International Journal of Grid and Distributed Computing
Vol. 9, No. 9 (2016), pp.403-418
http://dx.doi.org/10.14257/ijgdc.2016.9.9.34

Data Partitioning Strategy of GPU Heterogeneous Clusters Based on Learning

Jianjiang Li[1], Wei Chen[2], *Jin Tian[3], Hongyan Zheng[4], Peng Zhang[5] and Yajun Liu[6]

1-6Department of Computer Science and Technology, University of Science and Technology Beijing, Beijing 100083, P. R. China
1lijianjiang@ustb.edu.cn, 2chenweichina1992@163.com, 3tianjin_tian@163.com, 4hyjy2008@163.com, 5chinazhangpeng1992@163.com, 6liuyajun1992china@163.com

Abstract

With the rapid progress of computational science and computer simulation ability, a lot of properties can be predicted by the powerful ability of parallel computation before the actual research and development. With the development of high performance computer architecture, GPU is more and more widely used in high performance computation field as an emerging architecture, and a growing number of computations use GPU heterogeneous cluster architecture. However, how to partition workload and map to computing resource has always been the focus and difficult point. In the current study of GPU, according to the problems of the computing power provided by each node and the cluster hardware architecture which the application programmers don't understand, some partitioning strategies will result in serious load imbalance problem. Aimed at the complexity brought by the different computing ability of the nodes of GPU clusters, this paper proposes a GPU data partitioning strategy of heterogeneous clusters based on learning. It collects the states of each node in the process of running a program, and then estimates the calculation ability of each node dynamically, so as to guide the data partitioning. Actual testing results show that, this strategy allocates different tasks to nodes based on computing ability to ensure load balancing among nodes, so as to improve the execution performance of CUDA programs on heterogeneous GPU clusters and it laid a solid foundation for efficient computing on heterogeneous GPU clusters.

Keywords*: GPU clusters; Load balancing; Data partitioning; Learning strategy*

1. Introduction

Due to the rapid development of computer technology, the study in simulation of engineering problem by the way of high-performance computer in full swing, it is because: on the one hand, computer simulation has been obtained numerous information which is difficult for actual experiment to obtain; On the other hand, the development of parallel computing for improving reliability validation theory provides a good condition. Finite element method, molecular dynamics method, and artificial neural network are commonly used in computational science, and based on these research methods, a large number of valuable results have obtained.

Computer simulation is a system running on computer according to the research of system models, which is hardly affected by the limits of time and space, and the experimental conditions, so it has great flexibility and randomness. However, with the

ISSN: 2005-4262 IJGDC

development of large scale data, the bottleneck of parallel execution performance is particularly prominent, such as load balancing, communication overhead and partitioning strategy. At present, GPU is widely used in high performance computing, comparing to CPU, GPU's powerful computation ability can greatly improve the computing capacity and reduce the energy consumption of clusters. GPU's powerful ability of parallel computing and CPU's logical processing power supplement each other. Using CPU to complete the program logic processing and using GPU to do parallel computing for large amount of calculation has become an important way of the application of high-performance computing. The CPU+GPU heterogeneous clusters become the mainstream of the high performance computers, and by the way of powerful CPU+GPU heterogeneous cluster computing to serve for computational science research to breakthrough unformed theory systems and the limits of computer technology, it will become the key point to develop high performance computing in future.

However, load balancing is still a problem in the process of running the large of data and programs on GPU clusters. Because both of each node's computing capacity in clusters and the states of computing resource occupied by other users are different, thus the computing power provided by each node is also different. If uniform partitioning strategy is used, it will result in serious load imbalance problem. Due to the application programmers don't understand the cluster hardware architecture and the states of various nodes, they are not able to design adaptable programs according to hardware features of clusters. Therefore, a run-time library need to design for the application programmers, and it can partition data based on the computing power of each node in clusters to avoid unbalanced load.

To this end, this paper proposes a data partitioning strategy based on learning. This strategy estimates the computing power of each node on clusters according to the amount of time and calculation, in order to achieve load balancing among nodes.

2. The GPU Clusters Architecture and Programming Mode

2.1. GPU Clusters Architecture

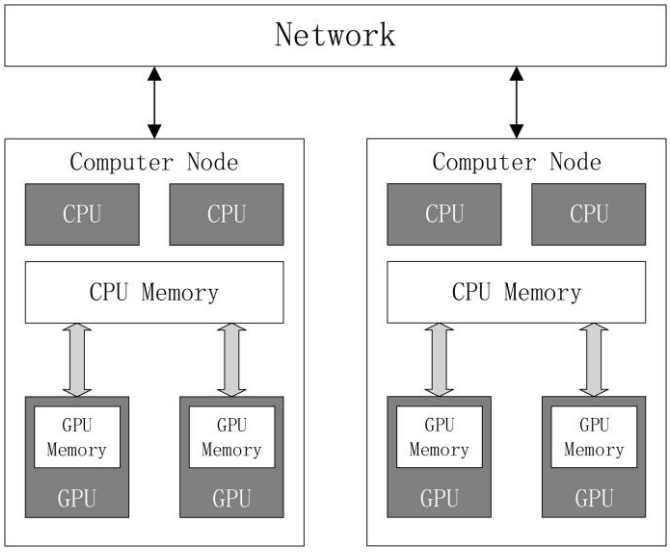


Figure 1. The Architecture of Multi-CPU and Multi-GPU

From the hardware architecture, GPU clusters can generally be divided into two kinds: one is installing multiple GPUs on a single computer, which is generally called the Multi-GPU architecture. Another one is each computing node connected by Ethernet and one GPU or more GPUs installed on each computing node, which is generally referred to as Multi-CPU and Multi-GPU clusters (shown in Figure 1). The GPU clusters used in this paper is Multi-CPU and Multi-GPU cluster. This method can be regarded as an extension of the original CPU clusters. In other words, GPU is installed on each node of the original CPU clusters to speed up the calculation. Within each node, GPUs are connected with main memory through PCI-e bus, and some computing nodes are connected by Ethernet [1].

