simplest algorithms in data mining, k-nearest neighbor (KNN) [26], which was proposed by Michael Steinbach. It memorizes the entire training data and performs classification only if the attributes of the test object match one of the training examples exactly. KNN provides a summary of the nearest-neighbor classification method. The algorithm computes the distance between test data and all the training objects to

determine its nearest-neighbor list. Once the nearest-neighbor list is obtained, the test object is classified based on the majority class of its nearest neighbors.

- BPNN: The back-propagation neural network (BPNN) is a multilayer negative feedforward neutral network trained by the error back propagation algorithm, which is the most widely used neutral network. It has the ability of non-linear mapping and parallel distributed processing, as well as being self-learning and adaptive. It also allows an arbitrary number of inputs and outputs. BPNN is also one of the common classification algorithms in IoT computing [27]. In addition to classification, it can also be used to predict conditions in the home environment, such as temperature and humidity. According to its algorithm structure, we know that BPNN consists plenty of operations which are suitable for matrix multiplication, dot product, etc., and this provides the basis for the parallel optimization of BPNN.
- SVM: Support Vector Machine (SVM) is a kind of supervision learning method. It is widely used in data applications, for example data mining [15, 26] in IoT. The basic idea of SVM is to map the vector into high-dimensional space, and then to establish the vector spacing of the largest hyperplane to separate these data. SVM involves three phases: learning phase, cross-validation phase and classification prediction phase. While in the learning phase, it can consume a large amount of computational resources. Therefore, how to improve the efficiency of processing massive data is the major issue.
- GMM: Gaussian mixture model (GMM) is also a clustering algorithm in IoT. Unlike K-means, GMM gives the probability that data points are assigned to each cluster [28], also known as soft assignment. In some applications, the probability has a great effect, for example disease diagnosis. Just as its name implies, it assumes that the data follows the mixture Gaussian distribution. In other words, the data can be thought of being generated from several Gaussian distributions. In addition, by increasing the number of models, we can arbitrarily approximate any continuous probability dense distribution, and each GMM consists of K Gaussian distributions.

Table 1. The benchmarks.

Algorithm	Application	Type	benchmark
K-means	Clustering	M	Rodinia
QT Clustering	Clustering	B	SHOC
KNN	Classification	B	Rodinia
BPNN	Classification	C	Rodinia
FFT	Basic Function	C	SHOC
Matrixmul	Basic Function	M	CUDA SDK
Reduction	Basic Function	M	CUDA SDK
Gaussian	Basic Function	B	SHOC

By combining the above algorithms and their basic functions, we extract the benchmarks to optimize our parallel optimization method, as shown in Table 1. All these algorithms have been paralleled in previous research [29], and we also modified the parallelism. Reduction is a class of parallel algorithms which produces one result using a binary operator $\oplus$ that conforms to the associative law. Such operations include minimization, maximization, sum, squared sum, logic and/or vector dot product, etc. This is an important basic step in other advanced algorithms, such as k-means, KNN and GMM. Matrixmul is used in many data mining algorithms, such as SVM, spectral clustering and PCA. The Gaussian kernel function is the most commonly used radial basic function (RBF) which has several important properties, such as rotational symmetry, separability, the single spectral Fourier transform, etc. These features indicate that the Gaussian smooth filter is a very effective low-pass filter in both the frequency domain and time domain. It is widely used for RBF, BPNN and GMM. The FFT is also an important basic algorithm in many fields. For example, it is used for signal processing [30] in data mining.

2.2. CUDA Programming Model and GPU Architecture

2.2.1. CUDA Programming Model

CUDA (Compute Unified Device Architecture) is a general parallel computing architecture released by NVIDIA. It enables applications to take full advantage of the respective merits of CPU and GPU, and solve complex computing problems. In CUDA, the CPU is regarded as the host and GPU as the coprocessor or device, and they work cooperatively to perform their functions. The CPU is responsible for logical transaction processing and serial computing, and the GPU focuses on highly linearized parallel task processing [31]. They each have their own independent memory address space: host memory and device memory.

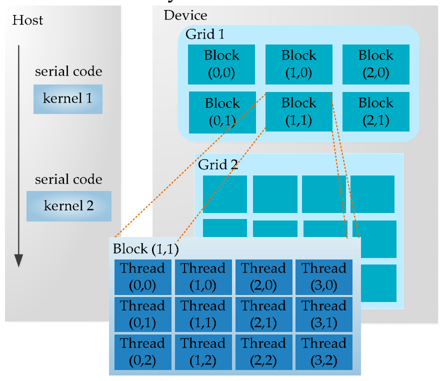


Figure 3. CUDA programming model.

Once the parallel part of the program is determined, this part is handed over to the GPU. The parallel computing function running on the GPU is called the kernel. A kernel is not a complete program, but a step that can be executed in parallel in the entire CUDA program. As shown in Figure 3, a complete CUDA program is composed of a series of device-end parallel kernels and the host-end serial processing steps. There are two-level parallel hierarchy in a kernel, and one is the blocks in the grid and another is the threads in the block [31]. The two-tier parallel model is one of the most important innovations of CUDA, which is also the original source of inspiration for our performance optimization model (in Section 4).

2.2.2. GPU Architecture

NVIDIA and AMD are the principal suppliers of civilian and military graphics cards. In this paper, we use three kinds of GPUs: Fermi, Kepler and Maxwell. For simplicity, we mainly focus on Kepler GTX 680 (called 680 hereon in) GPU in this section. Large-scale parallelism on GPU hardware is achieved by repeatedly setting up multiple identical general building blocks (SMs). Figure 4(a) shows an overview of the 680 block diagram. GPU receives CPU commands via the Host Interface. The GigaThread Engine [32] creates and dispatches blocks to SM thread schedulers, and manages the context switches between threads during execution. An individual SM in turn schedules, warps and maps the threads to the CUDA cores and other execution units (LD/ST, SFU). The basic unit of execution flow in the SM is the warp, which contains 32 parallel threads and executes in a single instruction multiple thread (SIMT) [31] model. Using GPU, one of the key aspects for applications is breaking down the kernels into appropriate-sized grids and blocksize, where the grid indicates the dimension of the thread block and the blocksize indicates the number of threads per thread block. A bad choice of thread layout typically also leads to a bad memory pattern, which will significantly harm performance [33]. GPUs always exploit high TLP to hide the processing latency of individual warps [34].

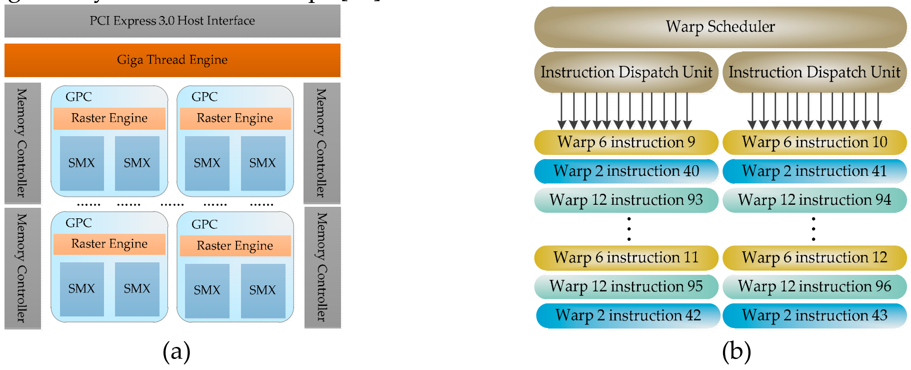


Figure 4. GeForce GTX 680 main architecture: (a) Block diagram; (b) A single warp scheduler.

From Kepler, each SM features four warp schedulers and eight instruction dispatch units, as shown in Figure 4(b), allowing four warps to be issued and executed concurrently. Having less than four warps means that one or more of the instruction dispatch units will remain idle, effectively damaging the instruction dispatch speed. Another important cause of performance constraint is shared resource usage. Resource contention limits the ability of shared resources (including CUDA cores, load/store unit, registers, shared memory, etc.) to continuously perform the same operations. According to the launch configuration [33], we try to optimize it in the following aspects: the number of threads per block, the number of blocks and the tasks performed per thread (ILP). Therefore, in this paper, we combine TLP and ILP to improve performance and reduce energy consumption.

3. Power Consumption Calculation Model

3.1. Traditional Power Consumption Model

According to the CPU power consumption calculation definition [35], GPU power consumption consists of three parts: dynamic power consumption, short-circuit power consumption and power loss due to transistor leakage currents:

$$P_{GPU} = P_{dynamic} + P_{static} + P_{leak} \tag{1}$$

In Equation (1), dynamic power consumption originates from the activity of logic gates inside a GPU, and static power is mainly determined by circuit technology, chip layout and operating temperature. Both dynamic and short-circuit power consumption are dependent on the clock frequency, while the leakage current is dependent on the GPU supply voltage.

3.2. Our Power Consumption Model

The power consumption model proposed in this paper is different from using a power analyzer to directly measure the power consumption of GPU, as we calculate the energy consumption of a computing node according to the work in [36]. The energy consumption E_{cn_i} of the ith computing node cn_i at the GPU cluster can be calculated by power consumption $P_{cn_i}(I_i(t))$ during time interval $[t_0, t_{n-1}]$, as shown in Equation (2).

$$E_{cn_i} = \int_{t_0}^{t_{n-1}} P_{cn_i}(I_i(t))d_t \tag{2}$$

where power consumption $P_{cn_i}(I_i(t))$ can be represented by Equation (3).

$$P_{cn_i}(I_i(t)) = P_{cn_i}^{idl}(I_i(t)) + P_{cn_i}^{dny}(I_i(t)) \tag{3}$$

where $P_{cn_i}^{idl}(I_i(t))$ is the power consumption when the ith computing node is in the idle state, $P_{cn_i}^{dny}(I_i(t))$ is the power consumption spreading from idle to full utilized, and $I_i(t)$ is the current on the ith computing node, which changes over time. We present the measurement method in Section 3.3.

