Enhancing Cloud-based Servers by GPU/CPU Virtualization Management

Tin-Yu Wu[1], Wei-Tsong Lee[2], Chien-Yu Duan[2]

Department of Computer Science and Information Engineering, National Ilan University, *Taiwan, R.O.C.*[1]
Department of Electrical Engineering, Tamkang University, Taiwan, R.O.C.[2]
tyw@niu.yku.edu.tw, wtlee@mail.tku.edu.tw, jason84195@hotmail.com

Abstract- This paper proposes to add the multithreaded Graphic Processing Units (GPUs) to some virtual machines (VMs) in the existing cloud-based VM groups. To handle the multidimensional or multithreaded computing that a CPU cannot process quickly by a GPU that has hundreds of Arithmetic Logic Units (ALUs), and to regulate the time for initiating physical servers by real-time thermal migration, our proposed scheme can enhance the system performance and reduce the energy consumption of long-term computing. Four major techniques in this paper include: (1) GPU virtualization, (2) Hypervisor for GPU, (3) Thermal migration implementation, and (4) Estimation of multithreaded tasks. In no matter quantum mechanics, astronomy, fluid mechanics, or atmospheric simulation and prediction, a GPU suits not only parallel multithreaded computing for its tens of times performance than a CPU, but also multidimensional array operations for its excellent efficiency. Therefore, how to distribute the computing performance of CPUs and GPUs appropriately becomes a significant issue. In general cloud computing applications, it is rarely seen that GPUs can outperform CPUs. Furthermore, for groups of virtual servers, many tasks actually can be completed by CPUs without the support of GPUs. Thus, it is a waste of resources to implement GPUs to all physical servers. For this reason, by integrating with the migration characteristic of VMs, our proposed scheme can estimate whether to compute tasks by physical machines with GPUs or not. In estimating tasks, we use Amdahl's law to estimate the overall performance include communication delays, Synchronization over head and me possible additional burden.

Keywords: Virtual Machine (VM), Multithreading, GPU, CUDA, Mapreduce

I. INTRODUCTION

In traditional computer science, computers processed tasks mainly by Central Processing Units (CPUs). However, the development of CPU has recently encountered bottlenecks because the computation speed-up of single-core processors may result in overheating and power consumption problems. Therefore, in place of single-core processors, multi-core processors are gradually used for parallel computing to enhance computer performance.

Parallel computing in the early days was usually executed by several computers and processed by traditional CPUs. Thus, organizations those needed to process large amounts of data established large-scale multicomputer systems and exchanged data through Message Passing Interface (MPI). In such a kind of multicomputer environment, every computer is a computational node, which has its own CPU, memory and networking interface. Thus, a multicomputer system usually transforms a parallel program into single program multiple data (SPMD) for every computer in the system to operate the same program but process different data.

In the past, display cards were defined as the auxiliary to CPUs to process image and graph related tasks. Later, GPU was presented to reduce display cards' dependence and occupancy of CPUs. Although the number of computing units on GPUs was not large previously, the computing ability of display cards has been enhanced recently: not only the improvements of computing clock, but also the increasing number of GPUs on display cards, which enhances the floating-point operations per second. Instead of being designed to finish heavy computing tasks within a limited number, GPUs are expected to process large amounts of data and tasks by parallel computing to improve the system performance. Because a large number of GPUs are suitable for parallel computing, many supercomputers in the word have started to use GPUs to support CPUs for a great deal of complicated computing tasks.

Since the future of computers keep stepping into cloud computing, we propose to add the multithreaded GPUs to some VMs in the existing cloud-based VM groups. To handle the multidimensional array operations or multithreaded computing that a CPU cannot process quickly by a GPU that has hundreds of ALUs, and to regulate the time for initiating physical servers by real-time thermal migration, our proposed scheme can estimate whether to compute tasks by GPUs or not, enhance the system performance, and reduce the energy consumption of long-term computing.

