AHEAD: **A** Tool for Projecting Next-Generation **H**ardware **E**nhancements on GPU-**A**ccelerate**d** Systems

Hazem A. Abdelhafez
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
hazem@ece.ubc.ca

Christopher Zimmer, Sudharshan S. Vazhkudai
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, USA
{zimmercj,vazhkudaiss}@ornl.gov

Matei Ripeanu
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
matei@ece.ubc.ca

Abstract—**Starting with the Titan supercomputer (at the Oak Ridge Leadership Computing Facility, OLCF) in 2012, top supercomputers have increasingly leveraged the performance of GPUs to support large-scale computational science. The current No. 1 machine, the 200 petaflop Summit system at OLCF, is a GPU-based machine. Accelerator-based architectures, however, add additional complexity due to node heterogeneity. To inform procurement decisions, supercomputing centers need the tools to quickly model the impact of changes of the node architectures on application performance. We present AHEAD, a profiling and modeling tool to quantify the impact of intra-node communication mechanism (e.g., PCI or NVLink) on application performance. Our experiments show average weighted relative errors of $\sim$19% and $\sim$23% for five CORAL-2 (a collaboration between multiple US Department of Energy, DOE, labs to procure exascale systems) and 12 Rodinia benchmarks respectively, without running the applications on the target future node.**

I. Introduction

As of June of 2018, 56% of FLOPS added to the TOP500 [1] [2] list of supercomputers come from GPU-based heterogeneous machines. This trend started with the Oak Ridge Leadership Computing Facility's (OLCF) Titan supercomputer, deployed in 2012. Titan was the first GPU-based machine to exceed 10 petaflops (PF) and as of November 2018, is still No. 9 on TOP500. OLCF has gone on to deploy its next CPU/GPU system, the 200 PF Summit machine, currently No. 1 on the Top500 list. While GPU-based systems offer more FLOPS, they also increase the complexity of the node architecture with intra-node connectivity for CPU-GPU data movement and deeper memory hierarchies. Further, the shrinking procurement cycles do not afford much time to completely explore the effects of such architectural shifts on extreme-scale applications.

Procurements traditionally start with a request for proposal (RFP) process. The clients contribute to an RFP and specify goals for the target systems. These goals will specify elements such as the aggregate memory capacity, flop rate improvement over preceding machines, or application speedup ratios (compared to a figure-of-merit (FOM) reference). Vendors use the FOM numbers provided by the clients to guide the design of next generation machines to achieve the target performance goals. Unfortunately, while vendors may have detailed simulation models of next generation systems, the clients do not, which makes it difficult to validate vendor proposals. Further, this can result in procurements that do not meet the stated RFP targets.

In response to these challenges, the complex procurement process for large machines has become a collaborative process. For example, in 2012, a collaboration between the Department of Energy's (DOE) Oak Ridge (ORNL), Argonne (ANL) and Lawrence Livermore (LLNL) National labs (CORAL) emerged to acquire three leadership scale supercomputers in a single request for proposal (RFP). CORAL resulted in the acquisitions of Aurora, Sierra, and Summit. Both Summit and Sierra, No. 1 and 2 on the TOP500 list as of November 2018, are heterogeneous node architectures, containing multiple GPUs per node. CORAL-2 is the continuation of this collaboration for the procurement of exascale systems in 2021-2022 time frame.

We contribute AHEAD, an application analysis tool that can be used to model data movement in GPU-accelerated node architectures. AHEAD, when used in concert with other existing techniques for modeling processors and GPUs will allow system designers to quickly understand the impact of complex intra-node communication on application performance. In addition we analyze our tool using the CORAL-2 benchmark suite as a case study and identify the set of applications where data-transfers have a significant impact on application runtime.

In AHEAD, we use analytical modeling to understand how a given computing platform's characteristics influence the runtime of target applications. Additionally, analytical modeling is used to identify bottlenecks in the applications. Although analytical modeling provides reasonable estimates and exhibits high flexibility, one of its main constraints in is measuring or estimating model parameters. This becomes even more challenging when we are trying to project the performance of a future system, one that we do not have a prototype of to collect measurements in order to estimate model parameters. In this work, we show how we partially overcome this problem when modeling the data transfer operations on current and

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

This work is supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

future supercomputing nodes.

