Task Scheduling of Parallel Processing in CPU-GPU Collaborative Environment

Lei Wang[1], Yong-zhong Huang[1], Xin Chen[1], Chun-yan Zhang[2]
[1]*Zhengzhou Information Science and Technology Institute, Zhengzhou, Henan 450002, China*
[2]*School of Information Science and Engineering, Henan University of Technology, Zhengzhou, Henan 450052, China*
E-MAIL: wanglei1167@gmail.com

Abstract

With the rapid development of GPU (Graphics Processor Unit) in recent years, GPGPU (General-Purpose computation on GPU) has become an important technique in scientific research. However GPU might well be seen more as a cooperator than a rival to CPU. Therefore, we focus on exploiting the power of CPU and GPU in solving generic problems based on collaborative and heterogeneous computing environment. In this work we present a parallel processing paradigm based on CPU-GPU collaborative computing model to optimize the performance of task scheduling. In addition, we evaluate a new task scheduling algorithm using NVIDIA GeForce 7600GT compare with traditional task scheduling algorithm. The results show that our algorithm increase average performance of 26.5% compared with traditional algorithm. Based on our results and current trends in microarchitecture, we believe that efficient use of CPU-GPU collaborative environment will become increasingly important to high-performance computing.

1. Introduction

With the current continuous improvements of computer performance, more and more applications of different task and demand are fast-growing. As a result, the processor goes in the two directions: general-purpose processor and special-purpose processor. Modern CPUs have been increasing their performance according to Moore's Law based on clock frequency and transistors [1]. Transistors in CPU are not only responsible for the interpretation, implementation and completion of the various commands and arithmetic logic operation, but also control and coordinate the function of most parts of computer. However, Modern GPU (Graphics Processor Unit) is specialized for compute-intensive, highly parallel computation. It has more transistors devoted to data processing rather than data caching and flow control.

In recent years, GPU is rapidly developing in performance with a high speed which has broken up the Moore's Law [2]. As GPU have a highly-efficient and flexible parallel programmable features, a growing number of researchers and business organizations started to use some of the non-graphical rendering with GPU to implement the calculations, and create a new field of study: GPGPU (General-Purpose computation on GPU) [3], and its objective is to use GPU to implement more extensive scientific computing. GPGPU has been successfully used in algebra, fluid simulation, database applications, spectrum analysis, and other non-graphical applications [4].

A large number of research results indicate that GPU in solving compute-intensive problems has great advantage compared with the CPU [5]. However, due to the special usage of GPU, it's impossible to complete all computing tasks solely on the GPU. Many control and serial instructions still need to complete on the CPU. GPU might well be seen more as a cooperator than a rival to CPU.

To utilize GPU for general purpose computations, in fact, is to exploit the power of CPU and GPU in solving generic problems based on collaborative and heterogeneous computing environment, and this viewpoint would be a better solution to solve general computing problems.

Generally, traditional task scheduling algorithm is used to schedule task to the first idle processing unit [6]. However, we should analyze the differences of the heterogeneous processors in performance and bandwidth. In CPU and GPU heterogeneous environment, partitioning and scheduling tasks depend on four characteristics:

- *Computing model of task adapted to the CPU or GPU programming model.*

978-0-7695-3308-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCSIT.2008.27

- *Reasonable amount of computing resources.*
- *Task of data in the capacity of GPU memory.*
- *Bottleneck of data transfers efficiency between CPU and GPU.*

In this paper, we focus on CPU-GPU collaborative computing model, which aims at optimizing the performance of task scheduling. We take the Vector Element Searching Algorithm for example, evaluate our algorithm using NVIDIA GeForce 7600GT compare with traditional task scheduling algorithm. The results show that our algorithm increase average performance of 26.5% compared with traditional algorithm. On this basis, the conclusion of our work can be further extended to the multi-processor for heterogeneous computing environment.

The rest of the paper is organized as follows: The subsequent section gives background of GPU programming model and issue to facilitate understanding of our work. In Section 3, we analyze the bottleneck of CPU-GPU collaborative computing, then describe the CPU-GPU collaborative computing model and our algorithm. Section 4 presents the experiment results with the Vector Element Searching Algorithm, and shows the performance between traditional task scheduling algorithm and our algorithm. Finally the conclusions and suggestions for future work are presented in Section 5.

