A GPU Heterogeneous Cluster Scheduling Model for Preventing Temperature Heat Island

Yun-Peng CAO[1,2,a] and Hai-Feng WANG[1,2]

[1]*School of Information Science and Engineering, Linyi University, Linyi Shandong, China 276005*
[2]*Institute of Linyi University of Shandong Provincial Key Laboratory of Network based Intelligent Computing, Linyi Shandong, China 276005*

Abstract. With the development of GPU general-purpose computing, GPU heterogeneous cluster has become a widely used parallel data processing solution in modern data center. Temperature management and controlling has become a new research hotspot in big data continuous computing. Temperature heat island in cluster has important influence on computing reliability and energy efficiency. In order to prevent the occurrence of GPU cluster temperature heat island, a big data task scheduling model for preventing temperature heat island was proposed. In this model, temperature, reliability and computing performance are taken into account to reduce node performance difference and improve throughput per unit time in cluster. Temperature heat islands caused by slow nodes are prevented by optimizing scheduling. The experimental results show that the proposed scheme can control node temperature and prevent the occurrence of temperature heat island under the premise of guaranteeing computing performance and reliability.

1 Introduction

After GPU (Graphic Processing Unit) was proposed by NVIDIA company and its birth, it has been developing rapidly beyond the speed of Moore's Law, its computing capability has been rising continuously. At SIGGRAPH conference in 2003, GPGPU(General-purpose computing on graphics processing units) was introduced. GPUs gradually shifted from dedicated parallel processors consisting of fixed functional units to architectures with primary general-purpose computing resources and secondary fixed functional units. GPU is composed of a large number of parallel processing units and memory control units, its processing power and memory bandwidth has obvious advantages compared with CPU. However, GPU cannot completely replace CPU, a lot of operating systems, softwares and codes cannot run on GPU. GPU general-purpose computing usually uses CPU/GPU heterogeneous mode, CPU executes complex logic and transactions and other tasks unsuitable for parallel processing, GPU implements compute-intensive large-scale data parallel computing tasks. With its high performance, low energy consumption and other advantages, CPU/GPU hybrid architecture has been widely used in graphics and image processing, video encoding and decoding, matrix computing and simulation, medical industry application, life science research, high-performance computing, signal processing, database and data mining and many other fields. With technology advances and breakthroughs, GPU is playing an important role currently in

[a] Corresponding author: lyucyp@163.com

large-scale parallel computing. With the rapid increase of problem scales of various application fields, single GPU’s computing capability has become insufficient, so multi-GPU and GPU cluster general-purpose computing has become a new research hotspot. As an important approach of high-performance computing, GPU clusters have such advantages as low cost, high performance and low energy consumption for compute-intensive applications. In constructing GPU clusters, CPU and GPU cooperate with each other, participate in data processing, and form GPU heterogeneous cluster. GPU heterogeneous cluster can make full use of hardware resources, improve processing speed and throughput. It has become an important means of big data processing.

Processing big data, especially real-time big data stream needs cluster’s continuous computing and processing, and it will inevitably require computer’s high-load and continuous work, so the temperature of CPU, GPU and other components will continue to rise. On one hand, computing energy consumption increases, on the other hand, fans and air conditioners are needed for reducing temperature, thereby increasing cooling energy consumption. When temperature rises to a certain extent, the temperature of one or some nodes will be too high. The node with too high temperature is known as temperature heat island. The occurrence of temperature heat island will reduce computing reliability, ranging from result error to system’s paralysis and halt. Once errors occur in computing results, recomputing is needed, resulting in time and resource waste, increasing processing costs. In this case, we must reasonably design cluster task scheduling scheme to minimize cluster overall run time, control temperature to appropriate range, prevent individual node from running so long that leading to over high temperature and forming temperature heat island, to ensure reliable computing results, reduce energy consumption as much as possible and achieve green computing.

This paper studied the task scheduling on GPU heterogeneous cluster, and proposed a task scheduling scheme of preventing temperature heat island. The scheme’s main features and advantages are:

(1) strong robustness. The structure of GPU heterogeneous cluster is complex, each node’s configuration is different, and the node is often changed and adjusted. This task scheduling scheme can sense and adapt to this complicated and changeable situation.

(2) high processing performance. A task is divided into some sub-tasks, and then they are scheduled to multiple nodes for parallel processing. The main problem is determining the mode of division and treatment. The concept of computing scale threshold and asymmetric partitioning method are proposed in order to adapt to the diversity and heterogeneity of node configuration, improve the parallelism and shorten the whole running time of cluster. This not only prevents temperature heat island from occurring because of individual node’s overlong running time, but also improves processing performance.