The architecture of Multi-CPU and Multi-GPU allows the extension between nodes and nodes, namely the calculation ability of each node can be increased by increasing the number of GPUs in a node, and it can also connect more nodes on the cluster to improve the computing power of the cluster [2]. Whether computing power or scalability, this kind of cluster is much stronger than the Multi-GPU cluster. Of course, the complexity of the architecture of this kind of cluster makes programming on it more difficult than on the Multi-GPU cluster.

2.2. The Programming Model on GPU Heterogeneous Clusters

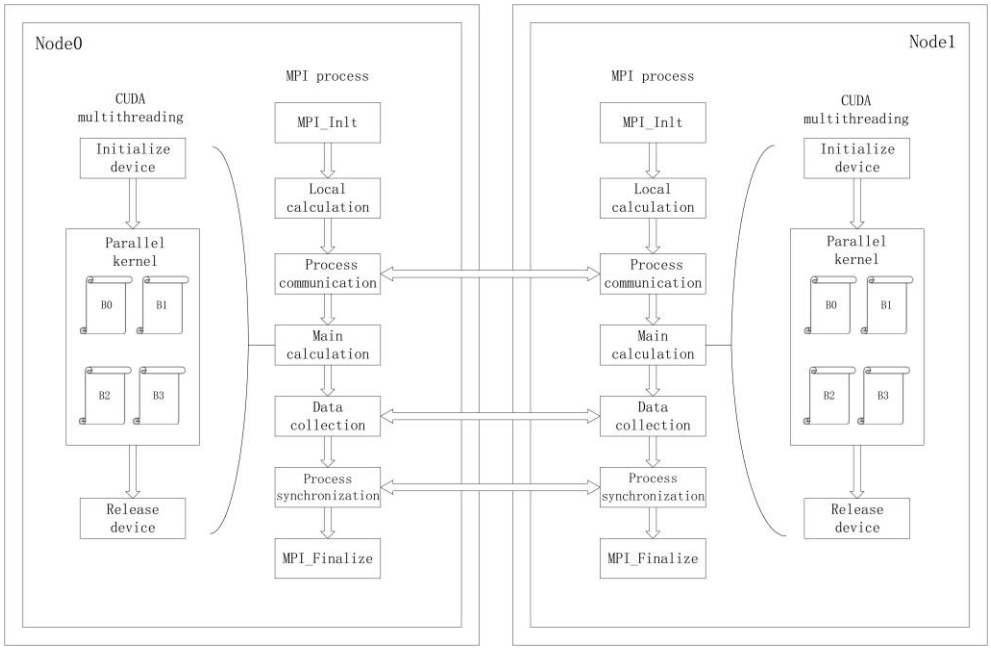


Figure 2. Schematic Diagram of MPI+CUDA Parallel Programming Model

Since the CUDA released by NVIDIA in 2006, MPI+CUDA has become the mainstream of programming models on CPU+GPU heterogeneous clusters. Figure 2 is a simple MPI+CUDA parallel programming model schematic (with two nodes as an example), the main calculation is the need to use the GPU to accelerate parallel tasks [3].

In the MPI+CUDA parallel programming model, data partitioning and data transfer tasks are accomplished by MPI, and the CUDA is used to accomplish the large amount of computation in the program. CUDA can make full use of GPU's high concurrent computing power, which accelerates the main calculation parts of the program and improves the execution performance of the cluster. During the execution of the program,

MPI is initialized, and when using GPUs for large-scale parallel computing, the MPI process initializes the CUDA and GPU devices , and copies the required data to the memory of GPUs to execute, and MPI is used to complete the communication such as data partitioning, data distribution, data transfer, and data collection [4].