2. GPU programming model

From a traditional standpoint of graphics processing, GPU is designed to process graphics made up of geometric primitives such as points, line, and triangles. Modern graphics pipeline shown in Figure 1 is mainly composed of Vertex Processor (VP), Rasterizer and Fragment Processor (FP) [3].

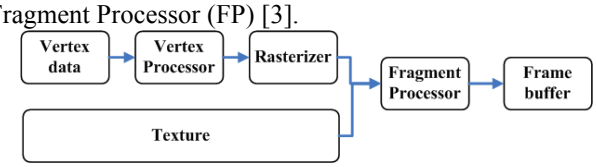

Figure 1. Programmable graphics pipeline

From an alternative point of view, GPU can be seen as a streaming processor contains arrays of VPs and FPs operating in parallel [7]. When the programmer specifies a shader program which is called kernel and a data stream, GPU maps this data onto the available processors to compute the result. Current GPU has Multiple-Instruction, Multiple-Data (MIMD) VPs and Single-Program, Multiple-Data (SPMD) FPs. GPU execute batches of fragment threads in Single-Instruction, Multiple-Data (SIMD) execution style (one thread per pixel).Both VPs and FPs are highly computationally capable [8].

Many research on GPGPU have been presented GPU is a large amount of programmable floating-point horsepower that can be exploited for compute-intensive applications completely unrelated to graphics processing. But GPU is hardly a computational panacea. Its arithmetic power results from a highly specialized architecture [2]. Today, with the rapid development of GPU, many programming challenges and limitations of architecture have been solved by hardware design and programming techniques. A new issue associated with the usage of GPU for general-purpose computation is how to make GPU in collaboration with CPU to achieve a better performance.

3. CPU-GPU collaborative computing model

3.1. Overview

To implement algorithm based on CPU-GPU collaborative computation, tasks partition can be divided into two categories: computing tasks and communicating tasks.

- Computing tasks run in CPU or GPU by several instructions act on data, and then retrieve computing results.
- Communicating tasks are responsible to control the data input and output of computing tasks.

Algorithm can be divided into multiple sub-tasks, and these sub-tasks are implemented through a certain processing flow. The structure of sub-tasks consists of data and operation. Therefore, we can map sub-tasks to CPU implementation or GPU implementation by scheduling sub-tasks to different hardware.

Hierarchical control data flow graph can be used to describe computing model of GPU-CPU heterogeneous environment. HCDFG allows nesting of the traditional data flow hierarchically [9]. Especially, it's not only a well description of multi-tasks in a algorithm, but can also optimize performance of scheduling fine-grained sub-tasks between CPU and GPU. As shown in Figure 2, Computing tasks are operating nodes, and Communicating tasks are transmitting nodes, edges between nodes describe direction of data flow.

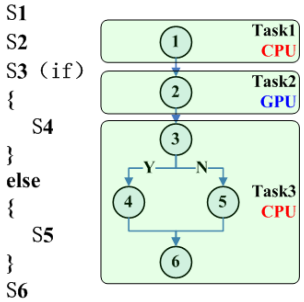

Figure 2. Paradigm of task partition in CPU-GPU collaborative environment

In order to exploit task parallelism on coarse grain decomposition, algorithm can be divided into many sections. For example, segment *S1* to *S6*. Therefore, we can map the implementation of each segment to CPU or GPU.

For simplicity, we evaluate the case of one CPU and one GPU. The execution time T_{ALL} is defined in equation (1). We denote the execution time of CPU as T_{CPU}, the execution time of GPU as T_{GPU}, and the communication time between CPU and GPU as T_{COMM}.

$$T_{ALL} = T_{CPU} + T_{GPU} + T_{COMM} \quad (1)$$

From equation (1), we can identify the following essential characteristics associated with how to minimize T_{ALL} to achieve maximize performance:

- Communication time between CPU and GPU, this might be a bottleneck of CPU-GPU collaborative computing environment.
- The paradigm of mapping sub-tasks to CPU implementation or GPU implementation by scheduling sub-tasks to different hardware.

3.2. Bottleneck of CPU-GPU collaborative computing environment

From the point of view of CPU-GPU collaboration, we can consider CPU as master processor can both distribute tasks and execute tasks, while GPU can only execute tasks. In order to analyze the performance bottlenecks of CPU-GPU collaborative computing, we exam the speed of upload-bandwidth (transfer results of GPU to CPU memory) and download-bandwidth (transfer data of CPU to GPU memory) to investigate the data transfers performance.