II. RELATED WORKS

2.1 CPU/GPU Collaborative Computing

In modern computer science, traditional CPU computing has reached a bottleneck while high performance computing systems are experiencing a revolution, in which novel architectures are presented one after another and the combination of multi-core microprocessor and GPU is one of the highest potential and prospective method.

GPU (Graphics Processing Unit) was first presented by NVidia in 1999[1]. With the evolution of semiconductor industry, the growth of GPU has been exceeding Moore's Law and reached more than 500 gigaflops of double-precision floating point operations. As for researches about GPGPU (General-Purpose computation on GPU), papers [2-5] have specified the history, architecture, software environment and several cases of GPU.

Because of its powerful computational capabilities, high cost performance and high performance but low power consumption, GPU has received great attention in such an eco-friendly era. In addition to traditional graphical computing, GPU has been greatly applied to general-purpose computing and thus formed GPGPU or General-purpose computing on graphics processing units (GP^{2}U). Due to its excellent general-purpose computational capabilities, GPU has been regarded as the future of computer science since 2003 [2].

CPU and GPU are designed with absolutely different goals. The design concept of CPU is to execute instructions and operations quickly with low delay and to use a great deal of IC for control and temporary storage. On the other hand, GPU is designed for graphical computing, in which great amounts of IC are used as ALUs for high intensity computing. Therefore, by utilizing CPU/GPU collaborative computing, we can use CPU for control and buffer and use GPU for processing a great deal of computing tasks.

In the scope of CPU/GPU collaborative computing, CUDA (Compute Unified Device Architecture)[5] presented by NVidia is currently the leading technique of GPGPU. The CUDA is a C-language development environment, in which tasks are computed by GPUs after NVidia GeForce 8 together with Quadro GPU. Commands in either CUDA C-language or OpenCL will be compiled into PTX code by driver programs for the display core to compute.

The latest CUDA-x86 compilers can support traditional multi-core CPU architecture and execute all parallel programs written by CUDA. Instead of outperforming CPU in all computing aspects, GPU only surpasses CPU in matrix computing and parallel computing, which are still rarely seen in the present program structure. Thus, when a VM of a physical server without GPU executes parallel computing, the system will estimate the computing cost of GPU servers. Supposing the computing amount is not large, we use CPU for parallel computing. On the contrary, we will use GPU for a great deal of computing amount.

2.2 GPU Virtualization

When cloud computing becomes the future of computers, how to virtualizes all computer interfaces and optimize the computer performance is the goal for all cloud service providers. While GPU computing has been integrated into new computer structure, traditional virtual structures will be challenged. VMware, the leading company in the virtualization, has presented some concepts about GPU virtualization in [6]. Like traditional virtualization, physical GPU is cut into several virtual GPUs and GPU resource is managed by a resource manager, which is similar to a VM monitor. However, to distribute GPU resource to all VMs equally is not exactly the best method, especially when the VMs are greatly different. For the diversification of server client mode after virtualization, we propose a novel distribution method of GPU virtualization together with thermal migration to achieve reasonable application of GPU.

2.3 MapReduce

[7] proposes to use GPU based on a computing concept similar to Hadoop. According to MapReduce, the program is first sent to a master node. In the Map phase, the proposed scheme divides the program into suitable sizes, distributes the tasks equally to worker nodes, and tracks the tasks. After the nodes complete the tasks, the worker nodes collect the results by Reduce, which can greatly decrease the computing time. But, one great restriction of this scheme is that only when the computers at all ends belong to the same specification and the type specification of CPU and GPU are the same, can the scheme find out α, the performance ratio of a GPU map task execution to a CPU map task execution. In addition, this scheme does not consider the loading conditions of all worker nodes in the virtual environment and the time difference may occur to parallel computing under different loading conditions, which causes further delay in Reduce.

III. ENHANCING CLOUD-BASED VIRTUAL SERVERS BY GPU PARALLEL COMPUTING

3.1 Integrating GPU into Cloud Server Virtualization

In the virtualization architecture, because GPU is one of the necessities for future computers, GPU virtualization is inevitable. According to the basic architecture of VMs and the initial ideas about GPU virtualization presented by VMware, GPU is virtualized, just like CPU and other computer devices, for resource management. As shown in Figure 1, each VM has a pass-through GPU to form a channel to stride the resource manager for GPU utilization and to establish GPU driver for Apps on all kinds of VMs. Moreover, there is another channel from the resource manager to Emulation for GPU management.