According to [37] and Section 3.1, power consumption $P_{cn_i}^{idl}(I_i(t))$ is mostly determined by the GPU state, the number of active CUDA cores and memory type etc. The power consumption of the application of DVFS [38] on the GPU is almost a linear power-to-frequency relationship for a computing node. So,

$$\begin{cases} P_{cn_i}^{idl}(I_i(t)) = U_{cn_i} \cdot I_{idl} \\ P_{cn_i}^{dny}(I_i(t)) = U_{cn_i} \cdot i_{cn_i}^{dny}(t) \end{cases} \tag{4}$$

where I_{idl} is the static current of per node, it is a constant, while $i_{cn_i}^{dny}(t)$ is the working current per node and is determined by the applications and the specific node.

The energy consumption E_{cn_i} of the node can be further described by Theorem 1.

Theorem 1. If the time interval $[t_0, t_{n-1}]$ can be further divided into $[t_0, t_1, \ldots, t_{n-1}], \forall [t_{k-1}, t_k], k \in [1, 2, \ldots, n]$, and $i_{cn_i}^{dny}(t \in [t_{k-1}, t_k]) \equiv i_{ik}$, where i_{ik} is a constant, and does not change over time, then energy consumption E_{cn_i} can be calculated by (5),

$$E_{cn_i} = U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + U_{cn_i} \cdot \sum_{k=1}^{n-1} \left(i_{i_{k-1}} \cdot (t_k - t_{k-1}) \right) \tag{5}$$

Proof. The power consumption $P_{cn_i}(I_i(t))$ can be divided into two parts, $P_{cn_i}^{idl}(I_i(t))$ and $P_{cn_i}^{dny}(I_i(t))$. Therefore,

$$\begin{aligned} E_{cn_i} &= \int_{t_0}^{t_{n-1}} P_{cn_i}(I_i(t)) d_t = \int_{t_0}^{t_{n-1}} \left(P_{cn_i}^{idl}(I_i(t)) + P_{cn_i}^{dny}(I_i(t)) \right) d_t \\ &= \int_{t_0}^{t_{n-1}} P_{cn_i}^{idl}(I_i(t)) d_t + \int_{t_0}^{t_{n-1}} P_{cn_i}^{dny}(I_i(t)) d_t \\ &= \int_{t_0}^{t_{n-1}} U_{cn_i} \cdot I_{idl} d_t + \int_{t_0}^{t_{n-1}} U_{cn_i} \cdot i_{cn_i}^{dny}(t) d_t \\ &= U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + \int_{t_0}^{t_{n-1}} U_{cn_i} \cdot i_{cn_i}^{dny}(t) d_t \end{aligned} \tag{6}$$

When the time interval $[t_0, t_{n-1}]$ is divided into *n-1* constant time interval $[t_0, t_1, \ldots, t_{n-1}], \forall [t_{k-1}, t_k], k \in [1, 2, \ldots, n]$, $i_{cn_i}^{dny}(t \in [t_{k-1}, t_k]) \equiv i_{ik}$, and i_{ik} is a constant, then we obtain,

$$\begin{aligned} E_{cn_i} &= U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + \int_{t_0}^{t_{n-1}} U_{cn_i} \cdot i_{cn_i}^{dny}(t) d_t \\ &= U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + \int_{t_0}^{t_1} \left(U_{cn_i} \cdot i_{i_0} \right) d_t + \int_{t_1}^{t_2} \left(U_{cn_i} \cdot i_{i_1} \right) d_t + \cdots + \int_{t_{n-2}}^{t_{n-1}} \left(U_{cn_i} \cdot i_{i_{n-2}} \right) d_t \\ &= U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + U_{cn_i} \cdot i_{i_0} \cdot (t_1 - t_0) + U_{cn_i} \cdot i_{i_0} \cdot (t_2 - t_1) + \cdots + U_{cn_i} \cdot i_{i_0} \cdot (t_{n-1} - t_{n-2}) \\ &= U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + \sum_{k=1}^{n-1} U_{cn_i} \cdot i_{i_{k-1}} \cdot (t_k - t_{k-1}) \\ &= U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + U_{cn_i} \sum_{k=1}^{n-1} i_{i_{k-1}} \cdot (t_k - t_{k-1}) \end{aligned} \tag{7}$$

The proof ends.

Lemma 1 (Cluster energy consumption). Cluster energy consumption E_{clu} is the quantitative denotation of the energy consumption of the GPU cluster computing environment. If the GPU cluster has *n* computer nodes, the cluster energy consumption E_{clu} during time interval $[t_0, t_{n-1}]$ can be quantified by Equation (8):

$$E_{clu} = \sum_{i=0}^{n-1} E_{cn_i} \tag{8}$$

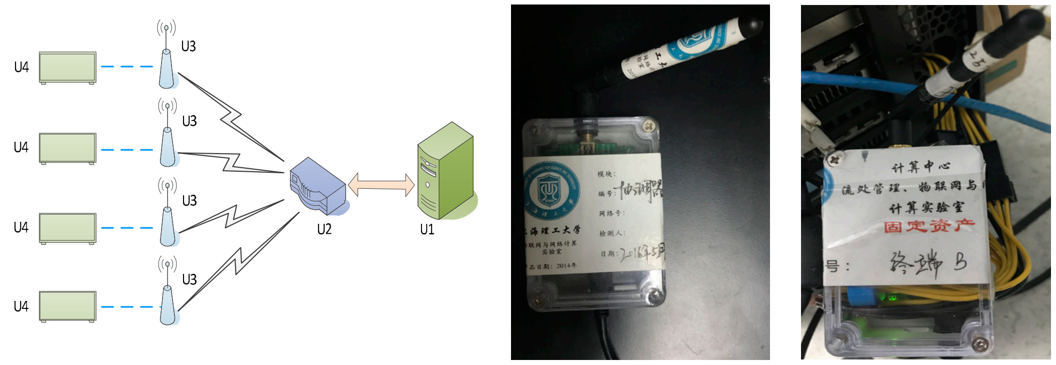
Hence, we have,

$$E_{clu} = \sum_{i=0}^{n-1} \left(U_{cn_i} \cdot I_{idl} \cdot (t_{n-1} - t_0) + U_{cn_i} \sum_{k=0}^{n-1} i_{i_{k-1}} \cdot (t_k - t_{k-1}) \right) \tag{9}$$

In our computing system, the voltage is a constant 12V, and the current is measured by the homemade power consumption monitoring system of the GPU cluster.

3.3. Power Consumption Monitoring System Based on WSN

In Section 3.2, we discussed that the current is obtained from a homemade wireless sensor network, the power consumption monitoring system of the GPU cluster [39]. As seen from Figure 5, it is composed of multiple components: a master monitor terminal U1, a Zigbee coordinator U2, multiple sensor nodes U3 and a GPU cluster (including multiple GPU computing nodes) U4. The system also contains multiple slave monitor terminals, but these are not shown in Figure 4. U3 contains a node controller and a Hall current sensor connected to the node controller and a Zigbee communication module. For each sensor node, the Zigbee coordinator U2 is inter-connected with the master monitor terminal through the communication line, and the Zigbee communication module is inter-connected with the Zigbee coordinator using the Zigbee communication mode. The node controller is inter-connected with the slave monitor terminal through the communication line, and the Hall current sensor is set on the power supply line of each GPU, respectively.



(a) (b) (c)

Figure 5. GPU energy consumption monitoring system: (a) The schematic diagram of the system; (b) The Zigbee coordinator; (c) A sensor node.

In our sensor network, the Zigbee coordinators use the CC2530 chip to form the network and transfer packets and instructions. The sensor node is responsible for collecting the current data of the computing node in a cluster, and transfers the collected data through Zigbee to the main monitoring node, and the main monitoring node stores and displays the data in real time. The sensor node can also transfer the collected data to the sub-monitoring node, and the sub-monitoring node stores and displays the data in real time. The CC2530 chip has different operating modes, which is suitable for a system that requires low power consumption, and the time of switching between different modes is short, which further ensures low energy consumption. The model of the Hall current sensor in the sensor node is WHB-LSP5S2H. The Hall current sensor mainly measures the current of the GPU power supply line, and transmits the measured current value to the node controller. Then, the node controller performs the analog-to-digital conversion.

Our sensor nodes have two kinds of working modes: wireless and wired. In the wireless mode, its maximum sampling frequency is 50Hz, while in the wired mode, its maximum sampling frequency is 100Hz. Having a sufficient number of sampling points in these two modes ensures the accuracy of the calculation result and provides an accurate basis for the evaluation of the experimental results.

