A Dynamically Adaptable Mechanism for PCIe-based Accelerators and Storage Systems

Eleni Bougioukou, Maria Varsamou and Theodore Antonakopoulos
Department of Electrical and Computer Engineering
University of Patras
Patras, 26504, Greece
e-mails: *bougioukou@upatras.gr*, *mtvars@upatras.gr* and *antonako@upatras.gr*

1. Introduction

Building high performance computing and storage systems using low-cost, off-the-shelf components is challenging and it can be exploited in various commercial, enterprise and scientific applications. Such systems are usually configured using a basic interconnect technology along with general-purpose (ie. GPUs, SSDs) and reprogrammable (ie. FPGAs) boards with custom functionality, integrating application-specific hardware or/and software modules, while direct communication or shared memory is used for data exchange. Solid-State Drives (SSDs) use non-volatile memories for storing and retrieving information in the form of sectors and/or pages and achieve access rates and response times much better than hard disk drives. Among the most well established interconnect technologies is PCIe (Peripheral Component Interconnect express) that is used in various systems, from personal computers up to enterprise servers and data center installations. PCIe bus is a serial interface that utilizes multiple lanes in parallel and achieves data rates higher than 1GB/s, whereas SATA rev. 3 can offer data speeds of approximately 600 MB/s.

In this work we present a dynamically adaptable data exchange mechanism for shared memory PCIe systems. The proposed mechanism is characterized as dynamically adaptable, since it adapts its functional parameters according to the application needs as they are expressed with the applied workloads, and that results to minimum latency when light workloads are applied, while the I/O performance is maximized when heavy workloads have to be serviced. The proposed data exchange mechanism can also be parameterized in order to achieve the best compromise between minimum latency and maximum I/O performance, according to the application specific needs.

2. PCIe Devices and Device Drivers

The increased complexity of today's commercial, enterprise or scientific applications, along with the huge processing power and very high data rates, has made necessary the use of accelerators for offloading the host CPU/memory system. To ensure maximum performance, these devices have to be connected to the host through high-speed I/O interfaces. The most commonly used I/O interface in today's motherboards is the full-duplex high-speed PCIe interface. PCIe devices can access the system host memory without CPU intervention, by performing direct DMA read/write memory transactions. On the other hand, user applications in the host computer can access and manage the PCIe devices only through operating system I/O calls to dedicated device drivers. Therefore, the maximum performance that can be achieved by a PCIe device depends on the raw physical data rates but also by the efficiency of the host-device I/O interface, ie. the transactions between the device driver and the device controller.

Usually, I/O data transfers are performed using a structure of descriptors located in a memory area, shared by both the host CPU and the PCIe device. Each descriptor contains the physical address and the size of the data to be transferred. A transaction is initiated by setting up the proper descriptors and sending an indication to the device controller. Then, the descriptors are transferred to the device's local memory, where the controller processes them, issues the respective DMA read/write operations from/to the host memory and executes the application specific data processing functions. When the descriptors have been served, they are returned to the host, including information regarding the completion status. This host-initiated descriptor-based approach can lead to idle execution phases, which basically results to under utilization of the system's capabilities [1].

The performance of I/O devices is mainly characterized by two performance metrics, I/O throughput and end-to-end latency. Depending on the application, the requirements regarding these two parameters may be different or depending on the applied workload. Due to the potential applications, the I/O workloads that access the device driver may

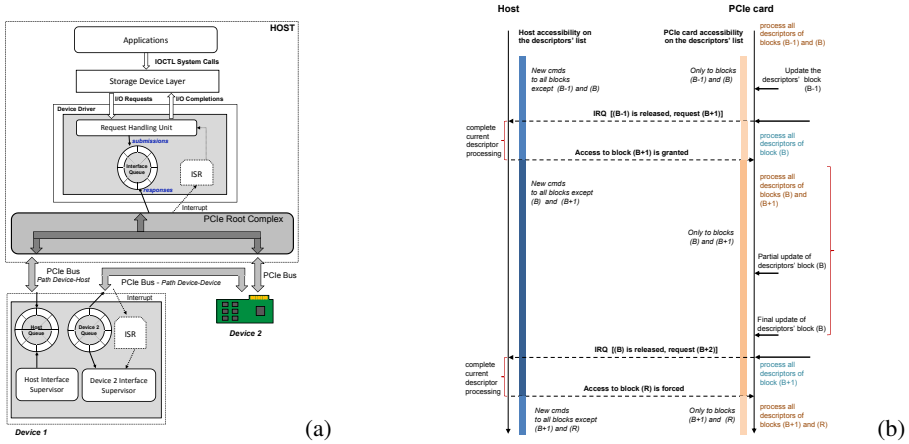

Figure 1. Architecture of the dynamically adaptable PCIe interface and the exchange of descriptors' blocks.

have different characteristics, e.g. frequent I/O commands of small data sizes, infrequent ones with large data sizes or a mixture of all these types.

3. Dynamically Adaptable PCIe Interface

In this work we present a descriptor-based PCIe interface that is dynamically adaptable to the offered workload. The architecture of the proposed dynamically adaptable PCIe interface is shown in Fig. 2a. The external device is connected as a PCIe endpoint on the PCIe root complex of the host motherboard. The device driver is a block device driver, compatible with the Linux I/O stack. The I/O requests that are issued by the user applications are processed in a First-Come First-Served order and are dispatched to the PCIe device through a list of descriptors that are dynamically manipulated using a circular buffer in the shared host memory space. The interface includes also a set of registers in the PCIe address space and an interrupt mechanism. Since the PCIe specification allows the PCIe endpoints to communicate with each other without host intervention, the proposed interface can be also used for direct device-to-device data exchange. This way complex systems with dedicated hardware accelerators and/or high-performance processing units, such as GPUs, can be built for supporting specific application needs.

Each host/device-to-device interface is serviced using a dedicated dispatch/response queue of descriptors. The descriptors are sent to the PCIe devices in blocks of variable sizes, where the maximum block size is specified by the processing capabilities of the devices. The device driver is constantly evaluating the offered load by observing the pending requests in its memory space and adjusts the block size accordingly. As the offered load increases, the descriptors' block size also increases, and higher I/O rate is achieved, while when the workload decreases, the descriptors' block size decreases as well, ensuring better latency. The PCIe device processes each block of descriptors and when all descriptors have been serviced, it sends the updated descriptors back to the host and issues an interrupt to inform the host CPU of the completion. The device driver is responsible for preserving the order of the completed commands to ensure data consistency with the application layers at the user level. The interface synchronization between the various devices, regarding the shared list of descriptors, is explained with the timing diagram shown in Fig. 2b. In a typical system, sequential block processing results to loss of processing power, since idle time occurs between issuing an interrupt and information update of the next block to be processed. Delays in the interrupt dispatching by the host, along with OS delays due to context switching, can lead to exceptionally big idle execution times despite the fact that the device driver may have already prepared the subsequent blocks of descriptors in host memory space. This can lead to a performance decrease. For confronting this drawback, we propose an enhancement to this interface with a smart forward command mechanism, where the driver not only sends the current block of descriptors but also informs the device regarding the pending descriptors, independent to the size of the next block. This way, when the device returns a serviced block of descriptors, it already has a number of the next descriptors in its local memory and starts processing them. As a result the device remains constantly active and the maximum performance is achieved. In the oral presentation we will present experimental results of the application of this mechanism to a high-performance storage systems using custom SSDs.

References

[1] Steen Larsen and Ben Lee. Chapter Two - Survey on System I/O Hardware Transactions and Impact on Latency, Throughput, and Other Factors. Advances in Computers. Elsevier, 2014.