Linux Kernel Driver Support to Heterogeneous System Architecture

Wenbo Zhang [1] [2]
College of Computer Science, Beijing University of Technology
100 Ping Le Yuan, Chaoyang District, Beijing 100124, China
E-mail: `zhangwenbo@bjut.edu.cn`

Chong Chen
College of Computer Science, Beijing University of Technology
100 Ping Le Yuan, Chaoyang District, Beijing 100124, China

Fei Liu
College of Computer Science, Beijing University of Technology
100 Ping Le Yuan, Chaoyang District, Beijing 100124, China

Zhenshan Bao
College of Computer Science, Beijing University of Technology
100 Ping Le Yuan, Chaoyang District, Beijing 100124, China
E-mail: `baozhenshan@bjut.edu.cn`

Jianli Liu
College of Computer Science, Beijing University of Technology
100 Ping Le Yuan, Chaoyang District, Beijing 100124, China
E-mail: `liujianl@bjut.edu.cn`

With the development of CPU-GPU fused architecture, the heterogeneous Computing was inevitable to achieve more and more higher performance. At the beginning of 2015, the AMDKFD driver could be merged for the upcoming Linux 3.19 kernel, which can provide the hardware abstract and management based on heterogeneous System Architecture (HSA) to the upper-level application. However, at present, the typical workloads for the HSA were still very inadequate, especially in aspect of quantitative, analysis and verification of the fine-grained. Several kind of classical workloads and the prominent heterogeneous features were given in this paper. For the Kaveri APU architecture, the experiment platform would be built. The results show that, the new kernel driver could provide better performance and energy.

ISCC 2015
18-19, December, 2015
Guangzhou, China

[1]This study is supported by the significant special project for Core electronic devices, high-end general chips and basic software products.(2012ZX01039-004)
[2] Correspongding Author

1. Introduction

During the past couple of decades, the classical computer system include different processing units beside the central processing units (CPU), such as the graphics processing unit (GPU), which originally performed specialized graphics computations in parallel. Over time, these processing units have become more powerful and more generalized, which were used to general purpose tasks with powerful efficiency. But these separated processing elements could not work together very well. At the same time, for the software developers, it was difficult to master the rules of writing code on all kinds of processing units.

To fully exploit the capabilities of these heterogeneous processing units and make them working together seamlessly, the designers re-built computer architecture to integrate the disparate compute on a platform into an evolved central processor and to provide unified programming mode. This kind of designed architecture was called heterogeneous System Architecture (HSA). The typical product was the AMD Accelerated Processing Unit (APU), formerly known as Fusion, which was a series of 64-bit microprocessors designed to act as a CPU and GPU on a single chip[1-2]. However, it was very difficult to achieve CPU-GPU collaborative computing, since the different architectures and programming models[3]. Thanks to acting as a kind of hardware reuse and the abstract model, the operating system provide the running and storage models to the applications on HSA, which would directly restricts utilization and performance of resource. At the beginning of 2015, the AMDKFD kernel driver was merged into Linux 3.19, which was used in conjunction with AMD's open-source graphics stack and the recently open-sourced HSA runtime library as initial HSA Linux implementation. Now the above architecture can be tested with AMD Kaveri and Carrizo APU hardware. While, based on the novel architecture, the typical cases for the HSA are still very inadequate[4], especially in aspect of quantitative, analysis and verification of the fine-grained. Several kind of classical workloads and the prominent heterogeneous features were given in this paper. For the Kaveri architecture, the experiment platform would be built. The results show that, the kernel driver could provide better performance and energy.

The remainder of the paper were organized as follows. In Section 2 the related works were given. There were some unique features analysis on Linux 3.19 kernel support to HSA in Section 3. While, in Section 4, we built experimental platform to verify these unique features. At last, the paper was concluded with a summary in Section 5.

2. Related Work

Recently, some researchers did some experiences to compare the fused CPU-GPU chips with discrete CPU-GPU system. Fused architecture (such as APU) replaced the PCIe data transfer overhead by the different shared memory schemes, which was expected to gain the performance improvements.