4. Two-Level Parallelism Optimization Model

Generally, an algorithm's performance is limited by multiple factors, such as memory/ instruction throughput, latency and so on. In previous studies, many researchers proposed various ways to improve the performance of the parallel algorithm. The work in [40] mainly studies the effect of warp sizing and scheduling on performance, and the work in [41] also analyzes the impact of warp-level sizing and thread block-level resource management. Both these studies adjust the number of active warps to improve performance. The work in [42] analyzes the usage of computing resources and memory resources for different applications, and it simulates the maximum TLP and exploits the underutilized resources as much as possible. ILP is presented in the work in [43], which proposes to build SCs for GPU-like many-core processors to achieve both high performance and high energy efficiency. The latter two methods are verified in the simulation environment. Here we combine TLP and ILP to address the performance and energy consumption problem, and we demonstrate our method in a real environment.

4.1. Distinguish Kernel Type

Though the NVIDIA Visual Profiler is able to analyze the limiters of the performance of parallel programs, it does not state the specify method by which to distinguish the type of kernel. Therefore, we first should determine the type of kernel according to the use of different resources by the program. We analyze the type of kernels based on two metrics: the ratio of compute utilization to hardware (HW) peak and the ratio of device memory utilization relative to HW peak. If one of the metrics is close ("close" is approximate, that is to say, 65% of theory or better) to the HW peak, the performance is likely limited by it [44]. If neither metric is close to the peak, then unhidden latency is likely an issue. We present a simple method by which to judge the type of kernel. Firstly, it is important to be aware of the related information about computing nodes at the GPU cluster, as shown in Table 2.

Table 2. Resource distribution of different GPUs.

GPU	Architecture	Register files	Shared Mem	Memory BW	Inst/clk	CUDA cores
GTX 280	Fermi	16K*32-bit	16K	142 GB/s	1024	240
GTX 680	Kepler	64K*32-bit	48K	192 GB/s	2048	1536
GTX 970	Maxwell	64K*32-bit	48K	224 GB/s	2048	1664

We can obtain the peak of memory bandwidth and maximum compute resource of different GPUs from Table 2. Then, we define $ratio_{mem}$, $ratio_{com}$, and they represent memory utilization and compute resource utilization of the GPU, respectively.

$$ratio_{mem} = \frac{global\,memory\,utilization}{memory\,throughout} \tag{10}$$

$$ratio_{com} = \frac{global\,compute\,utilization}{compute\,throughout} \tag{11}$$

$$R = \frac{ratio_{com}}{ratio_{mem}} \tag{12}$$

In terms of the above rules, if either of the ratios ($ratio_{mem}$ and $ratio_{com}$) is more than 65% firstly, the kernel is likely limited by the metric. To better characterize kernel behavior, we break the kernels down into the five types, as shown in Figure 6, based on their resource utilization ratios: compute-bound, weak compute-bound, balanced, weak memory-bound and memory-bound.

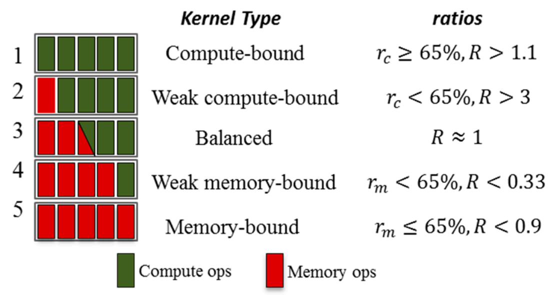


Figure 6. Kernel type categorization based on the resource utilization ratios. The critical values are empirically chosen.

The classification results not only help improve performance (as discussed in Section 4), but also provide guidance to assist system reliability (in Section 5.2). The classification results are presented in Table 1. In the next sections, for convenience, we classify weak compute-bound type and compute-bound type as compute-bound (CB) type, weak memory-bound type and memory-type as memory-bound (MB) type.

4.2. Coarse-Grained Thread-Level Parallelism

4.2.1. The Impact of Higher TLP

GPUs exploit tens of thousands of concurrent threads to hide processing latency for arithmetic calculation and memory access. If a warp is stalled by a data dependency or long latency memory access, then warp schedulers issue another ready warp from the warp pool so that the execution of warps is interleaved [42]. The availability of stall hiding relies on the number of eligible warps in the warp pool, which is the primary reason why GPUs require a large number of concurrent threads [45]. Here, we use TLP to quantify the proportion of active warps in a SM. Higher TLP does not always equate to higher performance, however, low TLP always lacks the competence to hide memory latency, resulting in performance degradation. In order to allocate the most appropriate TLP, we should adhere to the following rules.

4.2.2. Appropriate Block Configuration

For the Kepler GTX 680, similar to most other GPUs, most of the key hardware units for graphics processing reside in the SM. It has 8 SMs and 192 CUDA cores per SM, the specific parameters of which are shown in Table 3.

Table 3. Basic parameters according to NVIDIA.

Variable	Fermi-280	Kepler-680	Maxwell-970
SM	15	8	13
CUDACoreInSM	32	192	128
MaxBlockInSM	8	16	32
MaxwarpInSM	48	64	64
MaxthreadInBlock	1024	1024	1024
MaxThreadInSM	1536	2048	2048
MaxRegisterInThread	32	64	64

The number of threads per block should be a multiple of 32, the size of the warp, because this provides optimal computing efficiency and facilitates coalescing. The dimension of blocks per grid and the dimension of threads per block do not play a role in performance [31]. As a result, this section only discusses size but not dimension. When choosing the first configuration parameter, the number of blocks per grid, our goal is to ensure all the CUDA SMs are busy. So, the number of blocks in the grid should be larger than the number of SMs. Furthermore, there should be multiple active blocks per SM so that blocks do not have to wait for a __syncthreads() can keep the hardware busy. Another important parameter also needs to be considered, this being the number of threads per block. It is in some cases possible to fully cover latency with even fewer warps, notably via ILP (in Section 4.3). For TLP, we have,

$$N_{warp\_block} = \frac{N_{thread\_block}}{sizeof(warp)} \tag{13}$$

$$N_{warp\_SM} = N_{warp\_block} \times N_{block\_SM} \tag{14}$$

$$P_{degree} = \sum_{i=1}^{n} N_{block\_SM} \times N_{thread\_block} \tag{15}$$

$$N_{block\_SM} \le \left\lceil \frac{MaxThreadInSM}{N_{thread\_block}} \right\rceil \tag{16}$$

$$MinThreadInSM \le N_{thread\_block} \le MaxThreadInSM \tag{17}$$

where N_{warp_block} represents the number of warps in a thread block. N_{block_SM} represents the number of blocks in each SM, which may be not a constant for different SMs because blocks are assigned to SMs in a round-robin manner. However, no matter how the number of blocks and threads are changed, we must ensure that the total parallelism of the current program remains unchanged. As presented in Equation (15), n represents the number of SMs in the current GPU.

4.2.3. Register Usage

We tested our GTX 680 according to the work in [46]. Accessing a register consumes about 0.19 clock cycle per instruction, but delays may occur due to register read-after-write dependencies and register memory bank conflicts. The latency of read-after-write dependencies is approximately 24 cycles, but this latency is completely hidden on SPs that might require thousands of threads for a device of compute capability 3.x. The number of registers used by a kernel can have a significant impact on the number of resident warps and register storage enables threads to keep local variables nearby for low latency access. However, the amount of register files is restricted and they are allocated to the blocks all at once. So, if each thread block uses numerous registers, the number of active blocks on a SM will be reduced.

Without registers overflowing, the maximum number of register files is 64K per SM, and the total amount is less than the device limit. Under previous conditions, decreasing appropriately private variables in each thread can reduce register usage per thread, and thus increase the TLP per block. Using register files, we have,

$$N_{register\_block} = N_{register\_thread} \times N_{thread\_block} \tag{18}$$

$$N_{block\_SM} \le \left\lceil \frac{RegisterFileInSM}{N_{register\_block}} \right\rceil \tag{19}$$

$$N_{register\_thread} < 64 \tag{20}$$

when the number of private variables in a thread is much more than the register files it owns, register overflow will happen and private variables will be moved to other memory.

4.2.4. Shared Memory Usage

Shared memory is high-speed on-chip memory. It is effectively a user-controlled L1 cache in Fermi and Kepler. The L1 cache and shared memory share a 64 K memory segment per SM. Since Maxwell, its SM unit features a dedicated shared memory, while the L1 caching function has been moved to be shared with the texture caching function. Shared memory is a block of read/write memory which can be accessed by all the threads within the same block. By facilitating inter-thread communication, shared memory enables a broad range of applications to run efficiently on the GPU. Using shared memory is the best method to achieve inter-thread communication minimum latency.

Shared memory is a bank-switched architecture. For a device of compute capability 3.x, each bank has a bandwidth of 64 bits every clock cycle. If every thread in a warp reads the same bank address, it triggers a broadcast mechanism to all threads within the warp, which can avoid band conflict. However, if we have any other pattern, we end up with bank conflicts of varying degrees [33]. Using shared memory, we have,

$$N_{block\_SM} \le \left\lceil \frac{SharedMemInSM}{N_{sharedMem\_block}} \right\rceil \tag{21}$$

$$N_{sharedMem\_stastic} + N_{sharedMem\_dynamic} \le SharedMemInSM \tag{22}$$

when the kernel meets Table 3 and Equation (13)-(22), we can obtain appropriate TLP configuration. For general cases, the higher TLP the programmer can achieve, the faster the program will be executed. However, sometimes if the number of active warps is limited, the ability of the SM to hide latency also dramatically drops. At some point, this will hurt the performance, especially if the warp is still making global memory accesses. If the programmer wants to increase performance further, it is recommended that larger memory transactions are made or ILP is introduced, that is, use a single thread to process more than one element of the dataset. If we use ILP, it maximally hides latency by using ILP to increase the number of operations for every active thread. Furthermore, it makes the data acquisition operation from the global memory drop significantly, reducing the refresh rate of the memory and further lowering energy consumption.