2 Related researches

With the wide application of GPU heterogeneous cluster, its task scheduling, temperature and heat management and energy consumption optimization has become a research hotspot. Many scholars have put forward various scheduling schemes and methods to solve the problem of energy consumption and reliability. This has played a positive role in reducing cluster energy consumption and ensuring the reliability of computing results.

In [1] a dynamic task partition method was proposed. It divides parallel computing tasks according to execution speed to achieve best overall system performance. In [2] a multi-GPU self-adaptive load balancing method was proposed. GPU can self-adaptively select tasks to execute according to local free-busy state by establishing task queue model between CPU and GPU. In [3] a load balancing strategy that combines task partitioning and stealing was proposed. It takes into account task affinity and processor diversity to direct task scheduling between CPU and GPU. In [4] feedback controlling was combined with mixed integer programming, and the energy consumption controlling model of Web server cluster was constructed. In [5] model predictive controlling strategy was introduced from global perspective. The energy consumption state is changed by adjusting computing frequency and changing active stream multiprocessor. The feedback controlling and rolling optimization mechanism are used to predict future controlling to reduce redundant energy consumption. In [6] the energy loss at idle state is reduced by a specific node selection strategy. CPU resource utilization is improved by task type division, combination distribution and DVFS.

The above researches mainly focus on cluster task scheduling, changing CPU/GPU core voltage, frequency, hardware-based statistics, and so on to design cluster energy consumption model, study task scheduling algorithm and achieve energy-saving purpose, but do not consider temperature much. In GPU cluster computing, especially continuous computing, temperature has obvious relationship with energy consumption and reliability. When temperature is too high, energy consumption increases, reliability declines, and the probability of result error increases. Therefore, temperature should be controlled in a reasonable range to minimize energy consumption under the premise of ensuring reliability. The task scheduling scheme proposed in this paper distributes tasks reasonably among computing nodes to prevent the occurrence of temperature heat island and ensure the correctness of computing results.

3 Task scheduling model

In GPU heterogeneous cluster, CPU and GPU all participate in data processing. They are regarded as computing units uniformly when distributing tasks. The computers in cluster are classified as controlling nodes and computing nodes. The controlling node can be simultaneously used as a computing node. All tasks form a queue. Each task is decomposed into several sub-tasks to form sub-task queue. The controlling node runs the main scheduling process, Scheduler. Each computing node has a scheduling agent process, Agent. Scheduler and Agent cooperate to finish task scheduling. The architecture is shown in Figure 1.

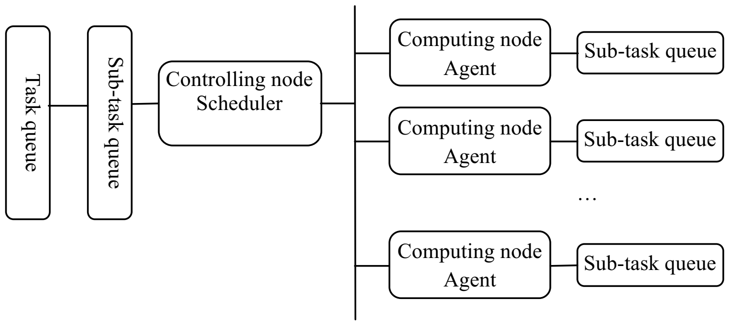

Figure 1. Task scheduling architecture

4 Task scheduling algorithm and strategy

Scheduler algorithm is as follows:

```
Algorithm 1 Controlling node Scheduler scheduling algorithm
1. Obtain a task from task queue
2. Obtain the hardware configuration and running status information of each computing node
3. Determine the number of computing nodes participating in parallel processing
4. Divide task into sub-task queue and assign sub-tasks to corresponding computing node
5. Wait for the results of each sub-task
6. Modify the status of corresponding sub-tasks and the associated tasks in queue.
7. Reschedule sub-tasks that timed out or requested to transfer, modify corresponding status
8. Go to 1
```

For each computing node, the sub-tasks that controlling node dispatches to it form a queue. The scheduling algorithm of Agent on computing node is as follows:

```
Algorithm 2 Computing node Agent scheduling algorithm
1. Obtain a sub-task from the sub-task queue of local node
2. Assign the sub-task to local node for processing
3. Wait for the result to be returned from local node
4. Report results to controlling node (completion, timeout, or requesting transfer)
5. Go to 1
```