2.3. The design of programs on GPU heterogeneous clusters

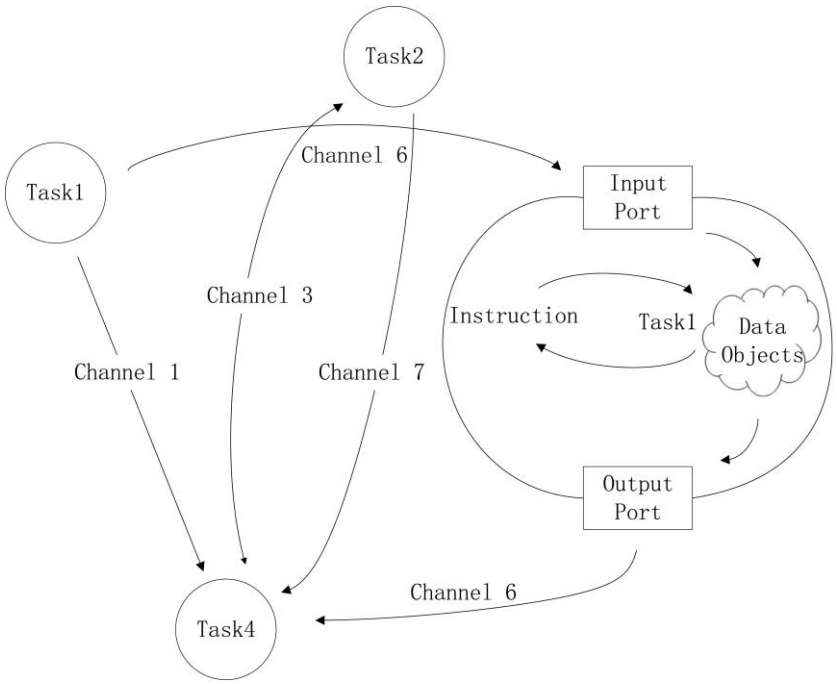


Figure 3. Schematic Diagram of Task Channel Model

It is feasible to use task/channel model [5] to design programs on CPU+GPU clusters. This model expresses parallel computing as a series of tasks and the communication between tasks completed by using channels. Here, the task consists of a program in which each parallel unit needs to execute in a parallel way, the local data of each calculation unit and the I/O of corresponding units. Among them, the local data includes the program instructions of local calculation and the data used to computation. A task can communicate with other tasks through the I/O ports, on the contrary, this task can also receive the data sent by other tasks through the I/O ports. The whole calculation process is expressed as a directed graph (shown in Figure 3). Among them, the task is expressed as a vertex and the communication channel is expressed as a directed edge.

Channel can be regarded as a message queue, and connects the input and output ports of the two tasks. The data appears in the output and the input port in a way of queues. Obviously, if a task want to receive data from one end of the channel, it must wait until the task which is at the other end of the channel sends data. If a task continues to execute only after receiving the data needed, it must wait until the data appears, then continues to execute the next step. At this point, the task receiving data is at the blocking state [6]. Conversely, even the task sent by a certain channel is not accepted, the task sending messages will not be blocked. In a word, in a task/channel model, receiving is a synchronous operation, and sending is an asynchronous operation. For the MPI communication on GPU clusters, the operations of MPI on CPUs is same, namely, the receiving operation of MPI is synchronous and sending is asynchronous. In this model,

local access to private data has great differences with non-local access occurring in the channel, namely the MPI communication on CPUs and the memory I/O operations on GPUs are two asynchronous processes which are mutually independent and have no interference with each other. This model can use the efficiency of accessing memory efficiently on GPUs better. According to the task/channel model, the development of parallel programs on CPU+GPU heterogeneous clusters can be divided into three processes: partition, communications and mapping. And the data partitioning strategy based on learning proposed in this paper is mainly aimed at the data partitioning and mapping process.

3. Data Partitioning Strategy of GPU Heterogeneous Clusters Based on Learning

3.1. The Description of the Learning Process

It is impossible for a single computer to complete complex calculation process, so only relying on powerful parallel computing clusters can scientists "predict" earthquake, weather and so on. In parallel computing, the division is a process of partitioning data and calculations. A good data partitioning method should be able to assign computing tasks to different nodes according to different computing ability of each node. In theory, the optimal partition should be a partitioning scheme in which all computing nodes can complete tasks at the same time. Of course, due to the calculation ability of each node and the situation where other users consume computing resources, as well as that each task is difficult to accurately estimate the amount of calculation, so, the optimal partitioning scheme is difficult to achieve. While a solution can be achieved, which is as close as possible to the optimal partitioning scheme to reduce the idle time of computing nodes and realize the load balance between nodes as much as possible.

In the process of data partitioning, there are two methods: one takes data as the center and others take calculation as the center. Because GPU is good at achieving parallel computing tasks which can process large amount of data, so in the same calculation algorithm, the amount of calculation is proportional to the amount of data. Therefore, to a certain extent, the amount of data can be used to judge the amount of calculation. This paper uses a method which takes data as the center and estimates the amount of calculation through the amount of data [7]. In the process of data partitioning, the tasks with large quantity of data will be assigned to the nodes with powerful calculation ability and the tasks with small quantity of data will be assigned to the nodes with weak calculation ability, which makes almost all tasks be completed at the same time and achieves load balance among nodes as much as possible.

The simplest data partitioning way on a cluster is even partition, namely, each node is assigned with same computational tasks. This partitioning method is simplest and most effective for homogeneous clusters, but for heterogeneous clusters, the computing capacity may vary considerably among nodes, which will lead to great uneven load on the cluster. For the GPU cluster, the computing capacity of different GPUs is also various widely. For example, it is estimated that the calculation ability of Kepler GPU is as roughly 3-4 times as the one of Fermi GPU. Thus, the load imbalance phenomenon brought by this even partition on heterogeneous clusters will be more obvious.

This paper uses a data partitioning strategy based on learning, its core idea is: after each node completes the calculation, it calculates the computing capacity of this node according to the amount of calculation assigned to this node and the time used by this node, and the computing capacity of all nodes will be used as the basis of next data partitioning. Under normal circumstances, after many iterations of the process, the

strategy can estimate the computing capacity of all nodes on GPU heterogeneous clusters more accurately.

Here, this paper takes two GPU nodes as an example to state the process of the data partitioning strategy in detail:

We assume the two nodes as GPU0 and GPU1 respectively and divide tasks when making calculation for the first time, and the time used by the two nodes is t_1^1 and t_1^2 respectively. At this time, we consider that the computing capacity of the two GPUs (named as p^1 and p^2 respectively) is inversely proportional to the time used, namely:

$$\frac{p^1}{p^2} = \frac{t_1^2}{t_1^1} \tag{1}$$

The amount of tasks executed in the first time is named as M_1, and the proportion of M_1 completed by the two nodes is named as d_1^1 and d_1^2 respectively. Where, $d_1^1 = d_1^2 = 1/2$.

