Hippogriff: Efficiently Moving Data in Heterogeneous Computing Systems

Yang Liu, Hung-Wei Tseng*, Mark Gahagan, Jing Li, Yanqin Jin, Steven Swanson
Department of Computer Science and Engineering
University of California, San Diego, La Jolla, CA, U.S.A.
{yal036, h1tseng, mgahagan, jil261, y7jin, swanson}@cs.ucsd.edu

Abstract—**Data movement between the compute and the storage (e.g., GPU and SSD) has been a long-neglected problem in heterogeneous systems, while the inefficiency in existing systems does cause significant loss in both performance and energy efficiency. This paper presents Hippogriff to provide a high-level programming model to simplify data movement between the compute and the storage, and to dynamically schedule data transfers based on system load. By eliminating unnecessary data movement, Hippogriff can speedup single program workloads by 1.17×, and save 17% energy. For multi-program workloads, Hippogriff shows 1.25× speedup. Hippogriff also improves the performance of a GPU-based MapReduce framework by 27%.**

I. Introduction

To provide sustainable high performance, a heterogeneous system needs both high compute power and efficient data movement. Existing systems only partially achieve this with accelerators (e.g., GPUs) and PCIe-based peer-to-peer communication only among compute devices. When it comes to data transfer between the compute and the storage, however, the existing systems still stick to the entrenched CPU-centric programming and execution model. This used be less of a problem because buffering the data in the host DRAM is necessary, given the moderate data sizes and the slow hard drive disks. But with the rapidly growing data sizes and much faster PCIe-based SSDs nowadays, the traditional way brings much overhead.

To improve the performance and attain the more complete peer-to-peer data transfer in heterogeneous systems, we present *Hippogriff*. Hippogriff first provides a subsystem called *NVMeDirect* to extend new NVM-Express (NVMe) [2] interface. Thus, Hippogriff can support both the direct peer-to-peer transfers between the SSDs and the GPUs, and the conventional way of sending data via the host. Nevertherless, it would still be challenging for users to choose the optimal data transfer route from the two for good performance. So Hippogriff further provides a set of simplified interfaces with the Hippogriff runtime system to automatically select the most efficient data tranfser route.

We evaluated the Hippogriff system on a set of comprehensive GPU benchmarks as well as a GPU MapReduce framework. Experiments show that for single programs, Hippogriff can achieve a speedup of 1.17×, and improve the energy consumption by 17%. Hippogriff can further accelerate multi-program applications by 19%–31% on average, and improve the performance of a highly-optimized GPU-MapReduce framework by 12%.

*Current Affiliation is Department of Computer Science, North Carolina State University.

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

II. Moving Data in Heterogeneous Systems

This section provides a brief introduction to the two types of data transfer between the storage devices and the computing resources in heterogeneous systems.

A. Conventional CPU-centric Data Transfer

When it comes to data movement between the storage and the compute, the existing programming models for heterogeneous systems rely on the CPU as both the control plane and data plane. For example, when an application needs a data transfer from the SSD to the GPU, the SSD's driver has to first transfer the data to a DRAM buffer via DMA, and then the GPU driver delivers the data to the GPU also via DMA.

The obvious problem is, although the application just needs the data transferred from the SSD to the GPU, the actual data transfer has to go through the DRAM buffer first. This inefficiency wastes CPU time and memory bandwidth, pollutes TLBs and caches, and consumes DRAM unnecessarily.

B. Peer-to-Peer Data Transfer

PCIe allows devices to send data packets directly from one to another, bypassing the CPU and the main memory. Supporting PCIe peer-to-peer data transfer model requires a device to map its memory to the base address registers (BARs) in the PCIe controller/switch. AMD's DirectGMA [1] and NVIDIA's GPUDirect [10] are technologies that allows the software to program GPUs for PCIe peer-to-peer communication.

However, the standard NVMe SSD cannot natively support peer-to-peer data transfer, as NVMe SSDs are not byte-addressable but use block addresses for their own data array.

III. HIPPOGRIFF

Hippogriff decouples the control plane from the data plane for applications, and manages the low-level details of scheduling and executing data transfers. By eliminating the main memory from the data path with NVMeDirect, Hippogriff can reduce the data transfer latency, and free up the CPU and the main memory for other tasks. Figure 1 presents the three main components of the Hippogriff system.