A. *Scope of this Work:*

In this work, we focus on the data transfers when offloading computations to the GPU because: 1) data transfers contribute significantly to the application run-time (20% on average of GPU time for five CORAL-2 applications); 2) we want to build a tool that allows decision makers study whether the proposed future nodes are over/under-provisioned with regards to intra-node connectivity resources.

B. *Objectives:*

Given two compute nodes A and B, where A is an existing/old node that we have access to, and B is a future/new proposed node; Our goal is to *project the impact of intra-node connectivity changes from node A to node B on the data transfer time in GPU-accelerated applications*. For example, if the CPU-GPU interface bandwidth ratio doubles, how would that impact the data-transfer time when migrating the applications from node A to node B.

C. *Contributions:*

Our contributions can be summarized in three main points:

- *We develop and release an open-source*[1] *application analysis tool (AHEAD).* This tool extracts all GPU-related operations to build an application profile. It is also modular such that one can plug different analytical models to estimate the runtime of different application operations. Section V, provides more details about AHEAD and its implementation.
- *We characterize five different CORAL-2 benchmark applications.* Our characterization is mainly focused on the data transfer aspect of these applications. We use AHEAD to extract the compute kernels and data transfer operations, and then use this information to understand the intra-node communication's contribution to the application runtime. Section III, presents this characterization results.
- *We create an analytical model to project GPU data transfer time on different nodes with PCIe or NVLink connectivity.* Our modeling approach extends a special case (single message transfer) of the LogGP [3] model with a detailed representation of both PCIe and NVLink. We plug this model into AHEAD and evaluate it using five different CORAL-2 benchmark applications and 12 Rodinia benchmark applications. It achieves an average weighted relative error of 19% for CORAL-2 and 23% for Rodinia. Section IV, provides more details about the analytical model. Sections VI and VII, present the evaluation methodology and results respectively.

II. Background

Over the past few decades, the area of analytical performance modeling has witnessed huge attention. We cover analytical performance models of GPU-accelerated systems. The goals of these models can be divided into three categories:

[1]https://bitbucket.org/HazemAbdelhafez/ahead/

GPU Compute-kernels Time Estimation: In this category, the proposed models [4] [5] [6] [7] [8] aim to estimate the runtime of compute kernels running on the GPU. However, none of them incorporate the data transfer time before and after the kernel. Thus, we believe that complementing any of the proposed models here with data-transfer modeling will yield a better estimate of the GPU runtime.

GPU-related Data-transfers Time Estimation: In this category, we cover prior work on estimating the data transfer time between a CPU and a discrete GPU within a single compute node.

In LogGP model [3], the time taken to transfer n bytes between two processors connected with a data link, can be expressed as follows:

$$T = L + O + n * G \tag{1}$$

where L is the link latency, O is additional overhead associated with the transfer, and G is the inverse of the link bandwidth.

Werkhoven et al. [9] extend this model to incorporate overlapping (multiple data transfers on different streams) impact on the transfer time. However, for a single data transfer stream, the model is reduced to the LogGP model. G is calculated as the average bandwidth for different data sizes. This model helps developers select a computation-communication overlapping strategy to reduce the runtime.

Neugebauer et al. [10] present a detailed theoretical and experimental analysis of PCIe interconnect performance. Although the main use case in their work is understanding the PCIe performance impact on the networking of the host machine, we find the theoretical model of the PCIe they presented helpful in estimating data transfer time for GPU accelerated applications. We use the theoretical model they presented in Subsection IV-A to estimate the effective bandwidth of PCIe at different data sizes.

GPU Compute-kernels and Data-transfers Time Estimation: This category includes previous work that aims to estimate both the compute-kernels and data-transfers time of an offloaded portion of a given application on the GPU. Meswani et al. [11] presented a performance estimation framework for hardware-accelerated applications. The main idea behind this framework is identifying common compute patterns in high-performance applications, automatically identifying those that are most likely going to benefit from hardware, and estimating the impact of a target hardware accelerator on them.