During the second data partitioning, the data is distributed in accordance with the circumstances of the first execution time, and the proportion of the amount of tasks distributed to the two nodes is p^1 / p^2. After the two nodes complete the task, the calculation time of the two nodes will be calculated which is named as t_2^1 and t_2^2 respectively, and the total amount of the second task is named as M_2. Where, the proportion of M_2 completed by GPU1 is $d_1^2 = t_1^2 /(t_1^1 + t_1^2)$, the proportion of M_2 completed by GPU2 is $d_2^2 = t_1^1 /(t_1^1 + t_1^2)$. Now we can get:

$$M_2 = (\frac{t_2^1}{t_1^1} \times d_2^1 + \frac{t_2^2}{t_1^2} \times d_2^2) \times M_1 \tag{2}$$

After the second run, the proportion of the latest computing capacity of GPU1 and GPU2 will be calculated depending on the weighted average between the completed amount of the previous two tasks and the execution time. The proportion is:

$$\frac{p^1}{p^2} = \frac{(d_1^1 \times M_1 + d_2^1 \times M_2)/(t_1^1 + t_2^1)}{(d_1^2 \times M_1 + d_2^2 \times M_2)/(t_1^2 + t_2^2)} \tag{3}$$

We regard the proportion as the basis of next data partitioning. As a result, we can analogize the weighted value of the proportion of the computing capacity of the two GPUs after n times calculation, and can obtain the proportion of data partitioning of the $n+1$ times by the execution time of the previous \$n\$ times calculation:

$$\frac{p^1}{p^2} = \frac{\sum_{i=1}^{n} d_i^1 \times M_i / \sum_{i=1}^{n} t_i^1}{\sum_{i=1}^{n} d_i^2 \times M_i / \sum_{i=1}^{n} t_i^2} \tag{4}$$

Where, d represents the every calculated proportion of the tasks obtained by the GPUs, M represents the calculation amount of each calculation which can be compared with M_1 and t represents the execution time. Additionally, the superscript stands for the serial number of nodes and the subscript stands the serial number of the calculation.

For the whole cluster, the computing capacity of the i th node on this cluster p_i can be represented as:

$$p_i = \frac{\sum_{j=1}^{n-1}(d_j \times M_j)}{\sum_{j=1}^{n-1} t_j} \tag{5}$$

After the completion of each task, the computing capacity of each node will be calculated according to the equation (5), and it will be regarded as the basis of next data partitioning. In equation (5), d represents the proportion of the tasks obtained by the GPUs and t represents the time of each calculation. Therefore, the error of the computing capacity of nodes calculated by equation (5) mainly comes from the error of M [8] which represents the number of tasks during each execution. If we use T to express the actual amount of each executed task and T_j to express the actual amount of the jth executed task, thus, in theory, the computing capacity of the ith node should be represented as:

$$p_i = \frac{\sum_{j=1}^{n-1}(d_j \times T_j)}{\sum_{j=1}^{n-1} t_j} \tag{6}$$

The difference between M and T is the error of every time estimating the time of the computing capacity of nodes.

In the process of each task execution, the execution can't be completely proportional to the amount of data (for example, in the process of matrix partitioning, the sparse and dense degree of partitioned matrix could not be absolutely same. So, the amount of computing tasks about matrix which each node obtains through partition can't be completely proportional to the amount of data about partitioned matrix obtained by data partitioning). As a result, the amount of computation which is judged according to the amount of data M exists error unavoidably. But each error is produced by randomness, so after tasks have been executed for infinite times, the total number of tasks assigned to each node and the one of data are same, that is:

$$\sum_{j=1}^{\infty}(d_j \times M_j) = \sum_{j=1}^{\infty}(d_j \times T_j) \tag{7}$$

So,

$$\frac{\sum_{j=1}^{\infty}(d_j \times M_j)}{\sum_{j=1}^{\infty} t_j} = \frac{\sum_{j=1}^{\infty}(d_j \times T_j)}{\sum_{j=1}^{\infty} t_j} \tag{8}$$

Namely, as the iteration times of this process becomes more, the error of the assessment for the computing capacity of all nodes becomes smaller. After tasks have been executed for infinite times, the assessment for the computing capacity of all nodes should converge to the actual computing capacity.

The data partitioning algorithm based on the above strategy is an ideal way for single user clusters, but for multi-user clusters, there are still some shortcomings. When a cluster is in a state where multi-user and multitask are concurrently executed, the computing tasks of a user cannot monopolize the computing resources of the cluster. So, at this point, because the computing resources of some nodes may be taken up much by the computing tasks of other users, if data is still partitioned based on the computing capacity of the

cluster, these nodes will spend more time in completing tasks divided at this time, which results in uneven load among nodes.

As users occupy the cluster continuously, namely in the period of executing the last task, if other users has occupied some computing nodes of the cluster, it is likely that the nodes will still be occupied when executing this task. Therefore, for this kind of problem, this paper assigns larger weighted value to the computing capacity of nodes obtained at the latest computing tasks. That is, we think that when the cluster completes the last computing task, other users occupy the same computing resources as they do at this computation. As a result, when assessing the weighted value of the computing capacity of nodes, assigning lager weight to the last calculation results will eliminate the effects which the computing tasks of other users under the situation of multi-user have on the balance of data partitioning on the cluster.

Thus, after finishing calculation for *n* times, we can get that the computing capacity of the *i*th node which is:

$$p_i = (1-\theta) \times \frac{\sum_{j=1}^{n-1} (d_j \times M_j)}{\sum_{j=1}^{n-1} t_j} + \theta \times \frac{d_n \times M_n}{t_n} \tag{9}$$

Where, θ is the weighted value of the last computing results.