Synopsis	Description
`int hippogriff_init()`	The `hippogriff_init()` function initializes the Hippogriff runtime.
`size_t hippogriff_send(int fd, void **gpuMemPtr, size_t offset, size_t size)`	The `hippogriff_send()` function creates the Hippogriff task of sending data from file descriptor `fd` with `offset` to GPU memory location `gpuMemPtr`. If the user passes a `gpuMemPtr` containing *NULL*, Hippogriff automatically allocates the GPU memory space fits the input file or requested size.
`size_t hippogriff_recv(int fd, void *gpuMemPtr, size_t offset, size_t size)`	The `hippogriff_recv()` function creates the Hippogriff task of sending `size` bytes of data from GPU memory location `gpuMemPtr` to the file that `fd` links to with `offset`.
`int hippogriff_deinit()`	This function releases the resource that Hippogriff uses for an application.

TABLE I: The Hippogriff API.

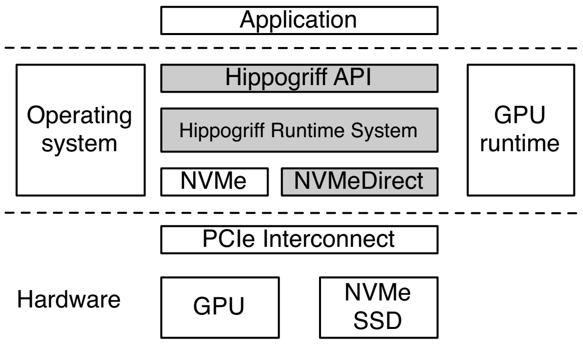

Fig. 1: The system components of Hippogriff

A. Hippogriff API

The Hippogriff API allows applications to just specify the sources and the destinations of data transfers, while leaving the actual data movement to Hippogriff, thus simplying the resulting code. Applications can use the API to initialize the environment, create data access requests, perform data transfers, and release the resources when the transfer is done.

The Hippogriff API interacts with the file system, operating systems, and the Hippogriff kernel module to acquire permission for accessing files. Table I documents the API.

B. Hippogriff Runtime System

The Hippogriff runtime system accepts the data transfer requests from the API, and schedules them to the optimal data transfer route. Between the two routes that the current Hippogriff runtime supports, the run time always favors the direct data transfer enabled by NVMeDirect. However, when the GPU does not have enough available memory or is under heavy load, Hippogriff can schedule the data transfer to the main memory buffer first before the GPU is ready. This results in lower overall latency and higher bandwidth between the GPU and main memory, compared to the SSD bandwidth.

C. NVMeDirect

NVMeDirect extends the NVMe interface with new `ioctl`-based read and write commands to move data directly between SSDs and GPUs. As an NVMe SSD is not byte-addressable memory (but the GPU is), the NVMeDirect driver together with GPUDirect or DirectGMA prepares GPU memory for the PCIe peer-to-peer transfers and generates NVMe read and write commands that use GPU memory as DMA targets.

When the SSD receives the command, it reads or writes data directly from or to the GPU without the involvement of the CPU or the main memory. Because NVMeDirect still relies on the CPU code to issue read/write commands, NVMeDirect does not incur any new file system integrity issues.

	Input Size		
Application Name	Small	Medium	Large
Breadth-First Search (BFS)	37 MB	587 MB	1.17 GB
Computational Fluid Dynamics (CFD)	16 MB	570 MB	713 MB
2D Discrete Wavelet Transform (DWT2D)	7 MB	27 MB	106 MB
Gaussian Elimination (Gaussian)	67 MB	268 MB	1.07 GB
HotSpot	34 MB	537 MB	2.15 GB
Hybrid Sort	40 MB	200 MB	1.2 GB
Kmeans	41 MB	136 MB	1.36 GB
LU Decomposition (LUD)	17 MB	268 MB	1.07 GB
k-Nearest Neighbors (NN)	71 MB	570 MB	1.14 GB
GPMR-IntCount	128 MB	256 MB	512 MB
GPMR-Kmeans	128 MB	512 MB	1 GB
GPMR-LinReg (Linear Regression)	128 MB	512 MB	1 GB
GPMR-MM (Matrix Multiplication)	128 MB	512 MB	2 GB

TABLE II: The benchmark applications and their input data sizes.

IV. Experimental Methodology

This section describes our test bed, benchmark applications, and the process of evaluating these applications.

