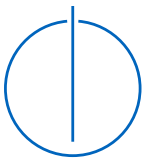


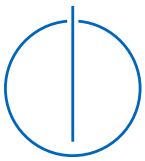
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Game Engineering

Thesis title

Ruilin Qi



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Game Engineering

Thesis title

Titel der Abschlussarbeit

Author:	Ruilin Qi
Supervisor:	Supervisor
Advisor:	Stepan Vanecek
Submission Date:	15th March 2022

I confirm that this bachelor's thesis in informatics: game engineering is my own work and I have documented all sources and material used.

Munich, 15th March 2022 Ruilin Qi

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Background and Motivation

1.2 Goals and Aims

- develop lightweight tool that has few requirements to gain insight to data movements in a PCIe link

1.3 Delimitation

- Heterogeneous systems have many different interconnects - Focus on PCIE-interface in this thesis - (why)? not sure yet, figure something out

1.4 Structure and Approach

- three approaches / benchmarks - bandwidth: for measuring the raw bandwidth capacity of the system - nvml: for measuring pcie link activity - copy: for measuring pcie link activity

2 Background

2.1 HPC-node

- consists of CPU, RAM, GPU, storage, etc. - cpu has multiple cores and threads - in short: a PC

2.2 PCI-Express

PCIe, or PCI-Express, shorthand for Peripheral Component Interconnect Express, is a "general-purpose serial I/O interconnect". [cite pciefaq] PCIe, as an interface, allows the CPU to connect with, as the name suggests, peripherals and components. [cite: pcmag] Common components and peripherals include, but are not limited to: Graphics cards, sound cards, video capture cards, WiFi cards, and storage. [cite: HP] PCIe is designed to replace the ageing PCI (Peripheral Component Interconnect), PCI-X (Peripheral Component Interconnect Extended), and AGP (Accelerated Graphics Port) standards. [cite here: verma/dahiya] These standards are developed, defined, and maintained by the PCI-SIG group, which is a nonprofit organization with 800+ member companies based in Beaverton, Oregon. [cite here: pcisig page] This chapter will briefly introduce the key features and functionality of PCI-Express.

2.2.1 Key Features

PCI-Express is, at its core, a serialized, point-to-point connection that is designed to be processor agnostic, scalable, and backwards compatible with PCI.[cite: lawley] [cite: pciefaq] [cite: pcisig] PCI-Express utilizes a dual-simplex connection to facilitate sending and receiving information concurrently. Additionally, to ensure backwards compatibility with PCI, PCI-Express shares the same memory configuration as PCI, which will be elaborated further upon in section [ref: section]. Further key features include better error handling and data integrity capabilities. [verma] To future-proof the standard, future generations of PCIe are to be designed to be compatible with current PCIe standards. [pcisig site] So far, each generation of PCIe doubled the previous generation's theoretical maximum bandwidth, as seen in table [ref: table]

[table here]

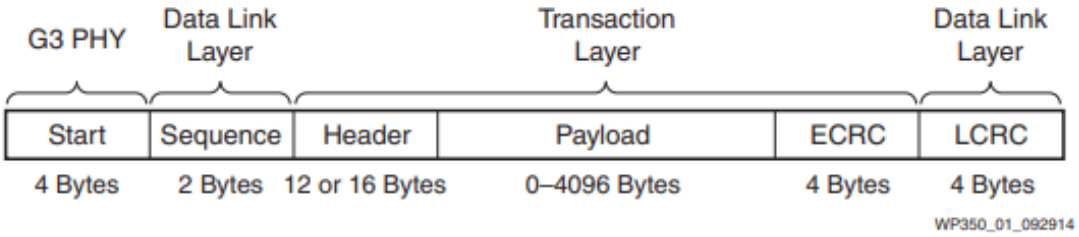


Figure 2.1: An example of a PCI-Express packet [cite source]

2.2.2 Functionality

packet

PCIe, similar to IPv4 or IPv6, utilizes packets to communicate between the host - the CPU - and the device. As shown in figure 2.1, the packet consists of a few different elements, which will be further expanded upon below.