For this kind of data partitioning strategy based on learning, if the calculation whose type is same is consecutively executed (for example, all of them are the matrix multiplication), the proportion of data partitioning can converge to the computing capacity of nodes by a very small number times of iterations, thus it can provide guidance for later data partitioning. While for the calculation whose type is different (for example, the matrix multiplication of fixed-point numbers and floating-point numbers), the computing core of each node is different, and different computing cores may make special optimization for different calculation, so the algorithm in this strategy needs more times of iterations to converge, but after making multiple computations to get the weighted average, we can also obtain the data partitioning conditions which conforms comparatively to the computing capacity of nodes.

In the iterative process of the strategy, there are some special operations (such as calculating the multiplication of sparse matrix on some nodes) which could damage the weighted results [9]. As the computing amount of the sparse matrix is far less than the one of the normal matrix, so the computing results will have adverse effects on the results of the overall weight. But because the iterative process will be executed for multiple times, the adverse effects will be diluted in the process of executing the algorithm in this strategy for multiple times. Therefore, this paper considers that a small number of special operations will not cause significant impact on data partitioning.

3.2. The Implementation of Data Partitioning Strategy

Algorithm 1 The data partition algorithm of multi-CPU and multi-GPU clusters based on learning

Input:

θ: The weighted value

Output:

GET-CLUSTER-OPTIMIZE-DATA-SPLIT-ARGUMENT

```
let p[m] be new array      //p stores the computing power of each node
alltime = 0
for h = 1 to m
    p[h]=0
for i = 1 to m
    //Calculate the computing power of each node according to the formula (9)
    quota = 0
    for j = 1 to n-1
        quota = d[i,j]*M[j]
        alltime = alltime + t[i,j]
    //In the actual process, θ=0.6 is ok
    p[i]=(1-θ)*quota/alltime+θ*d[i,n]*M[n]/t[i,n]
pall = 0
for k = 1 to m
    pall = pall + p[k]
for l = 1 to m
    d[n+1,l] = p[l]/pall
```

According to the previous section, the learning process can calculate the computing power of each node by the formula (9), then divide the data according to the computing power of each node. The workload obtained by each node is:

$$d_i = \frac{p_i}{\sum_{i=1}^{n} p_i} \tag{10}$$

In the implement of the algorithm in this strategy, two-dimensional matrix $d[m,n]$ represents workload on each node by data partitioning, two-dimensional matrix $t[m,n]$ represents the execution time on each node, one-dimensional matrix $M[n]$ represents the amount of workload, and one-dimensional matrix $p[m]$ represents the computing power of each node. The data partition algorithm of multi-CPU and multi-GPU clusters based on learning is shown in Algorithm 1.

After each operation is completed, the algorithm records the information such as the computation time on each node and the proportion of allocation tasks, and then calculates the computing power of each node based on the above information. When performing the task again, tasks will be partitioned according to the computing power of each node [10]. These calculations will be long stored on the hard disk. When the data will be partitioned, at first, the previous running condition of each node needs to be read from the hard disk, then the data will be partitioned based on the computing power of each node. The whole process of data partitioning is shown in Figure 4.

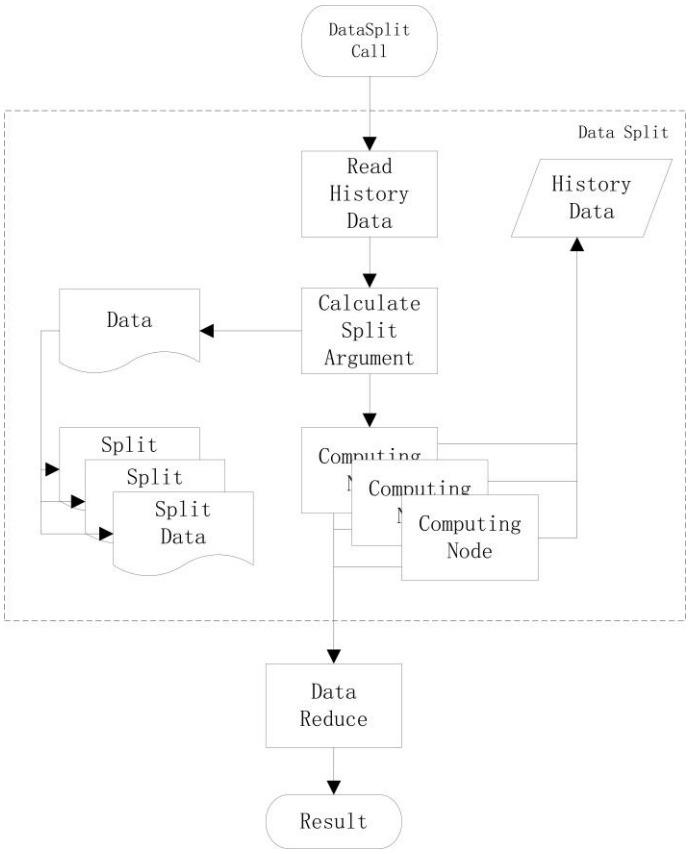


Figure 4. The Flow Chart of Data Partitioning

4. Test and Result Analysis

The algorithm of data partitioning based on learning proposed in this paper and corresponding runtime library will be tested through the matrix multiplication, and the test results will be analyzed.