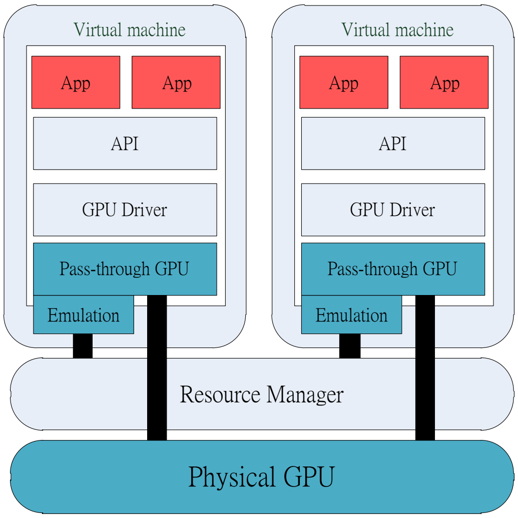

Figure 1 GPU virtualization of VMware (Source: VMware)

Nevertheless, GPU is not suitable for public sharing because large number of data transmissions will influence the efficiency of GPU computing. Furthermore, in general cloud computing applications, only high-performance and multithreaded computing, including real-time image processing, atmospheric simulation and prediction, astrophysics, quantum mechanics, fluid mechanics, etc., can make good use of GPU computing.

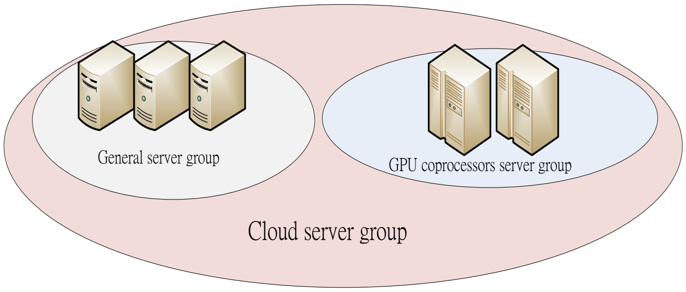

Figure 2. Cloud Server Group

Therefore, we propose to add GPUs to a cloud server group and classify the servers into two subgroups: general server group that occupy the great majority of the group, and GPU coprocessors server group, as displayed in Figure 2. As for GPU coprocessors server group, we suggest that a VM occupies a GPU at one time to complete one single task within the minimum time. Figure 3 shows that only one VM controls one GPU at one time. When a VM needs a GPU, the VM monitor gives the GPU usage right to the VM and completes the computing task by MapReduce within the minimum time. Finally, the GPU usage right will be returned to the VM hypervisor.

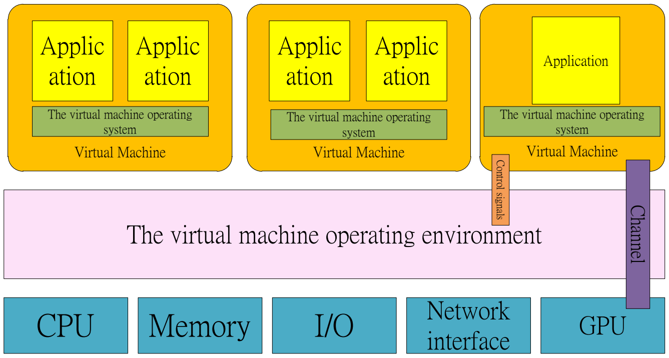

Figure 3 Architecture of GPU Virtualization

By making an improvement of the method presented in [8], our proposed scheme aims to achieve balance control but the control node does not take charge of all data transmissions for fear of causing heavy burdens. In our opinion, the balance control node is only responsible for resource management while all work nodes have to return CPU loading, GPU loading and the current user list to the balance control node.