A. Experimental Platform

The testing platform is a machine running Linux 3.16.3 on a 4-core Intel Xeon E5-2609V2 processor with 64GB DRAM. It includes an NVIDIA Tesla K20 card with 5GB GDDR5 memory through 16-lane PCIe providing 8GB/s bandwidth. We use an SSD with 768GB and a PMC-Sierra controller supporting NVMe 1.1 commands [11], which can sustain 2.2 GB/s for both read and write.

The system uses the default "performance" governor in the Linux Intel CPU driver as the power management policy. It dynamically optimizes the frequencies of processor cores from 1.2 GHz to 2.5 GHz. To measure the power consumption, we use the Wattsup power meter to read the total system power every second. The test system consumes 117.6 W when idle.

B. Benchmarks

We select 9 data-movement heavy CUDA applications from the Rodinia benchmark suite [12]. They accept files as inputs, and spend 55% of execution time moving data while wasting 88% energy in idle on average according to our experiments. For each benchmark, we generate three

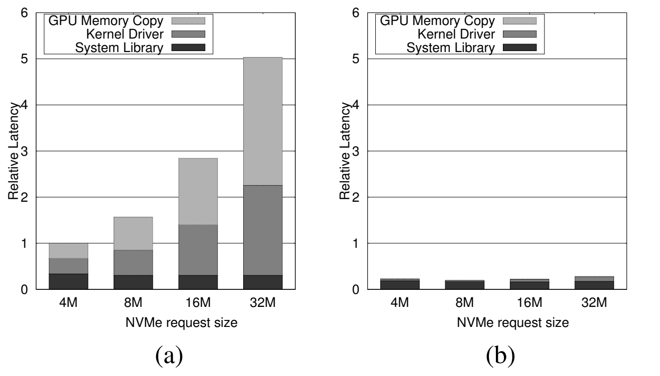

Fig. 2: The latency breakdown for moving data from an SSD to a GPU using (a) NVMe read and (b) NVMeDirect read commands

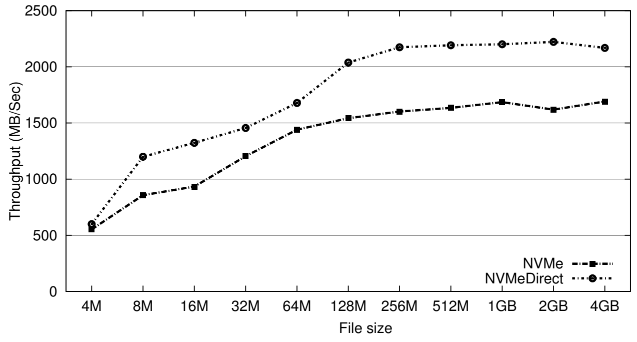

Fig. 3: The throughput comparison of moving data from an SSD to a GPU

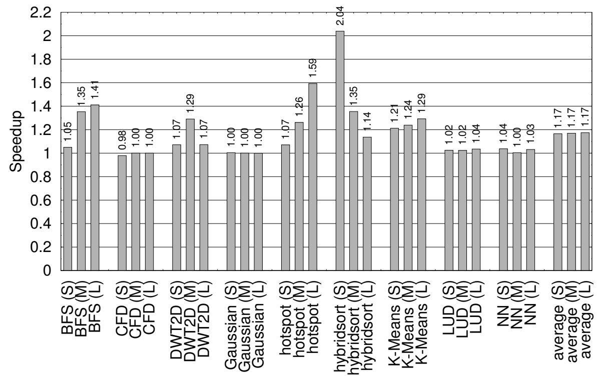

Fig. 4: The speedup of applications using Hippogriff

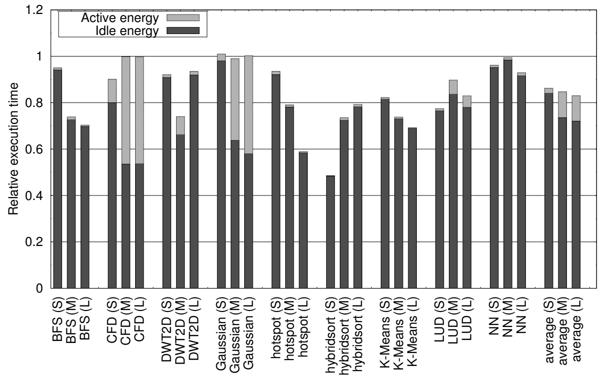

Fig. 5: The relative energy consumption of applications using NVMeDirect

different (small, medium, and large) input sizes, and simply rewrite the data movement part of these applications with `hippogriff_send()` and `hippogriff_recv()`, and eliminate unnecessary main memory allocations. To create multi-program workloads, we slightly modify the code to allocate the required GPU memory before the GPU kernel invokation. We also include GPMR [15], a GPU MapReduce framework, and modify only its file I/O code. Table II shows the benchmark details.