Boyer et al. [12] extend Grophecy [7] to incorporate data transfer time in the end-to-end runtime estimation which reduced the relative error from 255% to 9%.

Limitation in Current Approaches: We found that the closest prior work to ours are the models presented by Werkhoven et al. [9] and Boyer et al. [12]. Both of them are simple enough and at the same time achieve good prediction results. However, both models require direct access to target hardware platforms to collect transfer bandwidth measurements at different data sizes. This constraint makes it infeasible to project data-transfers time on future nodes.

Our Approach: We leverage technology specific knowledge to estimate the effective transfer bandwidth at different transaction sizes. Consequently, we can project the data-transfer time over PCIe and NVLink interfaces using data sheets or vendor provided information, and without requiring direct access to the hardware, except for fine tuning of specific model parameters.

III. CORAL-2 Application Benchmark Characterization

We use the CORAL-2 application benchmarks as a case study for our work. In CORAL-2, there are four scalable science applications, six throughput applications, two data science and deep learning suites, and seven Skeleton applications that stress-test specific performance aspects of a target machine. We focus on GPU-related data-transfers time estimation, therefore, we characterize data-transfers in five CORAL-2 applications that: 1) are mainly throughput-sensitive (four out of the five), and 2) readily support CUDA. Three out of the six throughput applications officially support CUDA, one of them depends on Unified Memory (QuickSilver), which is out of scope. We use alternative, mostly non-optimized GPU implementations of the rest of the applications to carry out our study.

Our characterization aims to: 1) quantify the significance of data-transfer operations compared to the compute kernels on the GPU, and 2) reveal the distribution of data-transfer sizes within each application to better understand the connectivity bottlenecks of each application. Table I shows the compute kernels time, the data-transfer time, the percentage ratio of data-transfer time to the total time (kernels plus data transfers), and the number of data-transfer operations per application on a Summit node (the CPU/GPU machine at OLCF). Moreover, it shows that data-transfer operations represent a major part of the GPU-accelerated portion of the application with an average of 20% across the five applications.

We inspected the timeline of the five applications to make sure that there aren't overlapped transfers or compute kernels running on different CUDA streams. We found that both Laghos and Pennant have only a single default stream for all CUDA operations, Minife and Lammps have an extra stream beside the default one, but it is only used for initialization operations and data transfer operations are called sequentially on a single stream. For QMCPACK, we found that there are two additional streams beside the default one, however, with visual inspection of a section of the timeline we found that the application follows a kernel launch pattern across the streams that also leads to a sequential execution of data transfer operations.

Fig. 1a shows the size-based distribution of the data transfer operations across the five applications. These statistics are input configuration dependent. This means that for different input configurations, we could witness a shift in these statistics. For instance, more computing iterations on the GPU would reduce the data-transfer time to compute-time ratio and a bigger problem size can lead to bigger transaction sizes.

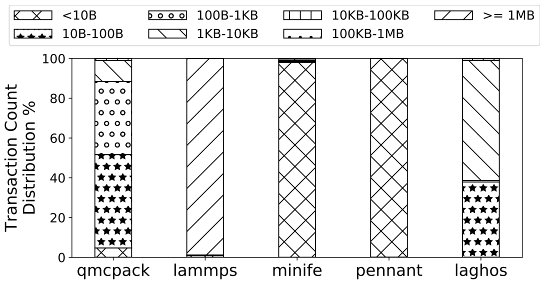
Fig. 1b shows the time-based distribution of the data transfer operations. This figure adds a rather important information not included in the size-based distribution figure. For example, although the majority of data transfers for Minife (95%) are less than 10 Bytes in size, we can see from Fig. 1b that the majority of the data transfer time is consumed by transfers of size greater than 1 MB in size. This means that Minife is throughput sensitive, the same applies for Lammps. We can also see a similarly interesting pattern in Pennant, where in Fig. 1a it seems that the majority of the transactions are less than 10 Bytes. However, in Fig. 1b, we can see that these tiny transactions consume around 60% of the data transfer time, while the rest of the time is consumed by transactions greater than 1 MB.