Our task scheduling is displayed in Figure 4. First, the client sends the task to the cloud server and the master node estimates the task. Second, when the program asks for more CPU/GPU resource, the master node cuts the task into small tasks of the same size. By referring to [7], we can find out the performance ratio of a GPU map task execution to a CPU map task execution, α. Let

$$\alpha = \frac{meanmaptaskexecutiontimeonCPUcores}{meanmaptaskexecutiontimeonGPU\ cores} \qquad (1)$$

Then we can get the number of small tasks. Third, according to the quantity of tasks, the program requests the computing resource from the balance control node, which distributes obtainable resource to available work nodes in the following step. Based on the available resource, the master node determines the optimal task allocation and maps them to each work node for computation. Finally, the master node reduces the calculation results and informs the control node the completion of the task to release computing resource.

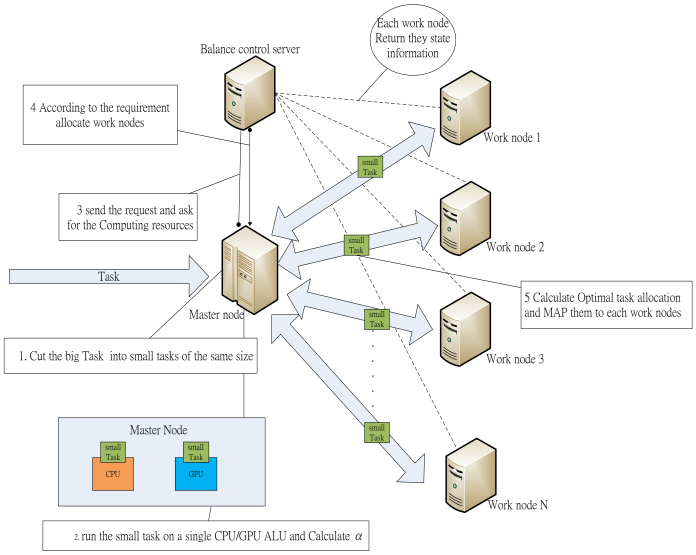
Each work node Return they state information
Balance control server
4 According to the requirement allocate work nodes
3 send the request and ask for the Computing resources
Task
Master node
1. Cut the big Task into small tasks of the same size
Work node 1
Work node 2
Work node 3
Work node N
small Task
5 Calculate Optimal task allocation and MAP them to each work nodes
Master Node
CPU
GPU
2. run the small task on a single CPU/GPU ALU and Calculate α

Figure 4 Task Scheduling

3.2 Related Parameters

Related parameters are divided into two kinds. The first is the loading condition of the server, which is sent to the control node as the index for the master node to request computing resource. We define CPU load as:

$$C_L = \frac{Computing\ the\ amount\ of\ use}{The\ maximum\ computation} \qquad (2)$$

and GPU load as:

$$G_L = \frac{Number\ of\ busy\ GPU\ devices}{max\ number of GPU devices} \qquad (3)$$

The resource for the control node to distribute is C_L <90% and 0%<G_L<100%.

The second kind of parameters are related to small task blocks and total calculation time. Let

- α be the CPU/GPU calculate time rate.

- C_n be the number of Assigned CPU cores.
- $G_n$be the number of Assigned CPU cores.
- $t$be the time for 1 GPU core calculate a task.
- $T = max\{\frac{x}{C_n}\alpha t, \frac{y}{G_n}\}$be the task computation time.