Delorme et al. settled down two strategies to implement radix sort on the fused APU[5]. At first, in the coarse-grained case, the input data would be sorted between CPU and GPU. And then, these data would be sort individually. At last, the output data were merged to a single output. While, in the fine-grained case, the data would be distributed dynamically after each step. So, a piece of data could not be hold in a specific unit, rather the whole data should be visible to both processing units during the kernel execution. The fine-grained case needs more communication and synchronization overhead than the coarse-grained one. As a result, it also acquired better performance because of it better load balancing.

Some researchers compared the performance of Mem-cached (a key-value store application) on discrete HSA with that on fused HSA. [6] On discrete systems, it took more time to transfer data between the CPU and the GPU than that on fused systems, which negatively affects the performance of the GPU. On excluding the data transfer overhead, however, the discrete GPU provides a large performance gain.

Similarly, researchers concluded that the low overhead on CPU-GPU data transfer is an advantage of fused HSA, which makes it more efficiently than that of a discrete system. [7] But, the contention penalty between the CPU and the GPU on the fused tem was higher than that on the discrete GPU.

Some researchers compared different memory accesses pattern between on the fused HSA and on the discrete one. [8] They concluded that the CPU memory access from GPU on the HSA was nearly as fast as the GPU memory access. The scheme gave the opportunity for the GPU to directly access data on the CPU without copying, and it especially benefits the applications with little data reuse.

While the researchers acquired the performance and energy consumption on discrete and fused HSA for different FFT implementations with different input sizes. [9] With the growing input data size, the energy efficiency of these HSA increases due to better utilization of the data-parallel resources on the GPUs. They also concluded that the power consumption increases with the number of OpenCL kernel calls and increased use of the GPU fetch unit.

3. Unique Featrue Analysis on Linux Kernl Support to HSA

HUMA (Heterogeneous unified memory access) and HQ (Heterogeneous Queuing) were the two core techniques for the HSA. HUMA defines how memory was accessed by the CPU and the GPU. At the same time, HQ defines the collaborative scheme how to communicate and cooperate between the CPU and the GPU. Furthermore, context switching and pageable Memory were also the important techniques supported in HSA.

3.1 HUMA

As the Fig1 showed, with the CPU and the GPU unified addressing and directly data exchanging between them in a bidirectional, HUMA changed the CPU and GPU isolated running pattern thoroughly. Thus, when assigning tasks to GPU, CPU only need to send a pointer (address) of the data to the GPU, and then GPU could directly handle data. After data was dealt with, CPU can directly get the result. This greatly cut down the overhead of data transferring, improved processor performance and reduced the occupied bus bandwidth.

To sum up, HUMA had three features:

1) Bi-Directional Coherent Memory: Both the CPU and the GPU had the same rights to process data. When one changed the data in the memory, the other can see or get the same data soon, vice versa.

2) Pageable Memory: The GPU can directly generate the missing page exception, which no longer limited to the traditional lock memory page table.

3) Entire Memory Space: Both the CPU and the GPU shared the entire memory space, and they also can access and distribute system memory.

In the Kaveri system, with the most advanced HUMA technique, the pattern of accessing to memory has changed a lot. The CPU and the GPU shared the same address space, which meant that the memory address was not specified to the CPU or the GPU respectively, but it can be arbitrarily assigned according to their needs, which avoided the waste of idle space due to the allocation of restrictions. We only need to pass a pointer referring to the data segment and code segment. This approach does not require developers to know that the data address in memory. It becomes easier in programming.

In addition, with IOMMU (memory management unit input/output) technique, the CPU and the GPU would connect with the memory directly, which did not need to pass through the low speed of the graphics card memory bridge. Not only can it increase the speed of access, but also it could cut down the system overhead caused by the transmission of data. By reducing the number of data transfer, the efficiency of the system was greatly improved, which made the performance advantages became more apparent.