We implement the shaders by GLSL [10] to test upload-bandwidth and download-bandwidth between CPU and GPU are simple operations shown in Figure 3.

```
[Upload-bandwith shader]
!!ARBfp1.0
MOV result.color.xyzw,
      fragment.color;
END
```

```
[Upload-bandwith shader]
!!ARBfp1.0
TEMP R0;
TEX R0, fragment.texcoord[0],
      texture[0], RECT;
ADD result.color, R0, R0;
END
```

Figure 3. Shaders for bandwidth performance evaluation

For the experiments presented in this paper we used NVIDIA GeForce 7600GT as graphics card. This card has an G73 graphics processor clocked at 560MHz, 256MB DDR3 video memory clocked at 1400MHz and the data transfers with the PC are done through the PCI-Express 16x interface. The G73 processor includes 5VPs and 12FPs. As for the computer systems we used Intel Pentium D 820 based system with 2GB RAM.

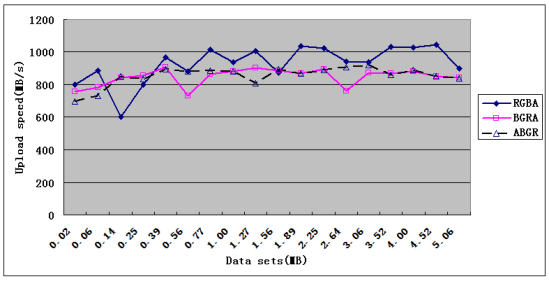

Figure 4. Upload-bandwidth of CPU-GPU performance

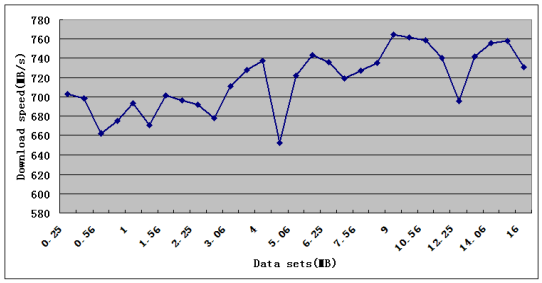

Figure 5. Download-bandwidth of CPU-GPU performance

As shown in Figure 4 and Figure 5, experiment results of bandwidth are much lower than the theoretical bandwidth of PCI-Express 16x (4GB/s). Therefore, data transfers efficiency between CPU and GPU is a bottleneck and it is more of a bottleneck than the hardware's capacity for computation in CPU-GPU collaborative environment.

3.3. Task scheduling algorithm

Task scheduling algorithm of CPU-GPU collaborative environment indicates relationship among computing capability of processor, communicating cost, and size of data sets. At first, we tentatively put forward three assumptions as the basis of task scheduling of parallel processing in CPU-GPU collaborative environment, and then an algorithm is given in this section later.

The computing capability of processor is denoted as *CP*, communicating cost of GPU (communication between CPU and GPU) as *(CM)g*, communicating cost of CPU (communication between CPU and main memory) as *(CM)c*, size of data sets as *k*. We assume following three assumptions**:**

Assumption 1:

$$k \propto (CM)_c \ , k \propto (CM)_g \quad (2)$$

Assumption 2:

$$\frac{\Delta(CP)}{\Delta k} < \frac{\Delta(CM)_c}{\Delta k} < \frac{\Delta(CM)_g}{\Delta k} \quad (3)$$

Assumption 3:

$$(CM)_c^k << (CM)_g^k \quad (4)$$

The assumptions described above have two characteristics. First, when the size of data sets changes, the communicating cost will be considered as a major factor of performance. Second, the GPU and CPU both have independent local memory. Communication can be divided into three parts: communication between GPU and video memory, CPU and main memory, and interface between CPU and GPU. As we described in section 3.2, data transfers efficiency between CPU and GPU is a bottleneck. Therefore, communication cost between CPU and GPU is most important than other costs.

On the basis of the concept mentioned above, we provides an algorithm for task scheduling of parallel processing in CPU-GPU collaborative environment.

Step 1: Ignoring traffic *CM*, we can get the executed time of GPU and CPU separately, which is $T_g(CP_i)$ and $T_c(CP_i)$ under some calculational measures by analyzing the different measures of the task. Supposing $T_g(CP_i) < T_c(CP_i)$ is right when i is located in the interval of *[m, n],* GPU will execute this task, otherwise CPU will do.