- Start: this is the start component which signals the begin of a packet to the physical layer.
- Sequence: This two-byte sequence is used by the Data Link Layer to determine the sequence of the packets.
- Header: The 12 to 16 Byte header will be discussed in further detail in subsection [reference to header]. This component belongs to the Transaction layer.
- Payload: The PCIe payload. This is optional, however any memory transferred via memory copy operations will have the memory as payload. This also is a part of the Transaction layer.
- ECRC: a CRC code for error-checking purposes used by the transaction layer.
- LCRC: a CRC code for error-checking purposes used by the Data Link Layer.

header

As with IPv4 or IPv6, PCI-Express uses headers to determine the purpose and target of each TLP (Transaction Layer Packet) However, instead of using IP-addresses, stored in the header, to determine the sender and the receiver, PCIe uses the Requester ID to determine the sender. The Address determines the receiver of the intended packet, as the device memory is memory-mapped into the host address domain to enable the

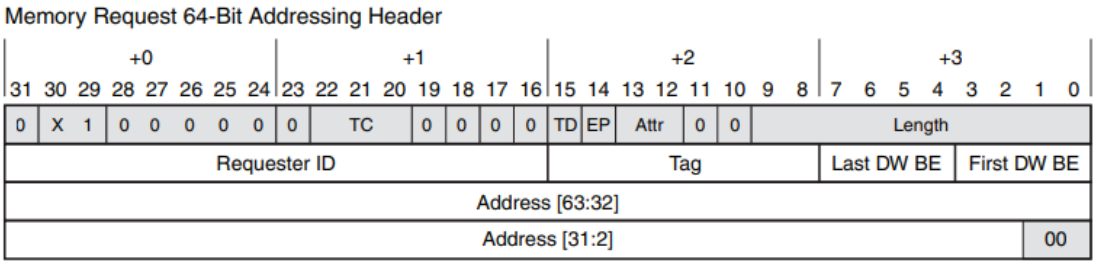


Figure 2.2: An example of a memory request header [cite source]

processor's native load or store instructions to work with PCIe devices. [cite oracle website] As seen in Figure 2.2, the header has a fixed format, similar to an IPv4 or v6 header. The fields and their uses are briefly explained below.

- TC: Traffic Class: this denotes the priority of the packet. A larger value represents a higher priority. [cite book]
- TD: The TLP Digest field. If TD is set to 1, it indicates that there is additional CRC data in the TLP data. [cite xillybus]
- Length: more or less self-explanatory: length denotes the length of the payload in Double Words. [cite xillybus]
- Requester ID: self-explanatory: the ID of the device that requested or sent the packet. [cite xillybus]
- Tag: The Tag field has the function of a tracking number, as for read requests, the device must copy this value to its response. All outstanding tags must be unique to ensure data integrity. [cite xillybus]
- DW BE fields: DW BE stands for Double-Word Byte Enable. This denotes which of the bytes in the first / last DWs are valid. [cite xillybus]
- Address: self-explanatory: The Address to which this packet is addressed, as explained above. Additionally, for read and write requests, this denotes the starting address of the read or write. [cite xillybus]
- The EP and Attr fields are not further elaborated upon as they are rarely used by PCIe endpoint devices. [cite xillybus]

[note: a lot of the specifications are unaccessible to me due to them being locked behind PCI SIG membership, so sources are difficult to find.] [note2: xillybus is a