TABLE I
CORAL-2 Data Transfer percentage ratio to total compute and transfer time

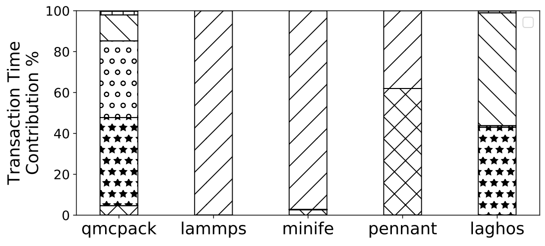
	Kernels(s)	Transfers(s)	$(\frac{\text{DataTransfer}}{\text{Total Time}})$ %	Count
qmcpack	1.01	0.53	34.42	400K
lammps	2.88	2.23	43.64	2K
minife	0.96	0.02	2.04	0.4K
pennant	196.05	0.18	0.09	75K
laghos	3.32	0.82	19.81	520K

IV. Modeling GPU-Related Data Transfers

In a typical GPU accelerated application, several data transfer operations are usually performed to move data between the main system memory and the GPU global device memory. Each data copy operation transfers the data across one or more data links within a compute node. A data link in this context is an abstraction for a physical interface (e.g. PCI link, NVLink, memory bus, etc.) between two or more components in the node.



(a) Transactions count percentage



(b) Transactions time percentage

Fig. 1. CORAL2 data transfer operations analysis

We extend LogGP model with interface-related information to estimate the transfer time. What differentiates our modeling approach is that it requires fewer number of measurements than existing approaches. This is critical for projecting the performance on future nodes.

As we mentioned in section II, the LogGP model according to equation 1 requires three parameters to estimate the transfer time of a single link: latency, overhead and effective bandwidth. In literature, latency and overhead are usually combined together and measured as a single attribute. We measure them by transferring a single byte between the main memory and the GPU memory.

To calculate the effective bandwidth instead of measuring it through micro-benchmarking, we define the effective bandwidth for a transaction of size n bytes as:

$$BW_{eff} = (n * Link\_BW)/\hat{n} \tag{2}$$

where $\hat{n}$ is the effective transferred bytes (the sum of n bytes plus the overhead bytes O_b added by the interface protocol layers).

In this section, we explain how both PCIe and NVLink interfaces work briefly, then we present how to calculate the effective transferred bytes for both of them. We combine this information with their theoretical bandwidth to estimate data transfer time.

A. *Estimating Effective Transferred Bytes Over PCIe*

Peripheral Component Interconnect express (PCIe) is a serial-based communication interface for point-to-point connections in computer systems [13]. It implements a packet-based protocol. Fig. 2 shows a typical architecture of PCIe in a computer system. In this architecture a Root Complex (RC) is the main component that connects the CPU and memory to the PCIe devices (endpoints). Currently, there are four generations of the PCIe specification with the fifth one expected to be standardized in 2019 [13].

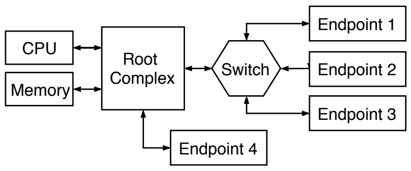


Fig. 2. PCIe architecture

PCIe specification defines three layers for any PCIe link as follows:

- ***Physical Layer***: A PCIe link incorporates 1,2,4,8,16, or 32 lanes. Each lane is a physical transmit/receive pair. Each lane supports a specific raw data transfer rate in each direction (transmit/receive) as shown in Table II. To maintain a DC balance on the transmission wire, PCIe uses an encoding scheme. For instance, Gen 1 and 2 use 8b/10b encoding scheme. This means that for each 8 bits of data, 10 bits are actually transmitted. For Gen 3 and 4, the encoding scheme was replaced with a 128b/130b scheme. Thus, the physical layer efficiency for Gen 1,2 and 3,4 are 80% and 98.4% respectively. The theoretical bandwidth of a PCIe link can be calculated using equation 3.

$$Link\_BW = \#\_of\_Lanes * Lane\_BW * \eta \tag{3}$$

where η is the encoding efficiency mentioned earlier. The per-lane unidirectional bandwidth for different PCIe generations is shown in Table II.