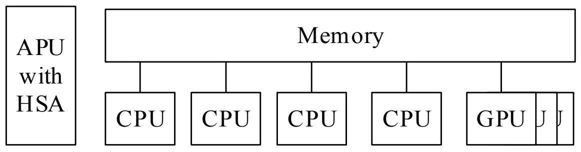


Figure 1: Heterogeneous Uniform Memory Access (HUMA) Technology

3.2 HQ

From the original one, GPU had been a slave-device, which finished the tasks and commands assigned or issued by CPU. In the conventional personal computer, GPU had been used as an external device. By CPU assigning tasks to GPU, Operating system and kernel driver managed GPU. In the past generational APU products, even GPU and CPU had been integrated into APU, this pattern had not changed. As the Fig2 showed, with the HQ technique, the interactive mode between the GPU and the CPU had been changed than that of the past, which made GPU and CPU have the same opportunities.

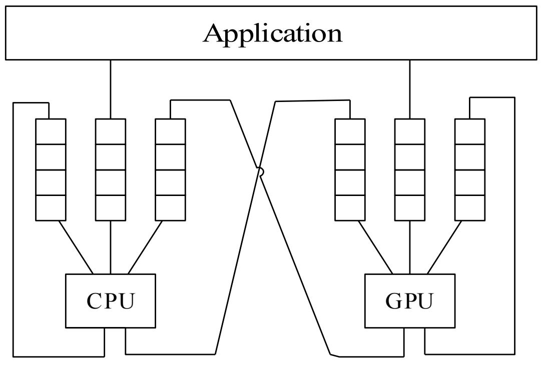


Figure 2: Heterogeneous Queuing (HQ) Technology

In the Kaveri architecture, GPU was not a kind of external device any more, but it can join in its tasks when the task queue exists. Heterogeneous queue allowed applications simultaneously issuing the task to CPU and GPU without interacting with the operating system. CPU can assign tasks to GPU, while GPU can assign tasks to CPU or itself in user mode. Therefore, CPU and GPU were good at different types of tasks, so the task pools can be quickly assigned to the CPU or GPU in accordance with their own characteristics. With the HQ, GPU can undertake the tasks just like CPU equally, which made both can deal with different affairs according to their own features. The above scheme made full use of their computing capabilities and achieved the load balance, which greatly improved the executing efficiency on the whole system perspective.

3.2 Pagable Memory

In order to map the virtual address to the physical address, a mechanism of the page table was proposed, which was called the pageable memory. The structure of the page table was divided into page number and page offset, and the physical address corresponding to the virtual address can be known by the query page table. To put the program into memory in the form of page, it would produce page fault exception when you want to access the page which was not called the memory, because it was not a one-time transfer complete program. And the system needed to assist the redeployment of the desired page.

In the system architecture of CPU as the main body in the past, if it generated the page fault exception, the page needed to be called by the CPU, then the page information was transmitted to the GPU, and operates by the GPU. In the HSA system, GPU had the same status as CPU. When it generated the page fault exception, the missing page can be directly called by the GPU to memory, saving the cost of data transmission between CPU and GPU. It was an improvement of HSA on the GPU.

3.4 Context Switching

The context switch was to transfer the control of the CPU from a running task to another ready task. In the context switch, there were two important processes, which were the state preservation and the state recovery. When an interrupt was generated, a context switch was required, which needed to save the state of the current process to the PCB, including the value of the register, process status, and memory management information. This process was called state preservation. After the completion of the handover process was executed, the saved context started to run again. This process was called the state recovery. Context switches had three forms, they had different priorities:

1) Switch: the task that was being performed can continue to execute until the end, and then switch to another ready task. This switch was the lowest priority.

2) Pre-emption: it was hoped that the context switch can be implemented as soon as possible. Saving the data information and working state of the currently executing task, and then performing the handover, after the task execution was completed, the context of the task to resume was to continue to execute.

3) The end and context reset: execute context switch immediately. Did not save the state of the current task. When the switching task execution was completed, the task to be seized was to be executed. This switch priority was the highest.