4.1 Acquiring hardware configuration information

Scheduler first obtains the hardware configuration information of each node in cluster. The information can be manually created in advance and saved in file. When cluster is started, Scheduler loads cluster hardware configuration information file. It polls each computing node, Agent responds to the poll and reports hardware change information to Scheduler. Or, Agent reports hardware change information to Scheduler actively. Then Scheduler modifies cluster's hardware configuration information. In this way, controlling node can grasp the latest changes in cluster hardware configuration, avoiding unnecessary acquisition and reporting of hardware configuration information, thus adapting to actual hardware configuration changes and reducing network communication overhead.

4.2 Scheduling strategy

Computing scale is used to measure task size. Computing scale is the number of instructions to be executed or the amount of data to be processed to complete the task. A task contains parallelizable and non-parallelizable part. Suppose the computing scale of a task is T, $T=T_s+T_p$, T_s is the computing scale of non-parallelizable part, and T_p is the computing scale of parallelizable part. Let T_t be the critical value of the computing scale of parallelizable part, then task scheduling strategy is as follows:

(1) $0\le T_p<T_t$, the task does not have parallelizable part or has relatively smaller parallelizable part, it cannot or has no need to be divided into small sub-tasks, the computing unit with strongest computing capability is selected directly from idle computing units to process the task.

(2) $T_p\ge T_t$, the task has parallelizable part and it reaches a certain scale. The parallelizable part of task is divided into smaller sub-tasks, many computing units with strongest computing capability are selected from idle processing units to process them.

4.2.1 Determining T_t and the number of computing units

The processing capability is assumed to be C_s when task is processed separately by a single computing unit. Without loss of generality, assuming that when parallel processing, the number of computing units participating in processing is n, their processing capability is all C_p. In order to obtain better performance, then:

$$\frac{T_t}{C_s} \ge \frac{T_t}{nC_p} + Q \tag{1}$$

Where Q is the additional time overhead required for parallel processing, including parallel computing preparation, result merging, synchronization, network transmission, and so on. At the same time in order to ensure high processing efficiency, then:

$$\frac{T_t}{nC_p} \ge Q \tag{2}$$

Solving the inequality group consisting of above two inequalities will get:

$$T_t \geq \max\{nC_pQ, \frac{nC_pC_sQ}{nC_p - C_s}\} \tag{3}$$

The value of Q can be determined experimentally or by accumulating historical empirical data. C_p can be taken as the average of the current computing capability of all computing units, and C_s is the average of the current computing capability of all CPUs in cluster. Let $T_m = \max\{mC_pQ, \frac{mC_pC_sQ}{mC_p - C_s}\}$, $m=1,2,...,N_{idle}$, N_{idle} is the number of all idle computing units in current cluster. In order to increase the parallelization degree of task processing, change from N_{idle} in descending manner until the first number k which lets $T_p \geq T_k$ is found, then k is the number of units involved in parallel processing, the algorithm to determine it is as follows:

Algorithm 3 Determining the number of parallel processing units

```
1. get $N_{idle}$
2. i←$N_{idle}$ $k$←1
3. if $i \leq 1$ goto 7
4. $T_i \leftarrow \max\{iC_pQ, \frac{iC_pC_sQ}{iC_p - C_s}\}$
5. if $T_p \geq T_i$ then $k \leftarrow i$ goto 7
6. $i \leftarrow i$-1 goto 3
7. end
```

If $k<2$ or qualified k value cannot be found, the task is handled by one CPU and not scheduled in parallel manner.

4.2.2 Partitioning parallel part

Assuming that the current computing capability of k computing units involved in parallel computing is $C_1, C_2, ..., C_k$, the scale of sub-tasks assigned to each processing unit is $T_1, T_2, ..., T_k$, then the time to complete the task is:

$$t = \max\{\frac{T_1}{C_1}, \frac{T_2}{C_2}, ..., \frac{T_k}{C_k}\} \tag{4}$$

Where $T_1+T_2+...+T_k=T_p$. It can be proven that when $T_i = \frac{C_iT_p}{C}$ $(i=1,2,...,k)$, t is minimum and $t = \frac{T_p}{C}$,where $C=C_1+C_2+...+C_k$. Therefore, the proportion of allocated task to total task scale being equal to the ratio of the current computing capability of the computing unit to the sum of the current computing capabilities of all computing units participating in parallel processing, can effectively reduce overall processing time, balance load, and avoid the case that some units are idle and some units run for long time and cause temperature heat islands to occur.

