UTILIZING GPUDIRECT 3RD PARTY DMA FEATURES FOR 10GbE NIC AND GPU APPLICATIONS

Presented by Colin Shea and Tim Gramiccioni
Ventura Solutions, Inc.
Overview

• Acronyms
• Project Goals
• Previous Solutions
• What is GPUDirect 3rd Party
• GPUDirect 3rd Party Implementation
• Test Setup
• Results
• Future work
• Acknowledgments
• Questions
Acronyms

- HPC - High Performance Computing
- IF – Intermediate Frequency
- VITA49 – Standardized transport for digitized IF data samples
- DMA – Direct Memory Access
- RDMA – Remote Direct Memory Access
- MTU – Maximum Transmission Unit
Project Goals

• Goals
  • Transfer Ethernet packets directly from 10GbE NIC device to GPU memory
    • PCIe Write Through
    • Skip system memory and CPU intervention
  • Code modification is universal and can be applied to nearly any network device
    • Modified goal to target Myricom 10GbE network card
Previous Solutions

• Before GPUDirect v3
• Method
  • Myricom driver creates pinned memory for a ring buffer
  • GPU creates pinned memory for data
  • Allow system call recv() to get data from socket
  • Written to GPU pinned memory
  • Copy pinned memory to GPU
What is GPUDirect 3rd Party DMA

- NVIDIA refers to moving data between two PCIe devices generically as RDMA (Remote DMA)
What is GPUDirect 3rd Party DMA

- In the HPC sector RDMA refers to a transport protocol for moving data between compute nodes
  - Moving data from one computer’s memory to another without involving the operating system.
  - Enables high-throughput low-latency networking
What is GPUDirect 3rd Party DMA

- Our solution refers only to moving data from one PCIe device to another PCIe device
  - Allows for direct read/write access from device one’s RAM to the GPU’s RAM
    - This is on the same host and on the same PCIe root complex
  - As we are not utilizing the network for moving data we are referring to this method as GPUDirect 3rd Party DMA
What is GPUDirect 3rd Party DMA

• Outlined from NVIDIA's white paper “DEVELOPING A LINUX KERNEL MODULE USING RDMA FOR GPUDIRECT” (Traditional DMA)
  • User space program requests a transfer via the user space communication library.
    • This operation takes a pointer to data (a virtual address) and a size in bytes.
  • The communication library checks memory region, corresponding to the virtual address and size, is ready.
  • The kernel driver receives the virtual address and size from the user space communication library.
What is GPUDirect 3rd Party DMA

• Outlined from NVIDIA's white paper “DEVELOPING A LINUX KERNEL MODULE USING RDMA FOR GPUDIRECT” (Traditional DMA cont.)
  • It then asks the kernel to translate the virtual address range to a list of physical pages
    • Referred to as pinning the memory.
  • The kernel driver uses the list of pages to program the physical device's DMA engine(s).
  • The communication library initiates the transfer.
  • After the transfer the communication library cleans up any resources used to pin the memory.
    • Referred to as unpinning the memory.
• Our implementation
  • Application creates a series of GPU memory buffers
    • Buffers are in Unified Virtual Address Space
  • Call to cuPointerGetAttribute() in which the p2ptokens are returned
    • Pertain to memory inside a specific CUDA context
  • Pass the data from the user space to the kernel space
    • Chose IOCTL to communicate between user and kernel space
    • The base address (UVA pointer)
    • The corresponding p2ptokens for the base address
    • The size of the allocated memory
  • Translate UVA pointers to GPU Physical addresses
    • Held in the NIVIDA driver’s page tables
    • Calls are NVIDIA driver API calls (nvidia_p2p_)
    • Returns pointer to page tables
      • physical addresses can be found in a list of pages within this region
  • Physical addresses are passed to NIC hardware for DMA transfer
GPUDirect 3rd Party DMA Implementation

• 3rd party driver needs to:
  • Include nv-p2p.h to reference nvidia_p2p_* calls
  • Incorporate a method to receive the following from user space
    • The base address (UVA pointer)
    • The corresponding p2ptokens for the base address
    • The size of the allocated memory
  • Link with the NVIDIA driver to provide the required symbol mappings
    • nvidia_p2p_get_pages()
    • nvidia_p2p_put_pages()
    • nvidia_p2p_free_page_table()
    • Generic unpin call back
• Challenges with the Myricom driver
  • Myricom does not use get_user_pages paradigm
    • Creates an explicit ring buffer
      • There are actually two different ring buffers for data
        • One is for packets up to 4KB
        • The second is for packets greater than 4KB
      • Each ring buffer element is a receive buffer descriptor
    • Interrupt Control Buffer is established
  • Had to remap all data target memory structures
    • Physical pointers to buffers created in GPU memory
    • Implemented a “lazy unpinning” approach where the memory buffers are reused over and over again
GPUDirect 3rd Party DMA Implementation