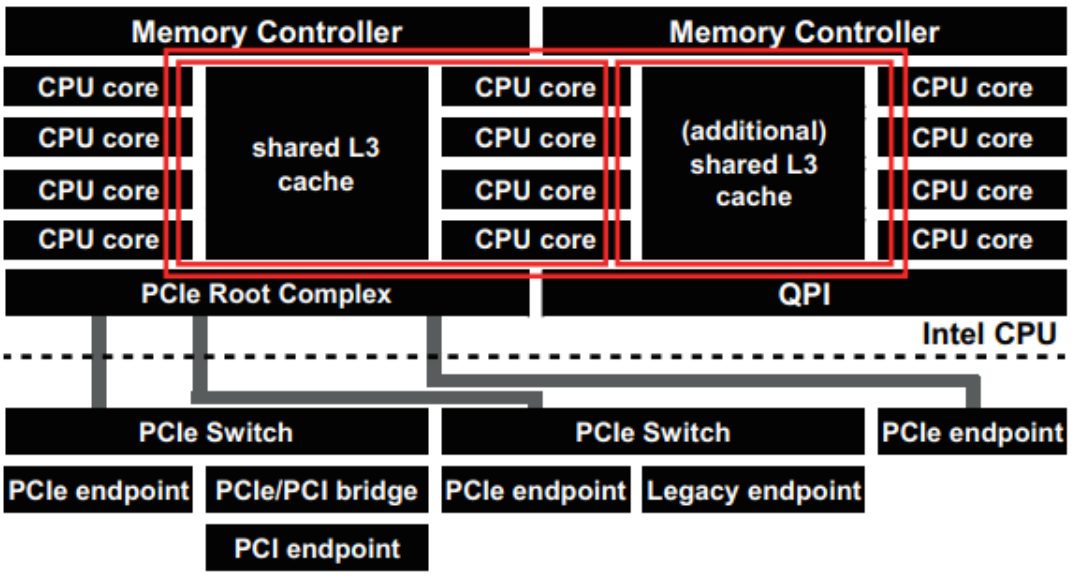


Figure 2.3: PCIe configuration on an Intel-based system [cite source]

website source, not sure if it is safe to use. Link: http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1]

2.2.3 Topology and Communication

Topology

There are four significant components to be mentioned when discussing the topology of a PCI-Express based system. PCIe endpoints, switches, bridges, and a root complex. The communication between CPU cores and memory controllers to the PCIe endpoint is handled by the PCIe root complex. This communication can be routed through (but does not require) PCIe switches. PCIe switches allow for cascading connections, however do not benefit the total bandwidth, which is limited by the PCIe root complex in a CPU. [cite: nakamura et al] Bridges are used to connect legacy PCI and PCI-X devices with the PCIe root complex. [cite: pciefaq] Figure 2.3 shows an example PCIe configuration of an Intel-based processor.

Memory Management

The PCIe memory structure is, due to compatibility reasons, the same as the memory structure found in the older PCI standard. This divides the device memory into three

major parts for addressing: [cite: book]

- Configuration
- Memory
- IO

The configuration address space enables software to both identify and correctly configure the device, and is defined by its bus number and device number. [cite: oracle] The memory address space is where the storage and registers of a PCIe device is mapped. [cite: book] This memory space is also memory-mapped to the host address domain for ease of access by the CPU. [cite: oracle] The IO address space is a place dedicated to accessing the internal registers / storage of a PCIe or PCI device. However, this is mostly deprecated in PCIe as the internal registers and storage of said devices are simply mapped into the memory address space instead. It is now common practice to map the same set of registers in both memory and IO address space. The PCIe specification discourages use of the IO space, which indicates that it remains here solely for legacy support purposes. [cite: book]

Links and Lanes

- lane, link, link width (graphic)

2.2.4 Revisions and Further Specifications

- mention gen 5 and 6
- gen6 paper
- other formfactors: thunderbolt / nvm-express

2.3 Graphics Processing Units

2.3.1 What are GPUs

To begin with, it should be noted that the term graphics processing unit (GPU) does not equate to a graphics card. A GPU is a specialized processing unit primarily designed for parallel processing and accelerating workloads that require parallel processing. [cite: intel] A graphics card, on the other hand, is the add-in card that features a PCI-Express link to facilitate communication between CPU and GPU, dedicated memory and power delivery for the GPU, and the GPU itself. It should also be noted that the graphics

card usually has its own dedicated PCB. There are also integrated GPUs, which can be embedded alongside the CPU. These integrated GPUs are usually less powerful compared to discrete GPUs. [cite: intel]

2.3.2 Uses of GPUs

GPUs originally began as, as their names suggest, dedicated graphics accelerators

- In the past: only for real-time 3D graphics
- now: more or less general-purpose GPUs as accelerators for several different use-cases and workloads
- Gaming / Video Editing / Content Creation
- Machine Learning: Tensor cores [nvidia]
- raw compute performance for single-precision [arxiv paper]
- AI and HPC tasks here, do further research (?)