4.3 Estimating current computing capability

Current computing capability is related to its own hardware configuration and hardware's current state of utilization. For computing units with same configuration, the busier ones have stronger current computing capability than the idle ones. By referencing [7] and improving, the current computing capability is estimated. For any computing node N_i, consider its five hardware configuration parameters: CPU frequency *rate_cpu*$_i$, memory size *mem*$_i$, cache size *cache*$_i$, GPU frequency *rate_gpu*$_i$, GPU memory size *mem_gpu*$_i$ and five state parameters: CPU utilization *utlz_cpu*$_i$, memory utilization *utlz_mem*$_i$, cache utilization *utlz_cache*$_i$, GPU utilization *utlz_gpu*$_i$, GPU memory utilization *utlz_gpumem*$_i$. The current computing capability of node N_i is:

$$C_i = k_1 Q_1 + k_2 Q_2 + k_3 Q_3 + k_4 Q_4 + k_5 Q_5 \tag{5}$$

k_1, k_2, k_3, k_4 and k_5 represents the level proportion weight of influence on node current computing capability of CPU, memory, Cache, GPU and GPU memory respectively. Their sum is 1. Q_1-Q_5 respectively denotes CPU current capability, memory current capability, cache current capability, GPU current capability and GPU memory current capability after normalization of node N_i. Q_1 is calculated as:

$$Q_1 = \frac{rate\_cpu_i \times (1 - utlz\_cpu_i)}{(\sum_{j=1}^{N} rate\_cpu_j \times (1 - utlz\_cpu_j))} \tag{6}$$

The formulas for Q_2-Q_5 are similar. For a certain node, C_i, Q_1, Q_2, Q_3, Q_4 and Q_5 can be determined experimentally, and then the approximate value of k_1, k_2, k_3, k_4 and k_5 can be determined by regression method.

5 Experiment and analysis

The scheme proposed in this paper was verified experimentally. Two experiments were conducted on same cluster. The experiment program is: Some relative softwares (such as CPU-Z, HWMonitor, CoreTemp, etc.) were used to measure temperatures of CPU and GPU of each computing unit at different time during cluster's running, and the temperature curve of each computing unit was drawn according to them.

Seven computers were used to constitute GPU heterogeneous cluster. Five of them have the configuration: model is Lenovo Erazer X700, memory is 16G, CPU is Intel i7-3930k, GPU is NVIDIA GTX660i, operating system is Ubuntu12.04 LTS, cluster environment is hadoop2.2.0, Java version is JDK1.7. The other two have lower configuration: CPU is Intel Pentium (R) Dual-Core E5300 2.60GHz, memory is 4G, GPU is NVIDIA GeForce 9400GT, operating system is Windows 7 64-bit Ultimate. The experimental data is taxi GPS data and data generated continuously by loadrunner.

5.1 Conventional scheduling method

Firstly, conventional scheduling method was used. Only task balanced scheduling was considered, regardless of temperature changes. Every 1 minute temperature was sampled once. The result is shown in Figure 2, where C1, C2, ..., C7 is each computing node.

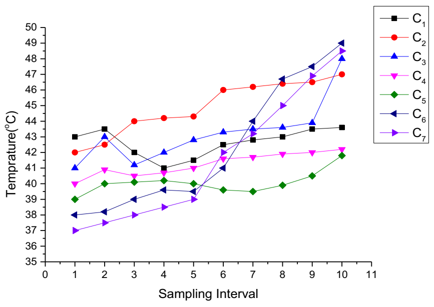

Figure 2. Temperature change in conventional scheduling method

The cluster processes taxi GPS data firstly. The data amount is larger, but because it is historical data, it does not take long time to process it. Temperature and power are measured with measuring instruments, temperatures of CPU, GPU and so on are monitored with softwares. It is found that the temperature and power of CPU and GPU are increasing during processing, but the task has been finished before temperature rises to the set threshold, and the problem of temperature heat island and reliability does not occur. Then simulation data that loadrunner software continues to generate is processed. At this time, CPU and GPU temperature continues to rise, energy consumption continues to increase. After a certain time, temperature exceeds the threshold and temperature heat island is formed, computing result error occurs. The difference between the lowest and highest temperatures of various computing nodes is about 12 °C.