According to Amdahl‘s law, we know that

$$Speedup = \frac{1}{s+\frac{p}{N}} \qquad (4)$$

where s denotes the part that is not improved in the system, p refers to the improved part in the system, and N means the enhance ratio. Next, we will integrate our defined parameters with the formula. Because our assumed scenario is a cloud-based parallel architecture, in which extra overhead must be computed, the formula is revised into:

- T_s = serial code Execution time
- $T_{p=}$parallelizable codeExecution time
- T_o = overhead expend time

$$Speedup' = \frac{T_s + T_p}{T_s + \frac{T_p}{N} + T_o} \qquad (5)$$

IV. Performance Simulation and Analysis

According to the architecture presented in the previous section, we made the following simulation and analysis: Assume that the specifications of the cloud servers are the same and the operational speed of a single CPU core is five times faster than a single GPU core. Each cloud server has four CPU cores and two CPU devices (2*128cores). The considered overhead include the time for data transmissions and MapReduce. Five kinds of program types are taken into consideration:

1. 1% serial code and 99% parallelizable code
2. 5% serial code and 95% parallelizable code
3. 10% serial code and 90% parallelizable code
4. 25% serial code and 75% parallelizable code
5. 50% serial code and 50% parallelizable code

Suppose that all servers are free.

Figure 5 reveals the parallel system gain according to traditional computer based concept. In our simulation, the system gain reaches the maximum when the number of nodes is unit digit. Also, the more proportion the parallelizable code occupies the program, the better the efficiency will be. When the parallelizable code occupies 99% of the program, the system gain reaches 89 times when there are 3 nodes. However, when the parallelizable code occupies 90% of the program, the system gain reduces quickly and the maximal system gain is 19.5 times when the number of nodes is 3 or 4. When the parallelizable code occupies only 50% of the program, the gain is only 2 times at most.

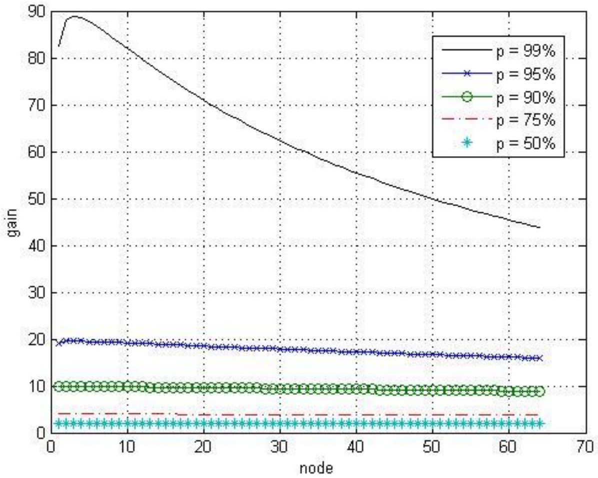

Figure 5 Theoretical gain of servers

Next, without considering the execution time for collaborative computing, we analyzed the transmission time and the execution time for small chunks. Supposing the execution time for a 10-megabyte serial code on a CPU is 15 seconds, 1% of the program, and 990-megabyte parallelizable code occupies the rest of the program. It takes 1485 seconds to compute the task by CPUs only and 29 seconds by two GPUs. But, it takes only 26.9 seconds to complete the task by 4 CPUs and 2CPUs.

Table 1 Total Execution Time (1% serial code)

	serial code	parallelizable code	Total time
1 CPU	15s	1485s	1500s
2 GPU	75s	29.01s	104.01s
1 node	15s	26.9s	41.9s
2 nodes	15s	13.45s	28.45s
3 nodes	15s	8.967s	23.967s

In the same way, by using one single CPU to using 1-3 nodes, we estimate the execution time of different program types, as displayed in Figure 6.

Figure 6 shows that to use CPUs only for computing, all kinds of program types can be completed within 1500s. If the program code occupies 99%, the task can be completed in 44 seconds and even 20 seconds with the support of more GPUs. However, the more proportion un-parallelized computing occupies, the less improvements parallel computing can make. Theoretically speaking, with the support of more GPUs, the execution time of the parallelizable code should be reduced to 0. But, more GPUs and more nodes in fact will result in more overhead.

In our proposed scheme, because the server's loading condition is considered, few nodes cannot offer enough GPUs for parallel support (compared with the above-mentioned situation that all servers are free). The maximal gain is obtained when there are 15 nodes. But, more nodes will decrease the performance due to synchronization and transmission. Moreover, when the server is near end, our simulation occupies approximately 2 seconds for transmission.