2.3.3 GPU Memory

- same addressing as other pcie endpoints [ref section] - faster VRAM and higher bandwidths compared to main memory -

2.4 CUDA

CUDA is a closed source API developed and maintained by Nvidia for use with their GPUs and graphics cards.

2.4.1 Features

- nvidia's programming-API
- closed-source
- offers way for the CPU to communicate with and to program nvidia GPUs
- use the .cu extension, called kernels (why? not sure.)
- features: memory management, data movement, etc. -> subsection?

- more features: Libraries for several different features such as linear algebra, signal processing, image processing [cuda11 features nvidia]
- CUDA is the preferred API for this thesis (reasons?)

3 Bandwidth Benchmark

3.1 Concept

- Measures raw theoretical maximum bandwidth of pcie link by measuring the duration of memory copies of various chunk sizes - Checks at which chunk sizes the bandwidth of the link is fully saturated

3.2 Implementation

- Compensates for delay - 1 packet with 4B measured as delay - Pageable and pinned memory benchmarks measured - Warmup-feature: first transfer usually has some sort of longer delay, compensates for that (windows-finding, verify on p6000)

3.3 Results

- Transfer durations don't really increase until 8kb - Due to the nature of the PCI-E packet having a max payload of 4kb - First transfer usually has a bit longer delay (warmup?) - On windows: not executable that calls functions, but rather nvcuda64.dll - requires compiling on windows and then using a profiling tool like AMDUprof to look at the program

3.4 Discussion

3.4.1 successes

- gives accurate reading of pcie bandwidths - non-linear scaling of packet transfer durations (2 packets does not equal double the duration of one packet)

3.4.2 shortcomings

- does not really compensate for other bottlenecks, as seen on time-x with gen4 link bandwidth speeds

4 NVML Counters

4.1 Concept

- nvidia has hardware counters, accessible via nvml library - counters measure average bandwidth over the last 20ms, in kb/sec - Transmit and Receive have seperate counters

4.2 Implementation

- method call to read counters takes about 20ms - to increase data granularity and measurement consistency, measuring of TX and RX was done in parallel

4.3 Results

- graphs

4.4 Discussion

4.4.1 successes

- measures bandwidths accurately to some degree - introduces little overhead (probably, still to be measured)

4.4.2 shortcomings

- granularity of method calls prevent more accurate readings -> short memory transfers may be not detected - black box approach of nvidia's source code doesn't allow for proper sanity-checking

5 Link Saturation

5.1 Concept

- if bandwidth is saturated, copy operations should slow down - full duplex, so HtoD and DtoH both need measuring

5.2 Implementation

- Started as a thread that just continuously monitored the counters for set duration of time and printed output into console - Added wrapper and file output in subsequent versions to simplify data-gathering - Added chunk size options for the buffer copy chunks to get as little overhead as possible while getting most consistent data gathering - Delays are measured, no clear correlation between delay and bandwidth - Overhead yet to be properly assessed, however, introduces somewhat significant overhead. TODO: ASSESS OVERHEAD - Measure transmit and receive in the same thread, sequentially

5.3 Results

- graphs - descriptors of graphs

5.4 Discussion

5.4.1 successes

- gives somewhat detailed going-on about PCIe link activities

5.4.2 shortcomings

- introduces some overhead due to occupying PCIe link - Delay compensation sometimes leads to negative values due to delay inconsistencies

6 Summary

Outlook

List of Figures

List of Tables