5.2 Scheduling scheme proposed in this paper

In the second experiment, the same experimental environment and data were used, but the scheduling scheme preventing temperature heat island proposed in this paper was used. During processing task, temperature is collected. The result is shown in Figure 3.

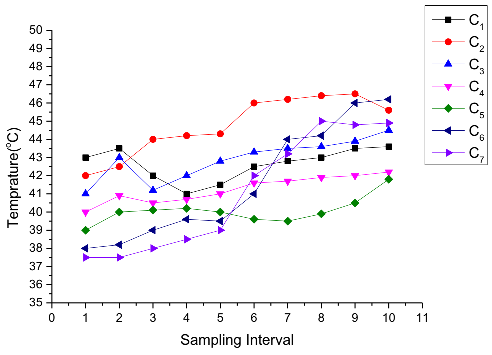

Figure 3. Temperature change in scheduling method preventing temperature heat island

The result of processing taxi GPS data is similar to the previous, but the result shows that the difference of temperature and energy consumption of each node tends to decrease. This shows that this scheme is more time-balanced in task scheduling to prevent temperature heat island from occurring and guarantee overall stability. Data streams generated continuously by program are processed by cluster. It was found that, although the temperature of CPU and GPU increased, the temperature and power did not increase continuously when temperature rised nearly to threshold value, and no temperature heat island and computing error occurred. When data supply amount was increased, the phenomenon that temperature and power increase did not occur. This shows that the scheduling scheme trys to balance running time, inhibit the increasing of temperature and energy consumption to prevent temperature heat island from occurring. The difference between the lowest and highest temperatures among various computing nodes is about 9 °C.

Analyzing above experimental results, it is shown that, if conventional method is adopted, the temperature of each computing node increases continuously with the processing of task, the temperature of some nodes exceeds threshold, and the temperature fluctuates greatly. However, when the scheduling method proposed in this paper is used, temperature is also rising, but because task division makes node running time be consistent as far as possible, the range of temperature fluctuation is small, the overall temperature change is relatively calm, thus it is avoided that the temperature heat island occurs.

Conclusion

The GPU heterogeneous cluster task scheduling scheme proposed in this paper avoids long running time of individual nodes as far as possible, prevents temperature heat island from occurring, guarantees computing reliability, controls energy consumption in a certain range, and also considers the constraints among temperature, reliability, performance and energy consumption, minimizes energy consumption or improves processing speed as far as possible under the premise of ensuring reliability. The main work of next step is to study how GPU heterogeneous cluster perceives and predicts cluster temperature and its variation, and apply it to cluster task scheduling.

Acknowledgments

This research project is supported by the joint special project of Shandong Provincial Natural Science Foundation (Project No.: ZR2015FL014) and the special project of Shandong Provincial Independent Innovation and Achievement Transformation (Project No.: 2014ZZCX02702).

References

1. C. Q. Yang, F. Wang, Y. F. Du, et al. *Adaptive optimization for petascale heterogeneous CPU/GPU computing*. *The 2010 IEEE Int'l Conf. on Cluster Computing*. (2010)
2. L. Chen, O. Villa, S. Krishnamoorthy, G. R. Gao. *Dynamic load balancing on single- and multi-GPU systems*. *The 2010 IEEE Int'l Symp. on Parallel & Distributed Processing (IPDPS)*. (2010)
3. E. Hermann, B. Raffin, F. Faure, T. Gautier, J. Allard. *Multi-GPU and multi-CPU parallelization for interactive physics simulations*. *The 16th Int'l Euro-Par Conf. on Parallel Processing: Part II (Euro-Par 2010)*. Berlin, Heidelberg: Springer-Verlag. (2010)
4. L. Bertini, C. B. Julius, D. Mosse. *Power optimization for dynamic configuration in heterogeneous web server clusters.* Journal of Systems and Software, **83**(4): 585-598. (2010)
5. H. F. Wang, Y. P. Cao. *GPU Power Consumption Optimization Control Model of GPU Clusters.* Acta Electronica Sinica, **43**(10): 1904-1910. (2015)
6. H. P. Huo, X. M. Hu, C. C. Sheng, B. F. Wu. *An energy efficient task scheduling scheme for node-layer heterogeneous GPU clusters.* Computer Applications and Software, **30**(3): 283-286. (2013)
7. H. Liu, J. G. Wang, Z. Z. Ge, et al. *Self-learning Load Balancing Scheduling Algorithm for GPU Heterogeneous Cluster.* Journal of Xi'an Shiyou University, **30**(3): 105-111. (2015)