4. Experiments and Analysis

4.1 Experimental platform based on Kaveri

In this project, we used HSA-supported central processor -- Kaveri, which was the new generation APU products. With the improvement on the function and the performance, Kaveri processor had excellent computing capabilities in the field of gaming and image processing prospects. Since calculating the lower workloads, this kind of design could not only ensure the performance, but also save power, improve the endurance of this architecture products

The basic steps of building a heterogeneous system based on the Kaveri APU were as follows:

(1)To prepare devices for HSA platform. In this project, we used AMD model for 7850K APU A10, ASUS A88X-PRO motherboard, ARC OCZ 100 (240GB) hard disk and (Kingston) Savage series DDR3 2400 memory.

(2) To verify if the system had never installed the graphics card driver. If the system had installed that, uninstall it.

(3)To install the suitable operating system. The Linux 3.19 kernel was supported by HSA, so the platform can be installed beyond this version. In this project we installed the ubuntu 15.04.

(4)To install the corresponding kernel driver. Since the kernel version of the operating system was 3.19.0, so we used KFD (version 1.2) as the kernel driver in this project.

(5)To install the runtime environment. In this project, we used the HSA-runtime (Version 1.0) which was provided by AMD.

(6)To install the compiler. In this project, we install cloc, which can translate the .cl file, written by OpenCL, into the HSAIL file which can be used by the HSA rules.

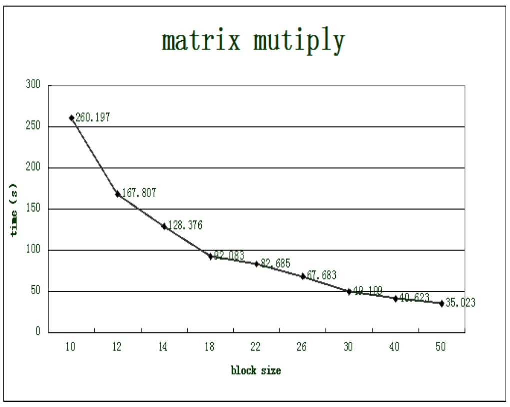
After the above steps, we built the basic development environment for HSA. And then, we could run some benchmark to verify the performance of this system.

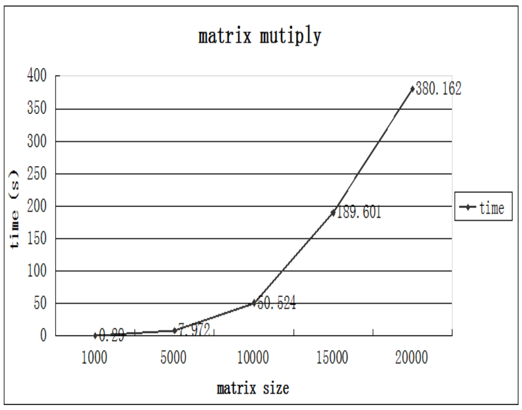
4.2 Results and Analysis

Up to now, the benchmarks, which can be executed on HSA platform, were still very inadequate, especially in aspect of quantitative, analysis and verification of the fine-grained. To get the goal of this paper, we changed the standard benchmarks into the tested programs which can be used on the HSA platform. And then, we used some monitor commands for Linux and some specific system monitoring tools, so that we can trace and profile the system changes. These monitor commands included top, uptime, and perf command. And the system monitoring tools included Systemtap for Linux and CodeXL by AMD.

For the single workload, the monitoring tools run twice to get all the performance parameter of the executing workload. At the first time, the “top” command was executed, and the command “perf stat -d” was gone, and then we can get the utilization of the CPU and memory, the number of the context switching and page fault interrupt. At last, the “uptime” command was used to get the system average loads. At the second time, the power profiling in CodeXL was on, the “time” command was used to get the executing time of the GPU, GPU utilization, the whole executing time and the time distribution on between user mode and the kernel mode. At last, the Systemtap tool was used to access the KFD and IOMMU function-call duration.

With the above process, we executed Matrix-vector multiplication as benchmark, which was the typical one in the area of the high performance computing challenge (HPCC). As Fig3a showed, with the matrix block size increased, the running time would reduce. And in this experiment, the whole matrix size was fixed on 10000 * 10000. As Fig3b showed, if the block size was fixed, with the whole matrix size increased, the running time would augment correspondingly.