- ***Data Link Layer (DLL)***: It is responsible of managing packet (re-)transmission and its integrity. It handles the ACK/NAK packets which control the retry mechanism in case of a packet drop. It also generates cyclic redundancy codes (CRC) when transmitting a packet and checks them on reception.
- ***Transaction Layer:*** All data transfers via PCIe are transferred as Transaction Layer Packets (TLPs). The Transaction Layer comprises the Configuration space which is necessary for communication with the application layer.

TABLE II
PCIe TRANSMISSION RATES AND BANDWIDTH

	PCIe 1	PCIe 2	PCIe 3	PCIe 4
Raw Bit Rate (GT/s)	2.5	5.0	8.0	16.0
Lane Bandwidth (GB/s)	0.25	.5	∼1.0	∼2.0

Each layer in the PCIe stack adds overhead bytes to carry on its designated tasks. Fig. 3 illustrates the generic format of a PCIe packet.

Transaction Layer Packets: There are several TLPs, however in our model we are only interested in three TLPs: Memory Read Request (MRd), Memory Write Request (MWr), Completion with Data (CplD). The MRd packet is used by an endpoint to request data from the system's main memory, and, if successful, the reply to this request is typically one or more CplD packets containing the requested data. A MWr packet is used by an endpoint to write data to the main system's memory.

PCIe Transmission Parameters: A Maximum Payload Size (MPS) parameter set at system configuration by the Root Complex (RC) determines the maximum amount of data in the TLP payload. The payload can range from 0 to 4KB but typical values for MPS are 128B or 256B. If there are multiple endpoints connected to the RC, the RC sets the MPS to the minimum supported value by any of the connected devices.

A Maximum Read Request Size (MRRS) parameter is set by each endpoint and it determines the maximum number of bytes that can be requested by a single read request. Thus, to read n bytes of data, the number of read requests sent is $\lceil \frac{n}{MRRS} \rceil$. An endpoint can send read requests with $MRRS > MPS$, however, the receiver will satisfy this request with fixed-size packets of size determined by the Read Completion Boundary (RCB). The RCB parameter determines the amount of data in the payload of a reply packet to a read request. Thereby, the number of packets required to satisfy a read request of MRRS size is $\lceil \frac{MRRS}{RCB} \rceil$ where $RCB \leq MPS$.

To understand how the previous parameters fit in our modeling approach of GPU data transfers, we illustrate how to calculate the total number of sent bytes for memory reads and writes as follows:

Memory Read Over PCIe: To satisfy a read request (e.g. cudaMemCpyHostToDevice) from the system's main memory, a requester sends several MRd packets governed by the MRRS size. Once the first MRd packet is received at the RC, it starts sending the requested data simultaneously as it receives the remaining MRd packets. So, in total, the number of bytes transferred to read n bytes of data from the main memory is:

$$\hat{n} = \underbrace{MRd\_O + MRRS}_{\text{First Read Request}} + \underbrace{\lceil \frac{n}{RCB} \rceil * CpID\_O}_{\text{Data-packets Overhead}} + n \tag{4}$$

Memory Write Over PCIe: Write requests (e.g. cudaMemCpy-DeviceToHost) have lower overhead compared to read requests since there are only write packets. Thereby, the total number of bytes sent over PCIe to write n bytes to the main memory is:

$$\hat{n} = \lceil \frac{n}{MPS} \rceil * MWr\_O + n \tag{5}$$

MRd_O, MWr_O and $CpID\_O$ are the overhead bytes for read, write and read-completion packets. They can be 8 (32-bit addressing) or 12 (64-bit addressing) bytes in length.

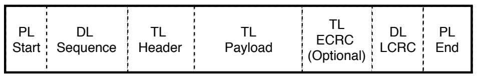


Fig. 3. PCIe packet format.