Step 2: By analyzing the different calculational measures of CP_i and corresponding traffic CM_i of the task, under some calculational measures we can get the executed time of GPU and CPU, which is $T_g(CP_i)+T_g(CM_i)$ and $T_c(CP_i)+T_c(CM_i)$, if the following equation is right when i is located in the interval of *[m, n],* GPU will execute this task, otherwise CPU will do.

$$T_g(CP_i)+T_g(CM_i) < T_c(CP_i)+T_c(CM_i) \quad (5)$$

Step 3: Based on step 2, we optimize the scheduling further. We suppose that code segments can be divided into subtasks $\{Task_{sub1}, Task_{sub2} \ldots Task_{subn}\}$. Under the premise that the size of data sets is the same both on CPU and GPU for each subtask, the compute-intensive subtask will be executed by GPU.

4. Performance evaluation

We take the Vector Element Searching Algorithm for example, and experiment our algorithm using NVIDIA GeForce 7600GT compared with traditional task scheduling algorithm. The Vector Element Searching Algorithm is composed of comparative operations and calculational operations (MAD, POW, SIN etc.). We use texture to load vector elements of RGBA format in video memory, and then use 12 FPs to execute comparative operations and calculational operations parallel.

We use two methods to evaluate the performance, and $(CM)_c$ is always ignored. The size of data sets indicates the number of single floating-point values.

Method 1: One processor to complete the task using traditional task scheduling algorithm.

Method 2: Both CPU and GPU complete the task in a collaborative environment using our task scheduling algorithm.

Table 1. Measurements from the Vector Element Searching Algorithm of two methods

Size of Data sets	Time (ms)							Speedup		
	CPU		GPU			Upload	Download	Method1 kernel	Method1 overall	Method2 overall
	Method1	Method2	Method1 kernel	Method1 overall	Method2 overall					
2^8	0.002	0.052	1.137	5.270	5.101	0.872	1.813	0.002	0.001	0.010
2^{10}	0.006	0.109	1.308	4.090	3.974	0.872	2.416	0.005	0.003	0.027
2^{12}	0.022	0.198	1.129	3.718	3.987	0.896	1.914	0.020	0.006	0.050
2^{14}	0.084	0.286	1.327	4.268	4.098	0.986	1.958	0.063	0.020	0.070
2^{16}	0.335	0.756	1.133	4.401	4.159	1.311	2.827	0.296	0.076	0.182
2^{18}	1.337	8.965	1.485	6.489	5.125	2.178	3.428	0.900	0.206	1.749
2^{20}	5.629	16.589	1.605	14.208	12.568	6.238	6.941	3.507	0.396	1.320
2^{22}	25.045	87.589	3.097	43.902	35.589	22.580	23.417	8.087	0.570	2.461
2^{24}	88.682	168.560	9.370	175.250	117.560	87.566	84.335	9.464	0.506	1.434

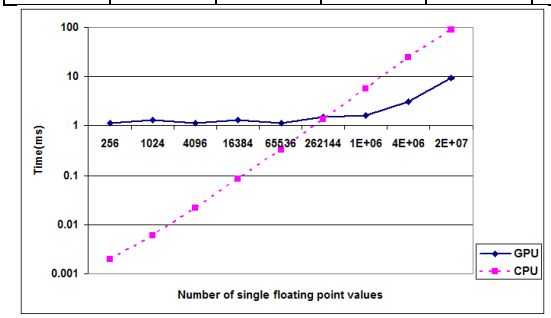

Figure 6. Using *method 1* and ignore *(CM)g*, comparing $T_g(CP)$ with $T_c(CP)$