4.3. Fine-Grained Instruction-Level Parallelism

Using ILP, the number of parallel instructions processed by each thread increased and the number of elements processed per thread also increased, so the number of TLP per blocks required for the same number of tasks is reduced. The number of blocks per SM decreases which means that the proportion of register files, shared memory and L1 Cache that can be allocated per block is improved. It is in some cases possible to fully cover latency with even fewer warps, notably via ILP [47]. The detailed patterns of ILP are as follows.

4.3.1. Loop Unrolling

Loop unrolling is a common compiler optimization technique that ensures there are a reasonable number of data operations for the overhead of running through a loop.

In terms of the ratio of memory operations to arithmetic operations (400+ cycles for memory operations, 20 cycles for arithmetic operations), the ideal ratio that is required is at least 20:1. That is, for every memory fetch the kernel makes from global memory, it does 20 or more other instructions. However, loops without unrolling always are very inefficient because they can generate branches, and resulting pipeline stall. What's more, they often waste instructions and do not perform any useful work. The cost of loops include: loop counter increment, loop termination test and iterative branch judgment, and computing the memory address each time.

Using loop unrolling can increase ILP and decrease the total number of iterative times. Therefore, it is able to decrease useless instructions and loop overhead. The NVCC compiler supports the #**pragma unroll** directive, which instructs the compiler to automatically unroll all constant loops.

4.3.2. Using Vector Type

Using the variable of vector type simply requires declaring the variable as vector_N type, for example, int2, int4, float2, float4, because CUDA supports the built-in vector type (in vector_types.h). The programmer can also create new types, such as uchar4, and define new operators. In effect, each vector_N type is a structure which contains N elements of the base type.

After using the vector type, it can increase the size of a single element, so it is able to process more than one element per thread. A certain amount of ILP can also be introduced.

Vector-based operations can allocate the cost of related operations to multiple operations rather than one, and can reduce the number of loop iterations. Correspondingly, the number of memory requests to the memory system is reduced, and the scale of request becomes larger. Using vector type will increase register files usage per thread, which in turn reduces the number of reside blocks for each SMX. It will further improve cache utilization, thereby reducing the global memory read / write transactions, and memory throughput will also increase as the total number of memory transactions decline.

4.3.3. Register Reuse

Register files are the fastest storage mechanism on the GPU. They are the only way of achieving anything like the peak performance of the device. However, they are limited in their availability. As previously mentioned, it is better to divide a task with lots of operations into many independent parts. If so, we can use a set of registers to compute independent problems, and it is able to maximize register usage.

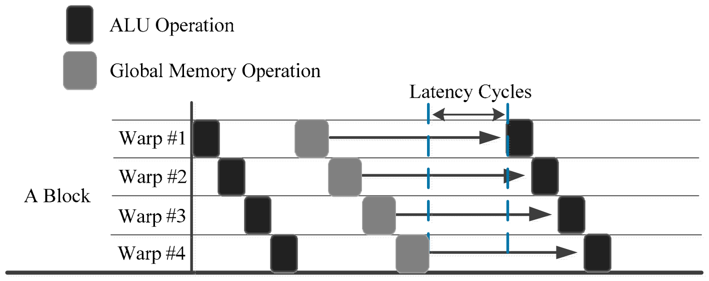
GPU uses a lazy evaluation model [33], only accessing memory for a variable when it needs the contents of the variable. By bringing the assignment and usage of a variable closer together, it enables the compiler to reuse registers. Therefore, at the start of the kernel, x, y and z might be assigned. If, in fact, they are only used later in the kernel, we can move the declaration and assignment of a register variable to the location where it can be used to realize register reuse, and thus reduce register usage. Then, the compiler is able to deal with these three independent variables being required in different and disjoint periods by simply using a single register.

By expanding the number of output data set elements of a single thread, it contributes to both MB type and CB type applications. This will increase register usage, but the number of threads being scheduled must be monitored to ensure that they do not suddenly drop off.

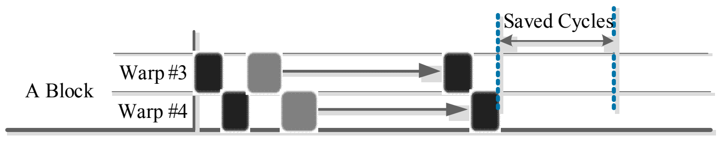
4.4. Execution Order Simulation

As a cost-effective alternative, we propose the TLPOM. This model takes full advantage of the fact that on-chip memory resources are not completely used. Therefore, we can simultaneously use TLP and ILP technology to increase the number of instructions in each thread and decrease the total number of threads per block to avoid excessive resource competition.

Figure 7 simulates the execution order of different ILPs using the TLPOM. Figure 7(a) presents the execution order with ILP0, and we assume for illustrative purposes that each block can issue four warps concurrently at this moment. After issuing a series of instructions from each warp in a round-robin order, each warp hits a global memory access stall [42], whereas in Figure 7(b), the value of ILP1 is twice as large as ILP0, so the number of instructions of each thread per warp/block is doubled. In view of the fact that the total number of tasks has not changed, the total number of threads per block are halved.



(a)



(b)

Figure 7. Inter-thread execution order simulation: (a) Execution order with ILP0; (b) Execution order with $\mathrm{ILP}_1 = 2 \times \mathrm{ILP}_0$.

With our TLPOM, we reduce long latency using much less warps per block, as shown in Figure 7(b). It also follows a round-robin order, but it has more memory throughput than ILP0. When executing the same number of tasks, it is able to exploit less warps (threads) which own more instructions concurrently per thread. Hence, the long latency operations can be more effectively hidden with our model. The intra-thread execution behavior is simulated as shown in Figure 8, showing the different instructions which are issued when ILP is changed for each warp per clock.

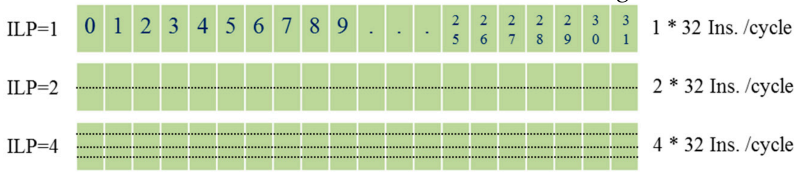


Figure 8. Intra-thread execution order simulation.

When ILP = 1, this means that every thread can only issue one instruction per cycle, and there are a total of 32 instructions in a warp; ILP = 2 means each thread can simultaneously issue two instructions per cycle, and there are a total of 64 instructions in a warp. When ILP = 4, it means a warp can issue 128 instructions per cycle. If there are sufficient ALU, Load/Store unit, SP and other resources, the more issued instructions per clock, the less time it takes to execute the same tasks. Of course, if we use varying degrees of ILP, it requires the simultaneously issued instructions to be independent.

5. Reliable GPU Cluster Architecture

System reliability represents the ability of the node to complete a defined function within specified conditions and a specified time. If the node fails to complete the prescribed function, it is called failure. Our computing system has multiple metrics, such as useful life, mean time between failures (MTBF) and mean time to repair (MTTF). Here we use MTBF. For repairable n products, assuming that N_0 times failures happen during execution, after being repaired, and continues to be put into use, working hours were $t_{1,} t_{2,} \cdots, t_{N_0}$ of each time, respectively. So the MTBF of the product is:

$$MTBF = n \cdot \frac{1}{N_0} \cdot \sum_{i=1}^{N_0} t_i \tag{23}$$

The computing system we are considered is a GPU cluster, where the GPU-enabled computing nodes can communicate through any message-passing standard, particularly the message passing interface (MPI). In the previous section, we committed to improve the algorithm's performance for a more rapid and real-time data processing environment, but such a system is prone to failure if it runs for a long time. Hence, a problem which cannot be ignored is how to ensure the system provides 24/7 uninterrupted and high-reliability services. Hence, to achieve this we develop a reliable GPU cluster architecture (RGCA) for IoT computing.

5.1. GPU Cluster Reliability Scheduling Under Limited Resources

To guarantee system reliability, each computing node and its components should be assigned a proper workload to avoid overusing a component with limited GPU resources. For instance, if a task needs a large amount of shared memory but it is assigned to a GPU with a small amount of shared memory, there will be a considerable amount of data which should have been stored in shared memory which has to move to global memory, which will cause the performance to drop. Taking into account the possible constraints, it is necessary to build a model which can infer the target GPU to which a task can be assigned. The main conditions are listed as follows:

1. The time that node i has executed, which is $T_{active}(i)$;

$$T_{active}(i) = time\_now(i) - time\_start(i) \tag{24}$$

2. The time that node i has been idle, which is $T_{idle}(i)$;

$$T_{idle}(i) = time\_now(i) - time\_end(i) \tag{25}$$

3. The various memory resources belonging to node i, which is $\sum R\_Component(i)$;

$$\begin{aligned}\sum R\_Component(i) = R\_Register(i) + R\_Shared(i) + R\_Global(i) + R\_Constant(i) \\ + R\_Texture(i) + R\_Constant\_cache(i) + R\_Texture\_cache(i)\end{aligned} \tag{26}$$

4. The different memory resources that are needed by task j, which is $\sum R\_task(j)$;

$$\begin{aligned}\sum R\_task(j) = R\_Register(j) + R\_Shared(j) + R\_Global(j) + R\_Constant(j) \\ + R\_Texture(j) + R\_Constant\_cache(j) + R\_Texture\_cache(j)\end{aligned} \tag{27}$$

Not all resources will be used by the kernels, and the resources used by different tasks are different.

