GPU-accelerated OS Kernel

Weibin Sun, Robert Ricci
University of Utah

Abstract

The throughput-oriented General Purpose Graphics Processing Unit(GPGPU) computing has already been adapted by many high performance computing applications including scientific computing, computer graphics, data mining...(**from [2, 1] and others**) and gained significant speedup over the CPU only computing. However, few applications of GPGPU acceleration are made on the operating system kernel. That's because the tasks on GPU are massively parallelized to improve the throughput rather than latency and on the contrary OS kernel is designed to be time efficient to reduce latency for fast response. In this paper, we argue that the OS kernel can also benefit from the GPU acceleration by offloading the traditional tasks of OS from CPU to GPU. We present a modified Linux kernel that can run and communicate with some of its tasks on GPU in time-efficent way with the help of well-designed latency-oriented GPU kernels(programs running on GPU also called kernels).

1 Introduction

This section introduces the heterogenous trend of our computer architecture, focus on GPU+CPU. Ref to Helios, Multikernel. Other processors except CPU are more and more powerful. Give some examples of the GPU computing speedup from the references. Most current PCs are with GPUs. So it is a waste if GPUs can only be used by computer games or videos. The so-called GPGPU, general purpose computing is not as general as it literrally means. Those computing are mostly scientific computing, database query and data mining and .... The OS itself doesn't benefit from GPU at all. So it will speedup the whole system if OS can be run on GPU. There are some papers(PacketShader, Gnort, malware-detection) try to accelerate some computation intensive tasks of system with GPU. Their results are excerllent but the latency sucks(give examples from the packetshader latency, gnort latency-IDS is not online checking at line rate, malware detection is not time-critical.). (Need more on our idea)We try to offload the tasks in OS kernel to GPU so that accelerate it. However, GPU is not fully programmable so that we can't simply copy our OS kernel to it and run just like multi-core. It acts as a co-presessor that works in a request-response way. (More on GPU computing model/limit) The memory copy between GPU and CPU also time-consuming, the kernel launches and completion are expensive in the OS kernel case. So we need a time-efficient design of the GPU kernels in order to accelerate our OS.

<table or fig to show the memory copy benchmark>

Figure 1: Wonderful flowchart

2 Problems of GPU in Latency-oriented Apps

Memory copy and many small kernels launch are time-consuming. First we need point out that OS kernel tasks are not very long and hence many small kernels. Also notice that the GPU working model is launching a kernel for a task.

2.1 Memory copy benchmark results

Analyse memory copy. Ref to 1. Point out the PINNED memory can help although it cost many physical memory. The answer is that current physical memory is enough, compared with the 1.5GB GTX480. Also mention that our design of the NSK can reuse the memory for different tasks and hence the cost is fixed size.

2.2 Kernel launch and completion

show our kernel launch benchmark results. The kernel latency is not only the launch but also the completion! Packetshader paper presented launch latency only. The solution is to eliminate launch and completion times by using NSK. Show our NSK results. Also notice NSK uses PINNED memory to communicate with CPU.
for noindent only, ignore this.

3 A GPU-accelerated Linux Kernel

A figure to show the NSK system, including kernel side, use side, GPU side. Explain them later. Another figure to show the ideal system, which CUDA can be invoked inside OS kernel. Say something about the Nvidia limitations.

3.1 Kernel Side

Intercept normal flow, put request data to the request queue, then done. A callback is set for that request when response OK.

3.2 User Side

Block reading or select-based polling to get requst from kernel, then request the NSK at the GPU side, notice the memory management. The result becomes response write to another fd rather than the request fd.

3.3 GPU Side

Launch when new task functions added, a master manages requests, fills responses. Slaves do the real work. Mention the memory access and synchronization/fence. Say more on the commmunication.

4 Evaluation

Simply show the comparison between: NSK, many kernel and vanilla Linux.

5 Discussion

Fast communication, fast launch, fast completion. Fixed tasks, to run new task, re-launch the kernel with new tasks filled in the task-function array. Fixed memory preallocated, but OK becasue of reuse among tasks.

6 Conclusion

We have presemted a GPU-accelerated Linux kernel. **It shows the feasibility of accelerating OS tasks with GPU, not only the traditional 'general purpose' computing.**

7 Acknowledgments

Thanks to guys and funds should be thanked.

References

[1] GARLAND, M., AND KIRK, D. B. Understanding throughput-oriented architectures. *Commun. ACM 53* (November 2010), 58–66.

[2] OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRGER, J., LEFOHN, A., AND PURCELL, T. J. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum 26*, 1 (2007), 80–113.

Notes

[1] Remember to use endnotes, not footnotes!