V. Results

We report the experiment results in this section.

A. Comparing NVMeDirect and conventional NVMe

As NVMeDirect removes the CPU and the main memory from the data transfer path, we expect smaller runtime overhead on the host side. To compare the runtime overhead with the conventional model, we examine the number of CPU instructions, and break down the latency for read requests varying from 4 MB to 32 MB as shown in Figure 2. The conventional mechanism devotes 25%–51% of its CPU instructions to moving data from the DRAM to the GPU, which accounts for 33%–51% of the latency. As NVMeDirect bypasses the CPU and the main memory, it eliminates the “GPU Memory Copy” overhead and the overhead of setting up DMA. Therefore, NVMeDirect executes much less CPU instructions with much lower latency.

Figure 3 compares the throughput of moving different sizes of file data from the SSD to the GPU with NVMeDirect against conventional NVMe. The performance advantage of NVMeDirect becomes more significant as file size increases. When transferring a 4GB file, NVMeDirect offers up to 2221 MB/s bandwidth, which is 31.3% higher than NVMe.

B. Impact on single process workload

Figure 4 illustrates Hippogriff can achieve a 1.17× speedup in the end-to-end latency on average for all the input sets. The average speedup for data transfer is 1.45× for large inputs, 1.31× for medium, and 1.24× for small. The effectiveness of Hippogriff becomes more obvious as the data set size increases, since larger input files help amortizing the initialization costs. Hippogriff also reduces the number of page faults by 21% for the small data sets, 26% for medium, and 32% for large.

Hippogriff reduces the load on CPUs during the data transfer, enabling processor cores to operate at lower clock rates or perform more useful work. Figure 5 presents the relative total system energy consumption of the applications using Hippogriff against the NVMe baseline. Hippogriff can reduce total energy consumption by 17%, 15%, and 14% with large, medium, and small inputs, respectively. Since Hippogriff reduces the execution time of programs, we observe 13% reduction in idle energy. Even with an Intel CPU driver that optimizes for energy efficiency, Hippogriff still reduces total system power consumption by up to 8% and saves 30% of application energy over the baseline.

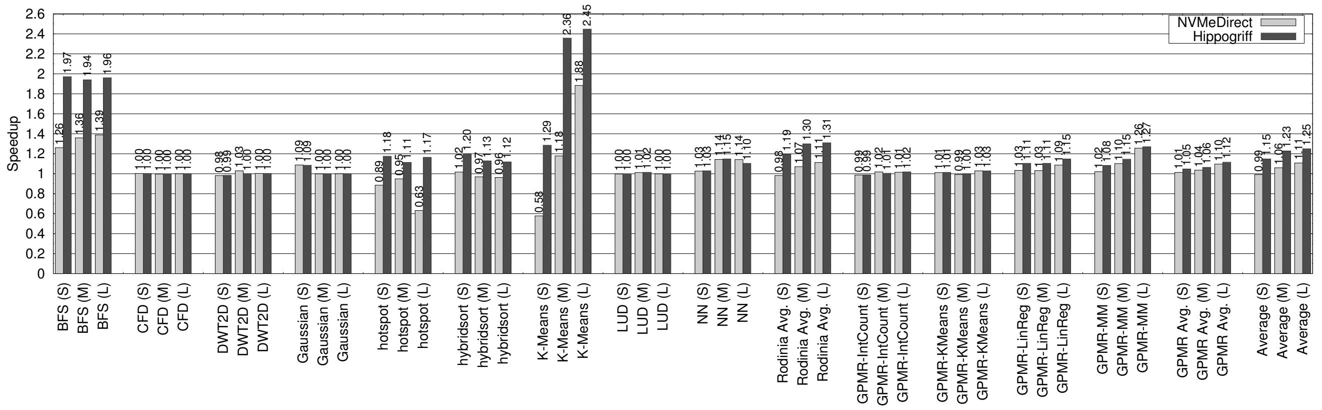
Fig. 6: The speedup of Hippogriff under multiprogram workload

C. *Multi-program workload*

To demonstrate the effectiveness of the Hippogriff runtime in the multi-program environment, we compare Hippogriff against a specialized Hippogriff that always uses NVMeDirect as the baseline. We investigate rodinia benchmark applications and GPMR applications. Figure 6 shows the performance results. With the optimal data transfer path automatically selected by NVMeDirect, Hippogriff achieves a speedup up to 2.45×, and average speedups of 1.25×, 1.23× and 1.15× for large, medium and small datasets, respectively, for all the workloads.