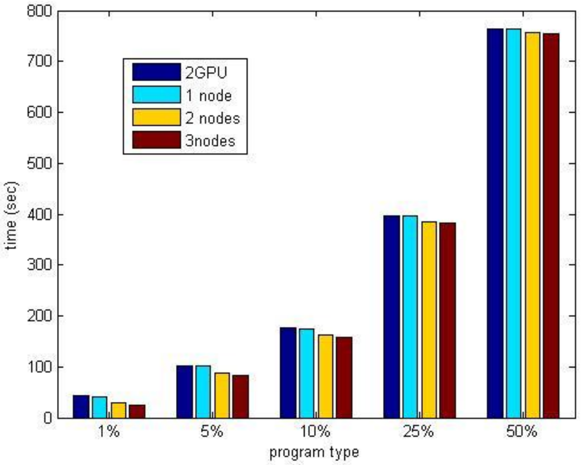

Figure 6 Execution time of different program types in GPU parallel computing

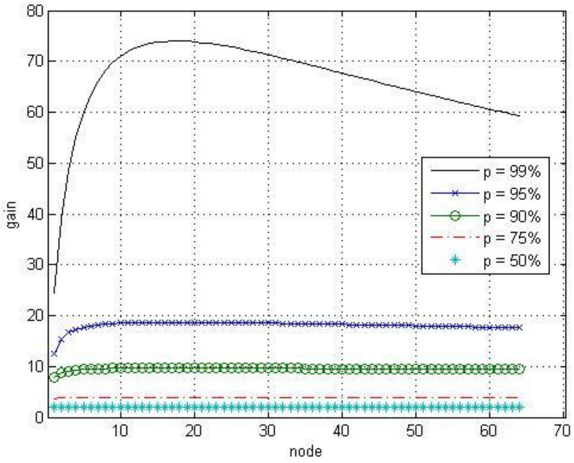

Figure 7 Server Gain under Load Condition

V. Conclusion and Future Objective

This paper proposed a scheme to integrate GPU with cloud computing for users to utilize high-performance but low-cost GPU resource without building devices by themselves. However, the biggest limitation of GPU computing is the market share of parallel computing, which needs to be popularized by future parallel computing service providers and developed/adopted by numerous application developers. Moreover, the transmission amount of the network and the internal bus in the computer is another bottleneck of parallel computing. Supposing the transmission amount can be further enhanced, a great deal of overhead will be reduced and the maximal gain will appear when much more nodes are utilized for collaborative computing.

Acknowledgement

This study was supported by the National Science Council, Taiwan, under grant no. NSC 100-2219-E-032-001.

References

[1] Macedonia, M., ‘The GPU enters computing's mainstream,” IEEE Computer Society, 2003, 36(10):106-108

[2] Owens, J.D. , Houston, M. , Luebke, D., et al ,“GPU Computing,”Proceedings of the IEEE, 96(5) **:** 879 - 899

[3] NVIDIA, NVIDIA CUDA Programming Guide,2010;http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf

[4] Khronos, The OpenCL Specification, 2011; http://www.khronos.org/opencl/

[5] E. Lindholm et al., ‘‘NVIDIA Tesla: A UnifiedGraphics and Computing Architecture,’’IEEE Micro, vol. 28, no. 2, 2008, pp. 39-55.

[6] Micah Dowty ,Jeremy Sugerman ,” GPU virtualization on VMware's hosted I/O architecture,” ACM SIGOPS Operating Systems Review,2009

[7] Shirahata, K. ,Sato, H. , Matsuoka, S. ,” Hybrid Map Task Scheduling for GPU-basedHeterogeneous Clusters,”IEEE International Conference on Cloud Computing Technology and Science,2010

[8] Wenwu Zhu , Chong Luo , Jianfeng Wang ,Shipeng Li ,“Multimedia Cloud Computing”

[9] Daga, M. ; Aji, A.M. ; Wu-chun Feng ; “On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing”, Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on

[10] Nickolls, J. ; Dally, W.J. ;”THE GPU COMPUTING ERA”, Micro, IEEE,2010