5. Whether the GPU is qualified for the task or not, which can be determined by our previous research. For example, although the GTX 280 supports the atomic operation, it doesn't support the floating point atomic operation. Hence, the optimized K-means cannot be assigned to GTX280 because it has float-point atomic add operations after optimizing.
6. All of the above relationships need to be met so that:

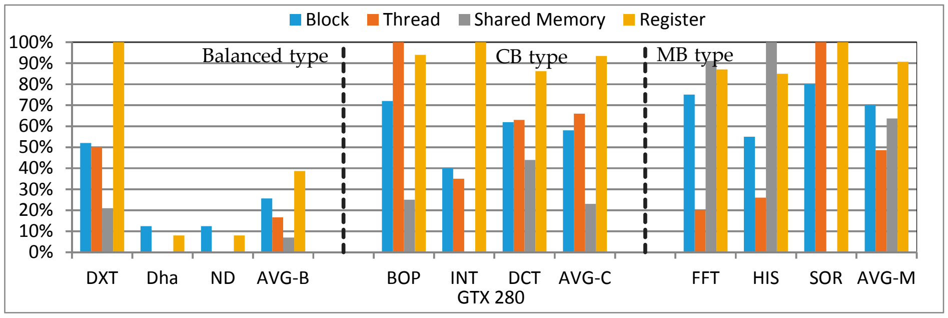
$$\begin{cases} Z = max(MTBF(i)) \\ i \in N_j \\ \sum R\_task(j) \le \sum R\_Component(i) \\ T\_active(i) \le MTBF(i) \\ T\_idle(i) \ge T\_recover(i) \end{cases} \tag{28}$$

In Equation (29), symbol i means the ith node in a cluster in this paper, and $i \in [1, 2, \ldots, n]$ in this paper. Symbol j means the jth task being executed in the cluster, and $j \in [1, 2, \ldots, n]$. N_j represents the node set which contains all the available nodes for task j. $MTBF(i)$ represents the mean time between the failure of node i, and we can obtain its value via multiple experiments and Equation (24). $T\_recover(i)$ represents the recovery time of node i, and this period ensures that the temperature of the node is returned to its idle state. The objective function Z means maximizing the MTBF for each node in the cluster. If the aim is achieved, the number of failures will be reduced which further guarantees the reliability of the computing system. The temperature is tested by GPU-Z under 26 degrees Celsius. The resource required by a program must be less than the corresponding resource available to the node, such as register files and shared memory.

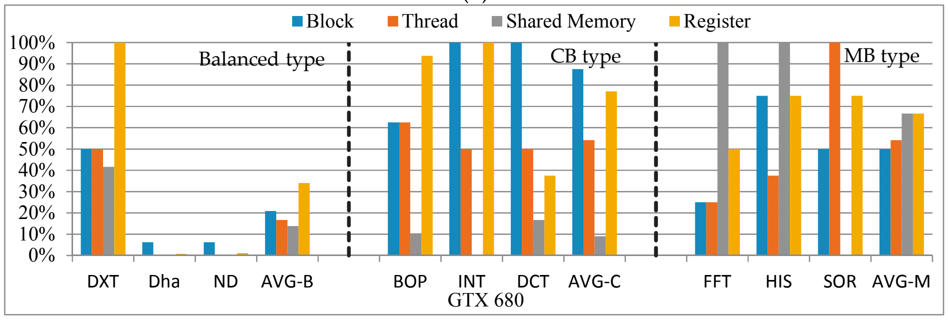
5.2. Determine Target Node for High Reliability

In addition to quantifying a variety of resources in the cluster, for different type kernels, we also need to statistically analyze the resource usage of different nodes. This analysis can assist the reliability scheduling of our cluster. We quantify four kinds of primary resources, as shown in Figure 9. If any of the resource usage reaches its limit, no more blocks can be scheduled even though all the other resources are still available. Figure 9 shows the fraction of SM resources used by primary kernel/kernels (execution duration more than 70%) of each algorithm.

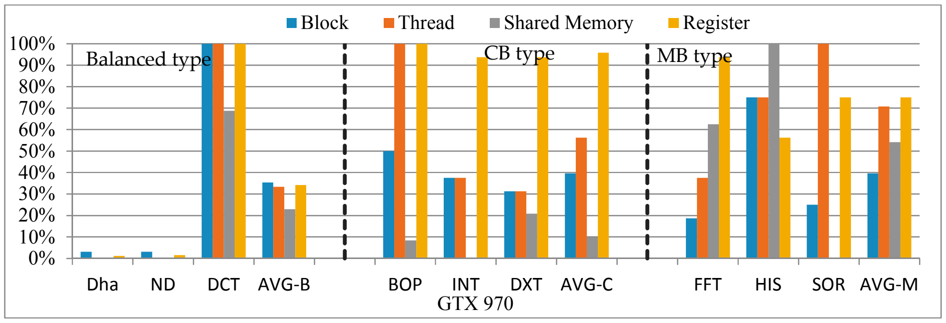
In Figure 9, we statistics 9 programs from different experiment suites, including CUDA SDK, Rodinia3.1 [48] and SHOC suite [49]. From these results, we can draw the following conclusions:



(a)



(b)



(c)

Figure 9. The resource utilization of different type kernels on different GPUs. (a) Resource utilization on GTX 280; (b) Resource utilization on GTX 680; (c) Resource utilization on GTX 970.

1. In terms of the balanced type kernel, for these GPUs, the utilization of four kinds of resources is relatively balanced, and the average utilization of each resource is significantly lower than the other two types of kernels. In this case, if more than one node can be assigned to the balanced type kernel according to Section 5.1, firstly we choose the lower architecture to fully exploit the resources (select 280 instead of 670/680);
2. In terms of the CB type kernel, the utilization of four kinds of resources is unbalanced. In particular, the average utilization of register files is significantly higher than other resources, and the utilization of shared memory is the lowest among them. In this case, firstly we choose the node who has the most register files available (select 680 instead of 280);
3. In terms of the MB type kernel, the utilization of four kinds of resources is also relatively balanced, but it is obviously higher than the balanced type kernel. Furthermore, both register files and shared memory have higher utilization. In this case, we select the most advanced node for the CB type kernel (select 970 instead of 680).

We also analyze our 8 algorithms, and the above conclusions are also applicable to our algorithms. In order to ensure a high-performance and high-reliability computing system, we undertake a comprehensive evaluation in the following two subsections. If more than one GPU can be available for a specified task according to Section 5.1, we choose the final scheme considering the type of kernels.

6. Evaluation

6.1. Experimental Environment

The experiment platform comprises two parts: GPU computing cluster and its energy consumption monitoring system based on WSN. The monitoring system provides a measured current to obtain the energy consumption of each node, and the primitive current value is sent to a data-log database every 10 milliseconds (an adjustable interval). We measured the idle current and idle temperature of the GPUs, as shown in Table 4. Here, the computing cluster includes one master node and 8 computing nodes, and each is configured with a 4-core CPU, Inter (R) Core (TM) i5-3470 CPU @ 3.20 GHz, 8 G memory, 500 G hard disk, Ubuntu 14.04 OS, and the node distribution of the cluster is shown in Table 4.

Table 4. GPU Cluster node configuration information.

Node	Number	Idle Current (A)	Idle Temperature (℃)
GTX 280	2	2.651	56
GTX 670	1	1.949	41
GTX 680	3	1.934	40
GTX 970	2	1.065	36

As shown in Table 4, the GTX 670 also belongs to Kepler. The other related information on these nodes was shown in Table 2 and Table 3. The node deployment is shown in Figure 10.

(a) (b)

Figure 10. GPU cluster: (a) The GTX 280 and 670/680 cluster; (b) The GTX 970 cluster.

We also use the largest available input set for each application since some programs dispatch only a small number of thread blocks with a small input set, which results in an artificially high resource underutilization.

6.2.Consequences

The test commences with the experimental environment and parameters set as aforementioned. We first rewrite and optimize all the algorithms using our TLPOM. Without loss of generality, both algorithms are executed ten times independently to obtain the average statistical results in each experiment. The experiment is designed with consideration to following aspects:

1. In a situation where the algorithms' performance is steadily increasing, the performance of the original algorithms and our optimized algorithms after using the TLPOM are compared;

2. The data needs to be processed per second from the real-time data packets of 420000 families, and each packet size is 125 bytes. As a result, the amount of data generated by the access system is approximately 48M per second. In order to verify the compute capacity of our system, we increase the amount of data processed to one million packets, that is, the amount of data is 119.2 million bytes;
3. In order to guarantee the accuracy of the experiments, we set up a number of TLP and ILP. The value of TLP can be set as 32, 64, 128, 256, …, 1024 and the value of ILP can be set as 1, 2, 4, 8, …, 32. But in practice, not all algorithms can take all of these values;
4. In order to compare different performance and energy consumption behaviors of different nodes for the same application in the GPU cluster, we undertake step 1 and step 2 using three different kinds of computing nodes.

6.2.1. Algorithms' Performance Analysis on a Single Computing Node

To validate the advantages of our algorithms' performance, we perform an exhaustive launch configuration exploration across all TLP and ILP combinations for our 8 algorithms. We previously stated that the optimized K-means cannot be executed in GTX 280 because of its floating-point atomic operation. For each of these architectures, we compare the baseline performance and its optimized (TLPOM) performance. All the results are normalized relative to the baseline performance. What we need to emphasize is that the baseline performance of the same algorithm is changed along with the GPU. For statistical convenience, we only present one set of baseline performances in Figure 11, but its value is changed along with different GPUs. That is to say, the value of the optimized performance of KM in GTX 680 is not necessarily better than in GTX 970.