VI. Related Work

Existing works provide peer-to-peer communication between GPUs or other devices [5], [8], but the direct data transfer between GPU and storage is missing. GPUDrive [13] enables direct transfer with a customized PCIe switch to access SATA SSDs, thus cannot fully utilize GPU's high bandwidth. With GPUDirect[10] or DirectGMA [1], Donard [14] and NVMMU [17] exploit commercially available NVMe SSDs and GPUs for direct transfer like Hippogriff does, but they may suffer from severe performance issues; while Hippogriff automatically chooses the optimal data path during runtime for best performance. Hippogriff reduces the importance of CPU performance in heterogeneous computing, and encourages researchers to revisit the design of server architectures together with work like FAWN [4], Gordon [6] and Blade [9].

Heterogeneous System Architecture (HSA) [3] integrates CPU and GPU on the same chip with shared memory, providing another approach for direct data transfer. However, limited by the shared main memory bandwidth and the power constraint of dark silicon [7], [16], such integrated GPUs can only accommodate less than 20% of the streaming units, thus can only deliver moderate performance compared to high-end discrete GPUs like used in this paper.

VII. Conclusion

This paper presents Hippogriff, a system that provides a direct, peer-to-peer data transfer mechanism, and makes the best choice regarding data routes during runtime. With the simplified API, Hippogriff reduce both the user effort in programming and the resulting code size.

Bypassing the CPU and main memory, Hippogriff brings a 1.17× speedup for single program applications, and reduces the energy consumption by 14%–17%. With the optimal data transfer route selected by the Hippogriff runtime, Hippogriff achieves an average speedup of 1.15×–1.25× for multi-program GPU workloads. Overall, Hippogriff completes peer-to-peer data transfer in heterogeneous computing with improved performance.

References

[1] Advanced Micro Devices, Inc. FirePro DirectGMA Technical Overview. http://developer.amd.com/tools-and-sdks/graphics-development/firepro-sdk/firepro-directgma-sdk/, 2014.

[2] Amber Huffman. NVM Express Revision 1.1. http://nvmexpress.org/wp-content/uploads/2013/05/NVM_Express_1_1.pdf, 2012.

[3] AMD. Heterogeneous System Architecture: A Technical Review. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/hsa10.pdf, 2012.

[4] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. SOSP '09, pages 1–14, 2009.

[5] R. Bittner, E. Ruf, and A. Forin. Direct GPU/FPGA Communication Via PCI Express. *Cluster Computing*, 17(2):339–348, June 2014.

[6] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications.

[7] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. ISCA '11, pages 365–376, 2011.

[8] S. Kato, J. Aumiller, and S. Brandt. Zero-copy I/O processing for low-latency GPU computing. ICCPS '13, pages 170–178, 2013.

[9] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. ISCA '08, pages 315–326, 2008.

[10] NVIDIA Corporation. Developing a Linux Kernel Module Using RDMA for GPUDirect. http://docs.nvidia.com/cuda/pdf/GPUDirect_RDMA.pdf, 2014.

[11] PMC-Sierra. Flashtec NVMe Controllers. http://pmcs.com/products/storage/flashtec_nvme_controllers/, 2014.

[12] M. B. S. Che, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. IISWC '09, pages 44–54, Oct 2009.

[13] M. Shihab, K. Taht, and M. Jung. Gpudrive: Reconsidering storage accesses for gpu acceleration. In *Workshop on Architectures and Systems for Big Data*, 2014.

[14] Stephen Bates. PROJECT DONARD: PEER-TO-PEER COMMUNICATION WITH NVM EXPRESS DEVICES.

[15] J. Stuart and J. Owens. Multi-GPU MapReduce on GPU Clusters. IPDPS ' 11.

[16] M. Taylor. Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse. DAC ' 2012.

[17] J. Zhang, D. Donofrio, J. Shalf, M. Kandemir, and M. Jung. Nvmmu: A non-volatile memory management unit for heterogeneous gpu-ssd architectures. PACT 2015, 2015.