4.1. The Implementation of Data Partitioning Strategy

Table 1. The Hardware Environment of Test Platform

Hardware Platform			
Node0	Host	processor	2 × Intel Duo 2.8GHz/3072KCache
		memory	8G
		hard disk	500G
		NIC	2 × Intel Ether Express/1000
		interface	Two 16x PCI-E slot
	Device	GPU model	2× NVIDIA GTX 295 Graphics
		core hardware	4×GT200
		video memory	2 × 1G
		SP	4 × 240
		MP	4 × 240
Node1	Host	processor	2 × Intel Duo 2.2GHz/2048K Cache

		memory	1G
		hard disk	250G
		NIC	1 x Intel Ether Express/1000
		interface	One 16x PCI-E slot
	Device	GPU model	1 × NVIDIA GeForce 8800 Graphics
		core hardware	1 × G80
		video memory	1 × 332M
		SP	1 × 96
		MP	1 × 12

Table 2. The Software Environment of Test Platform

Software Environment	
Operating System	Red Hat Enterprise Linux Server release 5.1 64bit
NVIDIA GPU Drive	NVIDIA-Linux-x86 64-185.18.14
NVIDIA toolkit	CUDA toolkit 3.0 linux 64
NVIDIA SDK	CUDA sdk 3.0 linux
Complier	GUN GCC 4.2.1

The test platform is a SMP cluster and the GPU server nodes will be used in this paper. The GPU server contains two GT200 series hardware cores, both for NVIDIA GTX 295 graphics and a dual-core Intel E7400CPU. Hardware test platform also includes a PC server equipped with GeForce8800GTS graphics card which belongs to NVIDIA G80 series, and a dual-core Intel E4500 CPU. The specific hardware platform and software environment are shown in Table 1 and 2.

4.2. The Implementation of Data Partitioning Strategy

This paper uses the two-dimensional matrix multiplication to test the various modules and their performance in the runtime library implemented in this paper. In order to test the modules in runtime library, this paper adopts the standard two-dimensional matrix multiplication, but doesn't use the commonly used optimization algorithms (such as: Strassen algorithm, *etc.*), however they use the most primitive one-way block algorithm. Using this way is to observe intuitively the way of data partitioning and guarantee optimization under the condition of load balancing. The classical methods of matrix multiplication is shown in Figure 5. Matrix A is divided into three pieces which respectively are computed by GPUs on the two nodes. After the calculation is finished, the three parts of matrix C will be compounded to generate a complete matrix C.

In the process of test, this paper respectively uses four different matrices with the size of 512*512, 1024*1024, 1536*1536, 2048*2048 to calculate, and the execution time is shown in Figure 6. When calling the data partitioning algorithm proposed in this paper and its runtime library, as well as the first task completed, the optimization module will estimate the computing power of each node and data partitioning according to the results of the first round execution. The actual test results show that the algorithm of data partitioning proposed in this paper and corresponding runtime library improves significantly the execution performance of the whole cluster.

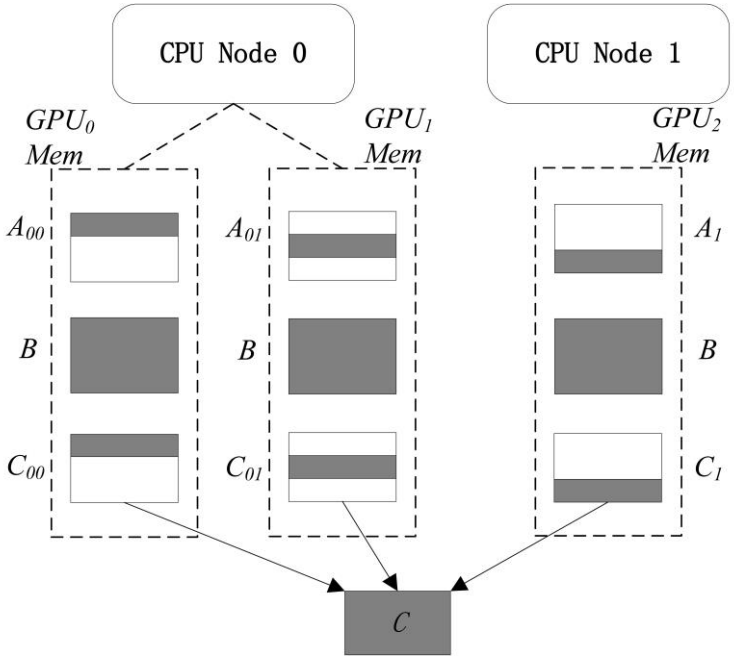


Figure 5. Schematic Diagram of the Division of Matrix Multiplication

GTX295 and GeForce8800GTS used in experiments belong to two different generations of graphics card, and the computing power of GTX295 obviously stronger than GeForce8800GTS. The partitioning results of the matrix with the size of 2048*2048 are shown in Figure 7.

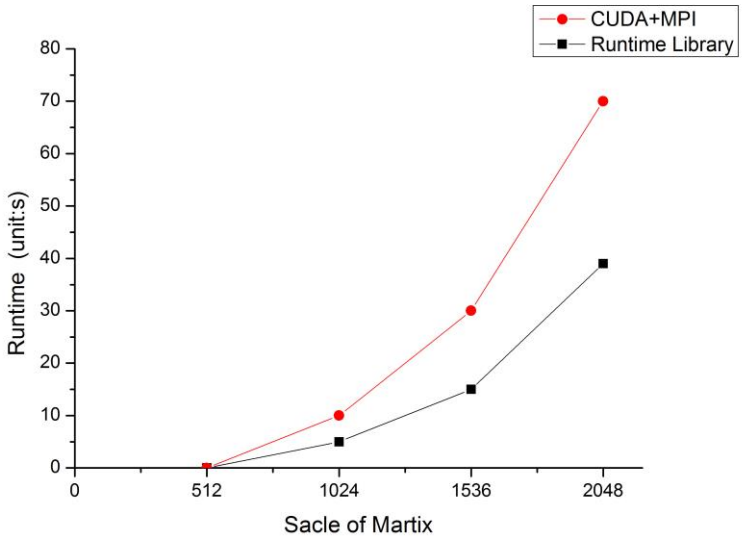


Figure 6. The Architecture of Multi-CPU and Multi-GPU

Figure 7 shows that, after 3 times calculation, the algorithm of data partitioning proposed in this paper can divide roughly the data according to the computing power of each GPU node.