- Challenges with the Myricom driver continued
  - Different page sizes for the Kepler and the Myricom device
    - Myricom by default expects 4KB pages
    - Kepler has default 64KB pages
  - Mapped 64KB page table entries into 16 4KB entries
    - Otherwise only 4KB worth of page data in a 64KB page
**Test Setup**

- We only implemented and tested the receive functionality
  - Only for small, 4KB, packet buffer
- Centos 6.4 64bit
- IOMMU is disabled at the kernel boot
- Devices are on the same PCIe root complex
- Using NVIDIA driver included in CUDA 5.0 install
- Using Myricom 10G-PCIE-8B-2S+E 10GbE network card
- Chassis is a Supermicro 7047GR-TPRF 4U
  - Dual Xeon E2630
  - 32GB of DDR3 per Socket
- NVIDIA GTK110 – K20 PCIe v2.0
- FPGA based sample packet generator
  - Allows for control of network traffic generation rate
Test Setup

- **Experiment setup**
  - **Step 1 – Copy packets to GPU**
    - NIC hardware verifies UDP packet checksum
  - **Step 2 – GPU operation:**
    - Strip UDP/VITA packet header
    - Checks packet counter for packet loss
Results

- Two memory copy
  - Pointer pinned memory with system call recv()
    - 400MB/s – 4MB buffer size
- GPUDirect Method using 10GbE PCIe v1.0 card
  - Max rate PCIe V1.0 is 250MBs per lane
    - 8 lane = 8*250MB/s or 2000MB/s
  - Myricom advertises 9.5Gb/s at MTU of 1500
    - 9.5Gb/s = 1132MB/s < 2000MB/s max
  - Ingest rate at GPU is currently 6.5Gb/s or 774MB/s with kernel application running
Future Work

• Improve DMA performance to equal line rate
• Produce a polished version of read from NIC
• Implement the write back to the NIC
• Expand the work to different NICs
• Implement the large packet buffer functionality
• Scatter/gather transactions for multiple GPUs to a single NIC
• Integration with Redhawk SDR framework
  • redhawksdr.org
Acknowledgments

• We would like to thank the following for their help
  • NVIDIA
    • Kevin Berce
    • Tom Reed
  • Ventura Solutions Inc
    • Adam Anderson
    • Ross Levin
    • Kaitlynn Elliott
Questions?

• Who is Ventura Solutions Inc.
  • Mid sized company in Annapolis Junction MD
  • Started in 2004
  • Engineering Services
  • Staffing skill sets
    • Software Engineers
    • Hardware Engineers
    • System Engineers
  • Primary customers
    • Department of Defense
    • Commercial
• Find out More at www.vsi-corp.com
• Developing a Linux Kernel Module using RDMA for GPUDirect – NVIDIA Application Guide v5.0
• Dynamic DMA mapping – D Miller, R Henderson, J Jelinek
• Linux Device Drivers - Corbet, Rubini, Kroah-Hartman O’Reily Press 2005
• Ethernet Device Driver Software Interface for Myricom Myri-10G Network Interface Cards
• CUDA C Programming Guide
• 10G-PCIE-8B-S Single-Port 10-Gigabit Ethernet Network Adapters Data Sheet
This Page is Intentionally Blank
• Receive Buffer Descriptor
  • Describes the pointer addresses (start/end) of free main memory buffers that the NIC can use to store received packets

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset (B)</th>
<th>Size (B)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSW Address</td>
<td>0</td>
<td>4</td>
<td>MSW of the start address of Receive Buffer.</td>
</tr>
<tr>
<td>LSW Address</td>
<td>4</td>
<td>4</td>
<td>LSW of the start address of Receive Buffer.</td>
</tr>
</tbody>
</table>

*Table 7: Receive Buffer Descriptor.*
Myricom Memory Mappings?

- Two different memory Receive Buffer Rings
  - Small buffer for packets up to 4KB
  - Targeted to MTU of 1500
  - Large buffer for packets larger than 4KB and power of 2

<table>
<thead>
<tr>
<th>Da...Db</th>
<th>Invalid Receive Buffer Descriptors. LSW Address contains all 1s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Di...Dj</td>
<td>Valid Receive Buffer Descriptors that contain the locations of free Receive Buffers that the NIC may use to store received packets. Note that Receive Buffers are 4B aligned, so valid Descriptors cannot have all 1s.</td>
</tr>
<tr>
<td>De...Df</td>
<td>Invalid Receive Buffer Descriptors. LSW Address contains all 1s.</td>
</tr>
</tbody>
</table>

*Figure 7: Receive Buffer Ring.*