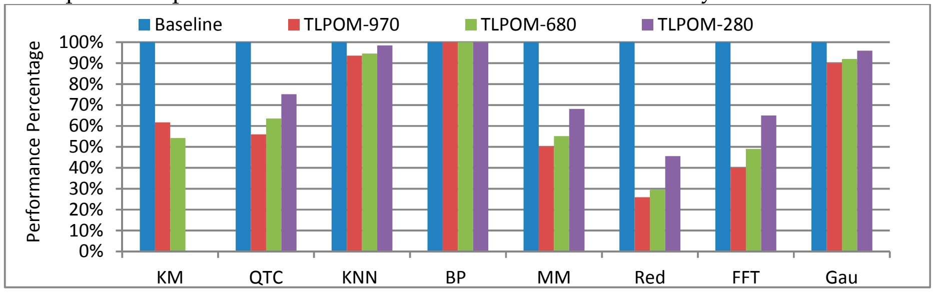


Figure 11. Algorithms' performance with different GPUs.

From the figure, it can be seen that when using TLPOM, most of the algorithms display a significant improvement in performance. In terms of GTX 970, the performance of K-means increased by 40% and Matrixmul also increased by 49.9%. For the other two GPUs, their performance also increased by different degrees. However, we observe that TLPOM is not applicable to BPNN. This is because BPNN has a lot of neurons (input layers, output layers and hidden layers) which add complexity to BPNN. Furthermore, the number of blocks and the number of threads per block are fixed, so our optimized model is not applicable to it.

In order to explore the reasons why TLPOM can deliver performance improvement, we take K-means and Matrixmul as examples for an in-depth analysis with GTX 680. Figure 12 presents the related instructions per clock (IPC) issued and the executed behaviors with different TLP and ILP. IPC means the achieved instructions throughputs per SM, and the theoretical maximum peak IPC is defined by the compute capabilities of the target device.

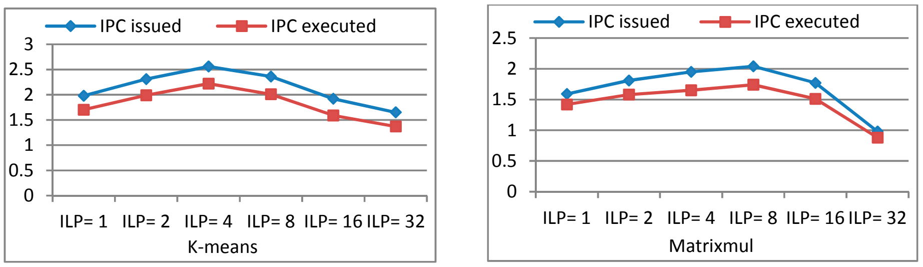
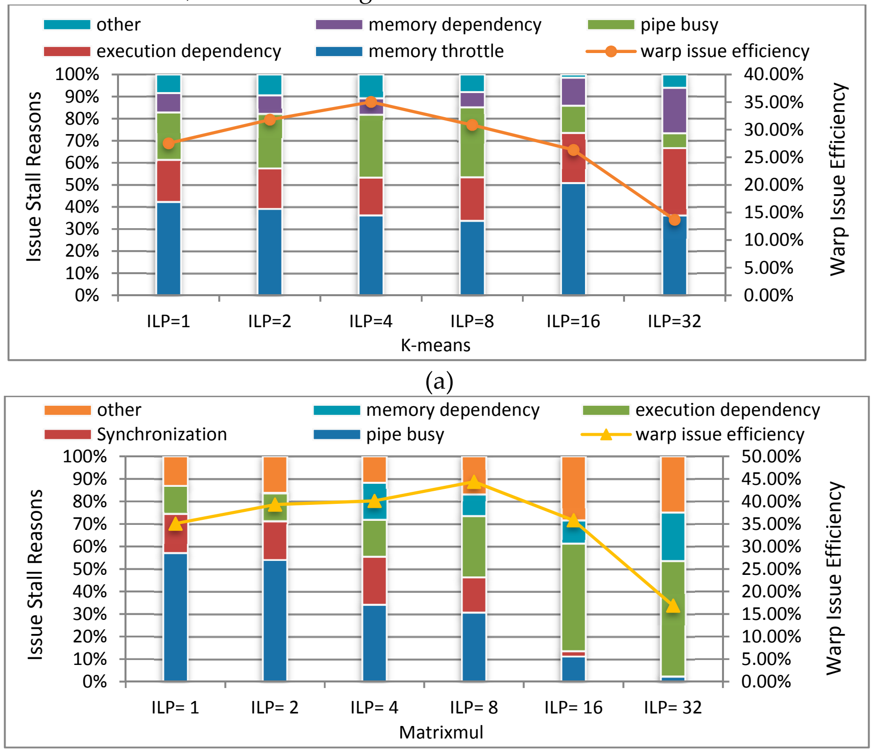


Figure 12. IPC issued and executed behaviors: (a) IPC of K-means; (b) IPC of Matrixmul.

With the changing of ILP (TLP is also changed), the issued IPC (the average number of issued instructions per cycle) and executed IPC (the average number of executed instructions per cycle) of the programs have corresponding changes. When the program has a higher issued IPC, more instructions can be issued per clock, so latency becomes short. A higher executed IPC indicates the more efficient usage of the available resources, and the programs have better performance. The maximum achievable target IPC for a kernel is dependent on the mixture of the instructions executed. From Figure 12, we know that K-means and Matrixmul can issue maximum instructions per clock when they have 4 and 8 independent instructions in a single thread, respectively. At their respective highest executed IPC points, they can achieve the best performance. Another important factor in performance improvement is the change of warp issue efficiency. This represents the percentage of cycles which have eligible warps across all cycles over the duration of kernel execution. When we use the appropriate TLP and/or ILP, both the warp issue efficiency and eligible warps per clock are increased, as shown in Figure 13.



(b)

Figure 13. Warp issue efficiency and warp issue stall reasons: (a) KM warp stall reasons; (b) MM warp stall reasons.

Figure 13 shows the maximum values of warp issue efficiency at the best performance point. There are multiple reasons resulting in a warp scheduler having no eligible warps to select from so therefore it does not issue an instruction. These are called Issue Stall Reasons. Figure 13(a) gives the warp issue stall reasons of K-means, that is, the reasons that capture why an active warp is not eligible. It is clear that the warp stall reasons change with the changeability of ILP (TLP can also be changed). Firstly, it has the highest warp issue efficiency (35.03%) when ILP = 4, as can be seen from the deputy coordinates. Secondly, the percentage of execution dependency and memory dependency are lowest at ILP =4 because the right combination of TLP and ILP can reduce issue stall. The data in Figure 13(b) further verify our views that the maximum warp issue efficiency is when ILP = 8. At this moment, the percentage of execution dependency and memory dependency are both lower than the others, so execution dependency stalls can potentially be reduced by increasing ILP. Using loop unrolling, loading vector types are also a way to increase the scale of memory transactions. Therefore, the warp issue efficiency can be increased.

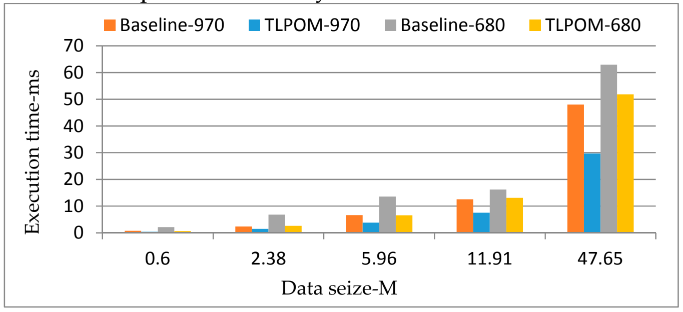


Figure 14. The performance of KM with different data size.

We also analyzed the performance of different amounts of data. We take K-means as the example to explain the results. As shown in Figure 14, our TLPOM works for different amounts of data for K-means, and the size of the data that needs to be processed does not affect the effect of our TLPOM. This conclusion also applies to the other 7 optimized algorithms. Though the results of Figure 12 to Figure 13 are obtained in GTX 680, these conclusions are also applicable to other GPUs in our computing cluster.

6.2.2. Algorithms' Energy Consumption Analysis on a Single Computing Node

We presented the related performance results in the previous section. Here, we analyze the energy consumption of the optimized algorithms and the results are shown in Figure 15.

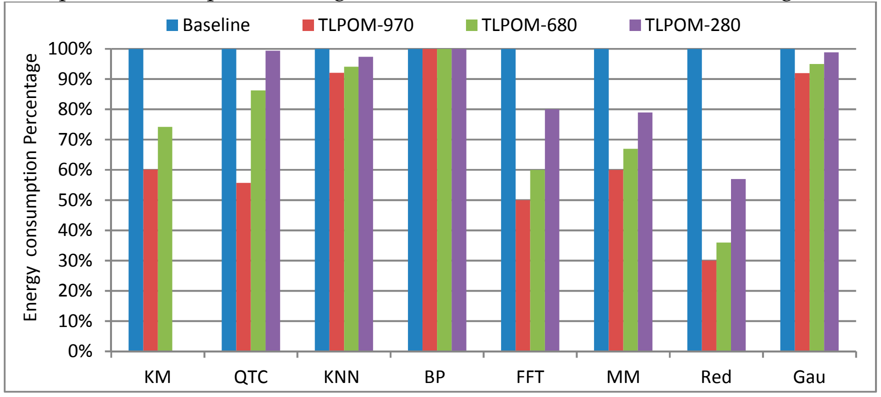


Figure 15. Energy consumption of data mining algorithms.