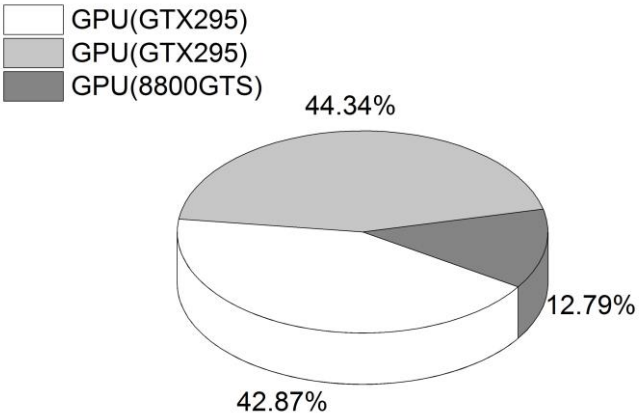


Figure 7. The Partitioning Results of the Matrix with the Size of 2048*2048

5. Related Work

In order to use the computing resources reasonably and compute efficiently, there are a lot of research on CPU/GPU heterogeneous clusters and load balancing. In paper [11], they propose the idea of query-aware data stream partitioning and it allows us to climb the performance of streaming queries in close to linear fashion. The stream partitioning mechanism includes two main components. The first component is a query analysis framework for determining the optimal partitioning for a given set of queries. The second component is a partition-aware distributed query optimizer that transforms a plan. They also demonstrate their partitioning approach by running sets of streaming queries of various complexities on a small cluster of processing nodes that use high-rate network data streams. They put forth a significantly new approach [12] to enhance the timing predictability of multicore architectures aimed at task migration in embedded environments. This paper regards task migration as a key contributor for unpredictability in determining WCET bounds of real-time tasks on multicore architectures. This paper proposes two schemes of push-assisted cache-to-cache migration in multicores as a means to diminish the dilation introduced by the target warm-up overhead. The first scheme, a hardware scheme replicating cache context of the task onto the target L2 cache, reduces dilation in execution time to less than a percent for the majority of simulated tasks. Our second scheme extends software support by a hardware scheme so that developers may specify address ranges associated with the task. In paper [13], the authors discuss a method that can dispatch the appropriate tasks to each node to achieve load balancing. They assume that each node has an initial capability of hyper computing, according to number of completed tasks in each cycle; this capability of each node will be updated dynamically. They will also show that how the tasks resend when some nodes disconnect to improve the system's reliability. In our experiments, the load of each computing node can be balanced within a few minutes, and if some nodes disconnect, the computing tasks can be completed normally. They present dJay [14], a utility-maximizing cloud gaming server that dynamically tunes client GPU rendering loads in order to ensure all clients get satisfactory frame rate, and provide the best possible graphics quality across clients. Their results show that when compared to a static configuration, they can respond much better to peaks and troughs, achieving up to four times the multi-tenant density on a single server while offering clients the best possible graphics quality. In paper [15], the authors present a high performance multi-GPU merge sort algorithm that solves the problem of sorting data distributed across several GPUs. The merge sort algorithm first sorts the data

on each GPU using an existing single-GPU sorting algorithm. Then, a series of merge steps produce a globally sorted array distributed across all the GPUs in the system. The merge phase is enabled by a novel pivot selection algorithm that ensures that merge steps always distribute data evenly among all GPUs. The authors also present the implementation of their sorting algorithm in CUDA, as well as a novel inter-GPU communication technique that enables this pivot selection algorithm. A traditional load balancer is very expensive, the policy sets of a load balancer need to be set in advance, its flexibility is very low, and it cannot deal with emergency situations very well. It also requires a specialized administrator to maintain and cannot make flexible strategies according to its own actual network conditions. In paper [16], they put forward the design and implementation of server cluster dynamic load balancing in virtualization environment based on OpenFlow, the architecture is not only low costs but also can provide flexible programmable modules which is used to achieve the policy sets that appropriate for your network in the Controller. With the flexible configuration ability of this architecture, internet application can achieve real-time monitor condition of loading and obtain the corresponding resources in a timely manner. In paper [17], the authors propose a framework for distributed thermal management for many-core systems where balanced thermal profile can be achieved by proactive task migration among neighboring cores. The framework has a low cost agent residing in each core that observes the local load and temperature and communicates with its nearest neighbor for task migration/exchange. By choosing only those migration requests that will result balanced load without generating thermal emergency, the proposed framework maintains load balance across the system and avoids unnecessary migration. Experimental results show that, compared with existing proactive task migration technique, our approach generates less hotspots and smoother thermal gradient with less migration overhead and higher processing throughput. High performance computing (HPC) has recently emerged as a promising class of computational science and have many potential applications in other sectors as well.

6. Conclusion

Using computer to deal with complex data has become an important tool for scientists. In order to improve the reliability of scientific research, save time, and solve the data partitioning problem causing by the GPU heterogeneous clusters, this paper proposes a data partitioning strategy based on learning. The strategy estimates the computing power of various nodes by analyzing workload and corresponding execution time of the last round calculation, and partitions data according to the computing power of each node in order to implement load balancing of all nodes. The test results show that 908, 878, 262 are the divided matrix dimensions respectively on GPU0, GPU1, GPU2. And the computing power of GTX295 obviously stronger than 8800 GTS, so data partitioning module is divided based on GPU computing ability to carry out the task. At last, the strategy of data partitioning proposed in this paper and corresponding runtime library can improve load imbalance and enhances significantly the execution performance of heterogeneous GPU clusters.