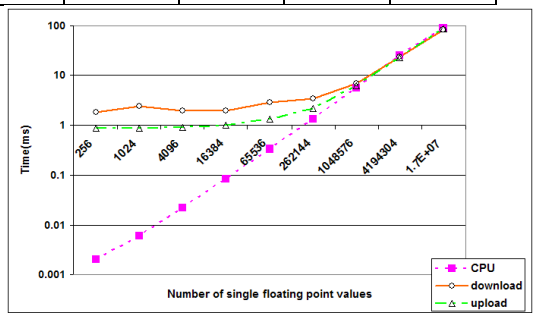

Figure 7. Using *method 1*, comparing $T_g(CM)$ with $T_c(CP)$

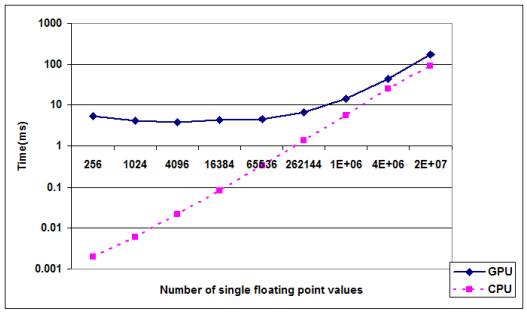

Figure 8. Using *method 1*, comparing $T_g(CP)+ T_g(CM)$ with $T_c(CP)$

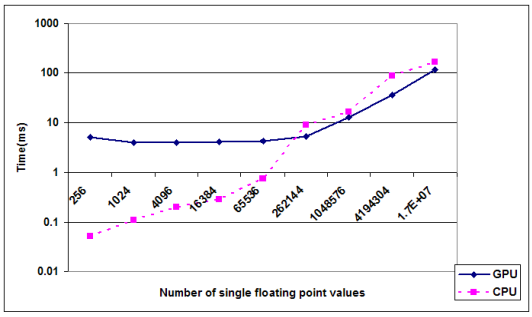

Figure 9. Using *method 2*, comparing *Tg(CP)+ Tg(CM)* with *Tc(CP)*

We use computing intensity to express the ratio of calculational measures to traffic, namely:

$$I = CP / CM \quad (6)$$

Because that the computing intensity of comparison operation lower than that of mathematical operation, the comparison operation will be executed by CPU while the mathematical operation will be done by GPU. From what Figure 6 to Figure 9 show that even $T_g(CM)$ *is* bottleneck, *method 2* is increased average performance of 26.5% compared with *method 1*.

5. Conclusion and Future Work

This work focused on how to exploit computation performance in CPU-GPU collaborative environment. The contributions of the work presented in this paper are twofold. First, this work presents CPU-GPU collaborative computing model and algorithm to optimize traditional GPU computing model. Second, this work analyzes the performance of two task scheduling methods to prove our model and algorithm.

The results of experiments indicate that our scheduling algorithm can better adapt to the collaboration of heterogeneous computing processors. In our experiments, we can obtain two conclusions.

(1) On the condition that the size of data sets is the same, we rescheduled the algorithm according to the intensity of calculating. The subtasks whose computing intensity is higher should be executed by GPU, and the one whose calculating intensity is lower should be executed by CPU.

(2) Because of the bottleneck of transfers bandwidth between GPU and CPU, the influence of subtasks with lower computing intensity is far smaller to CPU than to GPU.

Usually, the compute-intensive tasks have higher ratio of computation than communication. Higher calculating intensity can get the most efficiency while executed by GPU. Compared with the traditional task scheduling algorithm, the algorithm presented in this paper taking into account the differences between heterogeneous processors and their respective characteristics, which can obtain better performance in practical application.

In the future, we will evaluate performance in multiple CPUs and GPUs environment, and analyze the optimization of load balancing to achieve higher performance.

Acknowledgements

The authors would like to thank Lin Meng, Yang Qu, and all GPGPU research group members for their valuable comments.

References

[1] M. Ekman, F. Wang, and J. Nilsson, “An in-depth look at computer performance growth,” *SIGARCH Comput. Archit. News*, vol. 33, pp. 144–147, March 2005.

[2] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kr¨uger, A. Lefohn, and T. Purcell, “A survey of general-purpose computation on graphics hardware,” in *Eurographics 2005, State of the Art Reports*, August 2005.

[3] “www.gpgpu.org.” http://www.gpgpu.org.

[4] I. Buck, “Taking the plunge into GPU computing,” *GPU Gems 2*, March 2005.

[5] C. Thompson, S. Hahn, and M. Oskin, “A framework and analysis of modern graphics architectures for general-purpose computing,” in *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pp. 215–226, November 2002.

[6] C. Hua Pin, H. Liu Sheng, “Taxonomy of Task Scheduling in Parallel and Distributed Computing”, *Computer Science*, January 2001.

[7]“developer.nvidia.com.”,http://developer.nvidia.com/page/home.html.

[8] I. Buck, “Taking the plunge into GPU computing,” *GPU Gems 2*, March 2005.

[9] K. N. Levitt, W. T. Kautz. Cellular arrays for the solution of graph problems. *Communications of the ACM*, vol. 15(9), 1972.

[10] J. Kessenich, D. Baldwin, and R. Rost, “The opengl shading language, version 1.10, rev. 59.”