(a) (b)

Figure 3: Matrix-vector Mutiplication Operation

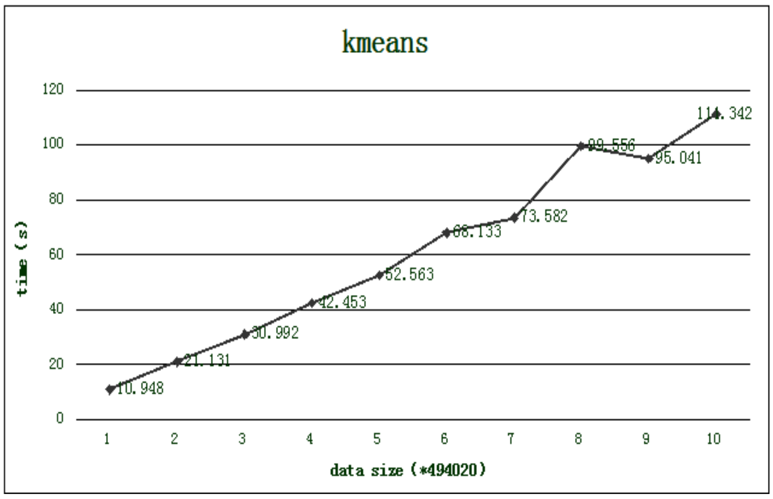


Figure4: K-means Operation

With the above process, we executed K-means as benchmark, which was the typical one in the area of the big data. As Fig4 showed, if the number of pointer was fixed, with the whole size increased, the running time would augment correspondingly.

5. Conclusion

Recently, the fused CPU-GPU architecture developed vey quickly. At the same time, the heterogeneous computing had been actively researched, which used in many kinds of applications. In this article, we had fused on the new feature -- Linux 3.19 kernel support to heterogeneous computing. After building the heterogeneous computing platform based on Kaveri APU, we executed two typical workload to verify the improvement on performance. The results showed that the HSA, supportted by Linux3.19, could provide better performance and energy.

References

[1] S. Mittal and J. S. Vetter. *A survey of CPU-GPU heterogeneous computing techniques. ACM Computing Survey*. 47, 4, Article 69 (July 2015), 35pages. DOI: http://dx.doi.org/10.1145/2788396 2015-10-20

[2] AMD Developer Central. *What is Heterogeneous System Architecture (HSA)?* http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-system-architecture-hsa/ 2015-12-07

[3] Wikipedia. *APU.* https://en.wikipedia.org/wiki/AMD_Accelerated_Processing_Unit. 2015-11-2

[4] M. Larabel. *AMDKFD Is Present For Linux 3.19 In Open-Source HSA Start.* http://www.phoronix.com/scan.php?page=news_item&px=MTg1MzE 2014-12-3

[5] M. C. Delorme, T. S. Abdelrahman, and C. Zhao. *Parallel Radix Sort on the AMD Fusion Accelerated Processing Unit.* 2013 42nd International Conference on Parallel Processing, IEEE Computer Society, 2013:339-348.

[6] T. H. Hetherington, T. G. Rogers, L. Hsu, M. O'Connor, and T. M. Aamodt. *Characterizing and evaluating a key-value store application on heterogeneous CPU-GPU systems,* In Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'12). 88–98.

[7] K. Spafford, J. Meredith, and J. Vetter. *Maestro: Data orchestration and tuning for OpenCL devices*. In Euro-Par 2010-Parallel Processing, Lecture Notes in Computer Science, Vol. 6272. Springer, Berlin, 275–286.

[8] K. Lee, H. Lin, and W. Feng. *Performance characterization of data-intensive kernels on AMD fusion architectures*. Computer Science—Research and Development 28, 2–3 (May 2013), 175–184.

[9] Y. Ukidave, A. K. Ziabari, P. Mistry, G. Schirner, and D. Kaeli. *Quantifying the energy efficiency of FFT on heterogeneous platforms*. In Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'13). 235–244.