The results are also normalized as in Figure 11, and the values of the baseline energy consumption of the three GPUs are different. The figure shows that the energy consumption of these algorithms is almost inversely proportional to their performance. Except for BPNN, all the others have different degrees of energy consumption decline, especially Matrixmul and Reduction. Why does the energy consumption of these algorithms reduce after adjusting the TLP and ILP? Because the TLPOM increases the operations/elements of a single thread to process and also increases the register files and the usage of other on-chip memory structures. Furthermore, it makes the data acquisition operation from the global memory drop significantly, reducing the refresh rate of the device memory and further lowering energy consumption.

6.2.3. Reliability Scheme of the System

In Section 5, we analyzed in detail the RGCA with *n* (*n* is a finite positive integer) computing nodes, and here we detail and verify our reliability scheme. Firstly, in order to obtain the MTBF of the cluster and consider the feasibility of the experiments, we execute three different types of kernels for 720 hours (the execution time is adapted to all reliability experiments) using all the nodes in the cluster, respectively. If an application fails on a node, the node will stop to rest until it is restored to the original state. We choose three of the algorithms as representatives, the MB type program – KM (in which the KM on GTX 280 is replaced by the same type -- MM), the CB type program -- BPNN and the balanced type program – KNN. Table 5 and Table 6 detail the fault node information of the original algorithms and the optimized algorithms in the cluster.

Table 5. Fault node information of original algorithms.

GPU\Algorithms	K-means	BPNN	KNN
GTX 280	4	3	3
GTX 670	1	1	0
GTX 680	1	0	0
Sum (times)	6	4	3

Table 6. Fault node information of optimized algorithms.

GPU\Algorithms	K-means	BPNN	KNN
GTX 280	3	2	2
GTX 670	1	0	0
GTX 680	0	0	0
Sum (times)	4	2	2

The two tables show the fault information in the cluster. For example, in terms of the original K-means, the GTX 280 cumulatively fails 4 times during the 720-hour period in the cluster. But after being optimized, the number of failures in the cluster is significantly reduced. For the optimized KM, the GTX 280 cumulatively fails 3 times during the experiment. Fortunately, for all of the experiments, the GTX 970 is able to operate reliably without any failures. This is because the GTX 970 was designed to provide an extraordinary leap in energy efficiency and deliver unrivaled performance while simultaneously reducing energy consumption from the previous generation.

Table 7. The MTBF of different types of applications in the cluster.

Algorithms (Type)	K-means (mem-bound)	BPNN (com-bound)	KNN (balanced)
Original MTBF(h)	960	1440	1920
TLPOM MTBF(h)	1440	2880	2880

Combining Table 5, Table 6 and Equation (24), we can get the MTBF of the cluster, as shown in Table 7. From this, we know that the MTBF will be changed after optimizing the algorithms. For the MB type program, KM has the lowest MTBF if it is assigned to GTX 280, regardless of whether it has been optimized or not. The balanced application, KNN has the highest MTBF regardless as to what kind of GPU it is assigned. Therefore, to obtain high reliability, we reallocate these algorithms. Firstly, we assign the MB type algorithms to 970 to avoid the lower MTBF at the cluster, and then assign the balanced type algorithms to the nodes with lower compute capacity.

Table 8. Reliability contrasted against the baseline model.

Scheme\GPU		280	280	670	680	680	680	970	970
Original	Scheme 1	BPNN	FFT	Gau	KM	KNN	MM	QTC	Red
	Fault node (times)	2	2	0	0	0	0	0	0
RGCA	Scheme 2	QTC	Gau	KNN	FFT	Red	BPNN	MM	KM
	Fault node (times)	1	1	0	0	0	0	0	0
	Scheme 3	KNN	QTC	Gau	FFT	Red	MM	BPNN	KM
	Fault node (times)	1	1	0	0	0	0	0	0

In Table 8, we compare the reliability of the original model with our RGCA. For Scheme 1, we use the original algorithms without any performance optimization, and these algorithms are randomly assigned to different nodes in our cluster (we allocate them in the order of the first letter of the algorithms). In this case, the cluster cumulatively fails four times during the experiment. But for the RGCA system, we have strict distribution standards according to Section 5. Here we only list two of them. In this case, Scheme 2 and Scheme 3 each fail twice. Compared to the original Scheme 1, the number of failures dropped by 50% in our two reliability schemes. This is a significant reliability improvement.

However, a non-ignorable problem is that the GTX 280 causes the node to fail in Scheme 2 and Scheme 3. This is because the compute capacity of the GTX 280 is only 1.3, and its compute resources and memory resources are lower than the other GPUs in the cluster. Fortunately, through our RGCA, we can reduce the node failure rate. So we can use the lower GPU to reduce costs and save computing power in our cluster. Finally, for different types of data mining algorithms of IoT computing, we can obtain a low-cost, high-performance and high-reliability computing system by using the heterogeneous cluster.

7. Conclusions

The Data Access System for the Internet of Things (DASIoT), as a crucial part of the sensor network application, plays a determinative role in large-scale sensor data access and processing. To cope with the massive original access data in the IoT, one of the most important technologies is data mining, because it can convert the data generated or captured by DASIoT into useful information and further provide a more convenient environment for the user.

In this paper, our goal is to provide a low-cost, high-performance and high-reliability computing system for common data mining algorithms in IoT computing. Our optimization scheme is divided into two parts. Firstly, we provide a two-level parallel optimization model (TLPOM)-based CUDA programming and to improve the performance of these algorithms. Then in order to obtain a long-term, error-free runtime environment, we put forward a reliable GPU cluster architecture (RGCA) under limited resources. In the process of optimizing performance, we assign a different number of blocks/threads per grid/block to acquire the best launch configuration, and we also use common compiler optimization techniques, for example loop unrolling. On the basis of the performance improvements, we define the GPU cluster energy consumption calculation model by capturing the real-time current to calculate the energy consumption. Then, in with full consideration of the compute capacity, GPU resources and the characteristics of algorithms, we obtain a high-reliability computing system. Our experiments demonstrate the effectiveness of our optimization scheme.

Further work will cover an extension of the proposed high-reliability computing system, and on this basis, we will study in-depth the reliability scheduling and fault recovery scheme for IoT computing. Further work will also take into consideration the GPU computing cluster to undertake large-scale real-time data processing.

Acknowledgments: The authors gratefully acknowledge the support of the National Natural Science Foundation of China (Grant Nos. 61572325 and 60970012), Ministry of Education Doctoral Fund of Ph.D. supervisor of China (Grant No. 20113120110008), Shanghai Key Science and Technology Project in Information Technology Field (Grant Nos. 14511107902 and 16DZ1203603), Shanghai Leading Academic Discipline Project (No. XTKX2012), Shanghai Engineering Research Center Project (Nos. GCZX14014 and C14001).