Acknowledgments

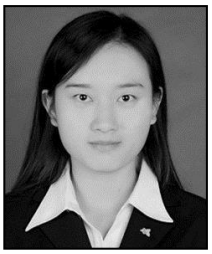
This work was supported in part by the National High Technology Research and Development Program of China (No.2015AA01A303), Beijing Key Subject Development Project of China (XK10080537).

References

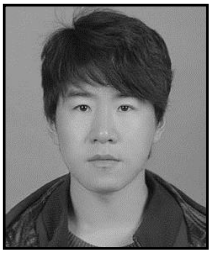
[1] V. T. Ravi, M. Becchi, G. Agrawal, S. Chakradhar, "Valuepack: value-based scheduling framework for cpu-gpu clusters", in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society Press, 2012, p. 53.
[2] N. Maruyama, T. Nomura, K. Sato, S. Matsuoka, "Physis: an implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers", in: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, IEEE, 2011, pp. 1–12.
[3] Cuda c programming guide, http://www.nvidia.com (2013).
[4] S. Yamagiwa, L. Sousa, "Caravelampi: Message passing interface for parallel gpu-based applications", in: Parallel and Distributed Computing, 2009. ISPDC'09. Eighth International Symposium on, IEEE, 2009, pp. 161–168.
[5] D. Chen, W. Chen, W. Zheng, "Cuda-zero: a framework for porting shared memory gpu applications to multi-gpus", Science China Information Sciences 55 (3) (2012) 663–676.
[6] H. Huo, C. Sheng, X. Hu, B. Wu, "An energy efficient task scheduling scheme for heterogeneous gpu-enhanced clusters", in: Systems and Informatics (ICSAI), 2012 International Conference on, IEEE, 2012, pp.623–627.
[7] L. Wang, Y. Wu, W. Jia, W. Gao, X. Chi, L.-W. Wang, "Large scale plane wave pseudopotential density functional theory calculations on gpu clusters", in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2011, p. 71.
[8] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, S. Matsuoka, "Statistical power modeling of gpu kernels using performance counters", in: Green Computing Conference, 2010 International, IEEE, 2010, pp.115–122.
[9] S. Hong, H. Kim, "An integrated gpu power and performance model", in: ACM SIGARCH Computer Architecture News, Vol. 38, ACM, 2010,pp. 280–289.
[10] Y. Jiao, H. Lin, P. Balaji, W.-c. Feng, "Power and performance characterization of computational kernels on the gpu", in: Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conferenceon & Int'l Conference on Cyber, Physical and Social Computing (CP-SCom), IEEE, 2010, pp. 221–228.
[11] T. Johnson, M. S. Muthukrishnan, V. Shkapenyuk, O. Spatscheck, "Query-aware partitioning for monitoring massive network data streams", in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM, 2008, pp. 1135–1146.
[12] A. Sarkar, F. Mueller, H. Ramaprasad, S. Mohan, "Push-assisted migration of real-time tasks in multi-core processors", in: ACM Sigplan Notices, Vol. 44, ACM, 2009, pp. 80–89.
[13] Y. Ou, H. Chen, L. Lai, "A dynamic load balance on gpu cluster for fork-join search", in: Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, IEEE, 2011, pp. 592–596.
[14] S. Grizan, D. Chu, A. Wolman, R. Wattenhofer, "djay: enabling high-density multi-tenancy for cloud gaming servers with dynamic cost-benefit gpu load balancing", in: Proceedings of the Sixth ACM Symposium on Cloud Computing, ACM, 2015, pp. 58–70.
[15] I. Tanasic, L. Vilanova, M. Jordà, J. Cabezas, I. Gelado, N. Navarro, W.-m. Hwu, "Comparison based sorting for systems with multiple gpus", in: Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units, ACM, 2013, pp. 1–11.
[16] W. Chen, H. Li, Q. Ma, Z. Shang, "Design and implementation of server cluster dynamic load balancing in virtualization environment based on openflow", in: Proceedings of The Ninth International Conference on Future Internet Technologies, ACM, 2014, p. 9.
[17] Y. Ge, P. Malani, Q. Qiu, "Distributed task migration for thermal management in many-core systems", in: Proceedings of the 47th Design Automation Conference, ACM, 2010, pp. 579–584.

Authors

Jianjiang Li, is currently an associate professor at University of Science and Technology Beijing, China. He received his PhD degree in computer science from Tsinghua University in 2005. He was a visiting scholar at Temple University from Jan. 2014 to Jan. 2015. His current research interests include parallel computing, cloud computing and parallel compilation.



Wei Chen, is currently an master degree candidate at University of Science and Technology Beijing, China. She received her B. S. Degree in Network engineering from North China Institute of Aerospace Engineering in 2014. Her current research interests include parallel computing, cloud computing and parallel compilation.



Peng Zhang, is currently a master degree candidate in University of Science and Technology Beijing, China. He received his B. S. Degree in information management and information system from Tangshan Normal University in 2015. His current research interests include parallel computing, cloud computing and parallel compilation.

Yajun Liu, is currently an master degree candidate at University of Science and Technology Beijing, China. She received her B. S. Degree in computer science and technology from Shanxi University in 2015. Her current research interests include parallel computing, cloud computing and parallel compilation.