Author Contributions: Qingkui Chen proposed the ideas and designed the architecture while Yuling Fang conducted the algorithm optimization. Qingkui Chen and Yuling Fang conceived and designed the experiments. Yuling Fang, Deyu Zhao and Jingjuan Wang performed the experiments and analyzed the data, results. Qingkui Chen and Neal N.Xiong served as advisors to the above authors and gave suggestions on experiments, performance evaluation. Yuling Fang wrote the paper. All authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, S. C.; Xu, L. D.; Zhao, S. S. The internet of things: a survey. *Inform Syst Front* **2015**, *17* (2), 243-259, 10.1007/s10796-014-9492-7. <Go to ISI>://WOS:000351520100002.
2. Xu, L. D.; He, W.; Li, S. C. Internet of Things in Industries: A Survey. *Ieee T Ind Inform* **2014**, *10* (4), 2233-2243, 10.1109/Tii.2014.2300753. <Go to ISI>://WOS:000344995800025.
3. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener Comp Sy* **2013**, *29* (7), 1645-1660, 10.1016/j.future.2013.01.010. <Go to ISI>://WOS:000320635700001.
4. Xiong, N.; Huang, X.; Cheng, H.; Wan, Z. Energy-efficient algorithm for broadcasting in ad hoc wireless sensor networks. *Sensors* **2013**, *13* (4), 4922-4946.
5. Vasilakos, A. V.; Li, Z.; Simon, G.; You, W. Information centric network: Research challenges and opportunities. *J Netw Comput Appl* **2015**, *52*, 1-10, 10.1016/j.jnca.2015.02.001. <Go to ISI>://WOS:000354589100001.
6. Fang, S. F.; Xu, L. D.; Zhu, Y. Q.; Ahati, J.; Pei, H.; Yan, J. W.; Liu, Z. H. An Integrated System for Regional Environmental Monitoring and Management Based on Internet of Things. *Ieee T Ind Inform* **2014**, *10* (2), 1596-1605, <Go to ISI>://WOS:000336669800074.
7. Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for Smart Cities. *Ieee Internet Things* **2014**, *1* (1), 22-32, <Go to ISI>://WOS:000209672000004.
8. Jin, J.; Gubbi, J.; Marusic, S.; Palaniswami, M. An Information Framework for Creating a Smart City Through Internet of Things. *Ieee Internet Things* **2014**, *1* (2), 112-121, <Go to ISI>://WOS:000209672100001.
9. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Sensing as a service model for smart cities supported by Internet of Things. *T Emerg Telecommun T* **2014**, *25* (1), 81-93, <Go to ISI>://WOS:000330794800008.
10. Abbas, Z.; Yoon, W. A Survey on Energy Conserving Mechanisms for the Internet of Things: Wireless Networking Aspects. *Sensors* **2015**, *15* (10), 24818-24847, <Go to ISI>://WOS:000364242300007.
11. Rani, S.; Talwar, R.; Malhotra, J.; Ahmed, S. H.; Sarkar, M.; Song, H. B. A Novel Scheme for an Energy Efficient Internet of Things Based on Wireless Sensor Networks. *Sensors* **2015**, *15* (11), 28603-+, <Go to ISI>://WOS:000365686400062.
12. Hussain, S.; Matin, A. W.; Islam, O. Genetic algorithm for energy efficient clusters in wireless sensor networks, In Information Technology, 2007. ITNG'07. Fourth International Conference on, IEEE: 2007; pp 147-154.
13. Yi, M.; Chen, Q. K.; Xiong, N. N. An Effective Massive Sensor Network Data Access Scheme Based on Topology Control for the Internet of Things. *Sensors* **2016**, *16* (11), <Go to ISI>://WOS:000389641700079.
14. Prinsloo, J.; Malekian, R. Accurate vehicle location system using RFID, an Internet of Things approach. *Sensors-Basel* **2016**, *16* (6), 825.
15. Tsai, C. W.; Lai, C. F.; Chiang, M. C.; Yang, L. T. Data Mining for Internet of Things: A Survey. *Ieee Commun Surv Tut* **2014**, *16* (1), 77-97, <Go to ISI>://WOS:000338700900006.
16. Chen, F.; Deng, P.; Wan, J. F.; Zhang, D. Q.; Vasilakos, A. V.; Rong, X. H. Data Mining for the Internet of Things: Literature Review and Challenges. *Int J Distrib Sens N* **2015**, <Go to ISI>://WOS:000361197700001.
17. Wu, X. D.; Zhu, X. Q.; Wu, G. Q.; Ding, W. Data Mining with Big Data. *Ieee T Knowl Data En* **2014**, *26* (1), 97-107, <Go to ISI>://WOS:000327656800009.
18. Madden, S. From Databases to Big Data. *Ieee Internet Comput* **2012**, *16* (3), 4-6, <Go to ISI>://WOS:000303277200001.
19. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context Aware Computing for The Internet of Things: A Survey. *Ieee Commun Surv Tut* **2014**, *16* (1), 414-454, <Go to ISI>://WOS:000338700300020.

20. Baraniuk, R. G. More Is Less: Signal Processing and the Data Deluge. *Science* **2011**, *331* (6018), 717-719, <Go to ISI>://WOS:000287205700055.
21. Ng, R. T.; Han, J. W. CLARANS: A method for clustering objects for spatial data mining. *Ieee T Knowl Data En* **2002**, *14* (5), 1003-1016, <Go to ISI>://WOS:000177745200006.
22. Zhang, T.; Ramakrishnan, R.; Livny, M. BIRCH: an efficient data clustering method for very large databases, In ACM Sigmod Record, ACM: 1996; pp 103-114.
23. Hammouda, K. M.; Kamel, M. S. Efficient phrase-based document indexing for web document clustering. *Ieee T Knowl Data En* **2004**, *16* (10), 1279-1296, <Go to ISI>://WOS:000223253200008.
24. Shen, B.; Liu, Y.; Wang, X. Research on data mining models for the internet of things, In Image Analysis and Signal Processing (IASP), 2010 International Conference on, IEEE: 2010; pp 127-132.
25. Jain, A. K. Data clustering: 50 years beyond K-means. *Pattern Recogn Lett* **2010**, *31* (8), 651-666, <Go to ISI>://WOS:000277552600002.
26. Wu, X. D.; Kumar, V.; Quinlan, J. R.; Ghosh, J.; Yang, Q.; Motoda, H.; McLachlan, G. J.; Ng, A.; Liu, B.; Yu, P. S.; Zhou, Z. H.; Steinbach, M.; Hand, D. J.; Steinberg, D. Top 10 algorithms in data mining. *Knowl Inf Syst* **2008**, *14* (1), 1-37, <Go to ISI>://WOS:000251795900001.
27. Choubey, P. K.; Pateria, S.; Saxena, A.; Chirayil, S. B. V. P.; Jha, K. K.; Basaiah, S. Power Efficient, Bandwidth Optimized and Fault Tolerant Sensor Management for IOT in Smart Home. *Ieee Int Adv Comput* **2015**, 366-370, <Go to ISI>://WOS:000380493300073.
28. Bouman, C. A.; Shapiro, M.; Cook, G.; Atkins, C. B.; Cheng, H., Cluster: An unsupervised algorithm for modeling Gaussian mixtures. 1997.
29. Xiao, H. Towards parallel and distributed computing in large-scale data mining: A survey. *Technical University of Munich, Tech. Rep* **2010**.
30. Bagheri, B.; Ahmadi, H.; Labbafi, R. Application of data mining and feature extraction on intelligent fault diagnosis by artificial neural network and k-nearest neighbor, In Electrical Machines (ICEM), 2010 XIX International Conference on, IEEE: 2010; pp 1-7.
31. Nvidia, C., C Best Practices Guide, 2012. 2012.
32. GTX, N. G., 680: The fastest, most efficient GPU ever built. Whitepaper, NVIDIA: 2012.
33. Cook, S., *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes: 2012.
34. Kayıran, O.; Jog, A.; Kandemir, M. T.; Das, C. R. Neither more nor less: optimizing thread-level parallelism for GPGPUs, In Proceedings of the 22nd international conference on Parallel architectures and compilation techniques, IEEE Press: 2013; pp 157-166.
35. CPU power dissipation. https://en.wikipedia.org/wiki/CPU_power_dissipation.
36. Sun, D. W.; Zhang, G. Y.; Yang, S. L.; Meng, W. M.; Khan, S. U.; Li, K. Q. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. *Inform Sciences* **2015**, *319*, 92-112, <Go to ISI>://WOS:000357707700007.
37. Beloglazov, A.; Abawajy, J.; Buyya, R. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems-the International Journal of Grid Computing and Escience* **2012**, *28* (5), 755-768, <Go to ISI>://WOS:000301819800006.
38. Ma, D. S.; Bondade, R. Enabling Power-Efficient DVFS Operations on Silicon. *Ieee Circ Syst Mag* **2010**, *10* (1), 14-30, <Go to ISI>://WOS:000275668500004.
39. Chen, Q. K. L., C.F.; Chao, Q.C.; Yi, M. The Power Consumption Monitoring System of GPU Cluster. CN205983447U, 2017.
40. Rogers, T. G.; Johnson, D. R.; O'Connor, M.; Keckler, S. W. A variable warp size architecture, In ACM SIGARCH Computer Architecture News, ACM: 2015; pp 489-501.
41. Ausavarungnirun, R.; Ghose, S.; Kayiran, O.; Loh, G. H.; Das, C. R.; Kandemir, M. T.; Mutlu, O. Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, In Parallel Architecture and Compilation (PACT), 2015 International Conference on, IEEE: 2015; pp 25-38.
42. Yoon, M. K.; Kim, K.; Lee, S.; Ro, W. W.; Annavaram, M. Virtual Thread: Maximizing thread-level parallelism beyond gpu scheduling limit, In Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on, IEEE: 2016; pp 609-621.
43. Xiang, P.; Yang, Y.; Mantor, M.; Rubin, N.; Zhou, H. Many-thread aware instruction-level parallelism: architecting shader cores for GPU computing, In Proceedings of the 21st international conference on Parallel architectures and compilation techniques, ACM: 2012; pp 449-450.

44. Cappelli, R.; Ferrara, M.; Maltoni, D. Large-scale fingerprint identification on GPU. *Inform Sciences* **2015**, *306*, 1-20, <Go to ISI>://WOS:000351803800001.
45. Kim, K.; Lee, S.; Yoon, M. K.; Koo, G.; Ro, W. W.; Annavaram, M. Warped-Preexecution: A GPU Pre-execution Approach for Improving Latency Hiding. *Int S High Perf Comp* **2016**, 163-175, <Go to ISI>://WOS:000381808200014.
46. Wong, H.; Papadopoulou, M. M.; Sadooghi-Alvandi, M.; Moshovos, A. Demystifying GPU Microarchitecture through Microbenchmarking. *Int Sym Perform Anal* **2010**, 235-246, Doi 10.1109/Ispass.2010.5452013. <Go to ISI>://WOS:000393518300030.
47. Volkov, V. Better performance at lower occupancy, In Proceedings of the GPU technology conference, GTC, San Jose, CA: 2010; p 16.
48. Che, S.; Boyer, M.; Meng, J.; Tarjan, D.; Sheaffer, J. W.; Lee, S.-H.; Skadron, K. Rodinia: A benchmark suite for heterogeneous computing, In Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on, Ieee: 2009; pp 44-54.
49. Danalis, A.; Marin, G.; McCurdy, C.; Meredith, J. S.; Roth, P. C.; Spafford, K.; Tipparaju, V.; Vetter, J. S. The scalable heterogeneous computing (SHOC) benchmark suite, In Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM: 2010; pp 63-74.