B. *Estimating Effective Transferred Bytes Over NVLink*

NVLink is an interconnect technology developed by NVIDIA to replace GPU-GPU communication within a compute node [14]. It can also support CPU-GPU communication and Currently only IBM Power processors support NVLink.

NVLink is a packet-based interface. Each packet can contain several (FLow control unITs) Flits. The size of a single Flit is 128 bits. An NVLink packet contains a header Flit, and two optional Flits, Address Extension (AE) Flit, and Byte Enable (BE) Flit. The rest of the Flits can carry payload data up to 16 Flits (MaxPayload of 256 bytes). Fig. 4 shows a typical NVLink packet format.

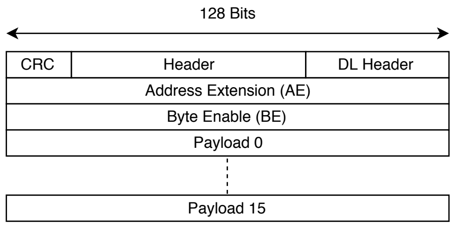


Fig. 4. NVLink packet format.

The header Flit contains 25 bits for error correction (CRC data), 83 bits as transaction information (e.g. control flow, packet type, address), and 20 bits for Data Link (DL) layer information (e.g. packet length, application identifier) [14].

To read n bytes of data, a total of $\lceil \frac{n}{MaxPayload} \rceil$ read requests are sent from the requester to the data store. We assume that there is no AE or BE Flit, up to 16 Flits can carry back the requested data, and that the read requests are pipe-lined same as in PCIe case. Therefore, total number of bytes transmitted over an NVLink interface to read n bytes can be calculated as shown in Equation 6. However, to write n bytes of data, only data packets are transmitted, therefore, the number of bytes required to write n bytes can be calculated as given in Equation 7.

$$\hat{n} = \underbrace{Flit\_size}_{\text{First Read Request}} + \underbrace{\lceil \frac{n}{MaxPayload} \rceil * Flit\_size}_{\text{Data-packets Overhead}} + n \tag{6}$$

$$\hat{n} = \lceil \frac{n}{MaxPayload} \rceil * Flit\_size + n \tag{7}$$

An NVLink is composed of several lanes, each lane has a specific signaling rate that determines how many bits are transferred per second. Therefore, an NVLink unidirectional bandwidth can be calculated as given in equation 8

$$NVLink\_BW = \#\_of\_Lanes * Lane\_BW \tag{8}$$

Each NVLink chip can support several number of NVLinks, hence, the total bandwidth per chip can be calculated as $BW = \#\_of\_Links * NVLink\_BW$.

C. *Estimating Data-Transfer Time*

In CUDA-based applications, host memory can be allocated as either a pageable memory allocated with *malloc*, or page-locked (pinned) memory allocated with *cudaHostAlloc*. The GPU cannot directly access pageable memory, therefore, to copy data from a pageable host memory to the device (GPU) memory, the data has to be copied first from the pageable memory buffer to a temporary page-locked buffer in the device's driver mapped memory region, then to the device memory over an interconnect interface such as PCIe or NVLink.

To copy data from a page-locked host memory to the device memory, there is no need for a temporary buffer and the GPU can either update the pinned memory data directly or transfer the data over the interconnect interface to the device memory. Figure 5 shows reads and writes with pageable and page-locked memory respectively. Therefore, each CUDA memory copy operation is translated to different path on a compute node.

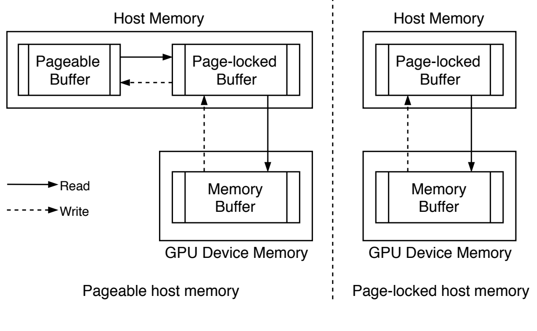


Fig. 5. CUDA memory copy from pageable (left) and page-locked (right) host memory

We model the transfer time of a memory copy operation of n bytes to/from a pageable host memory as given in equation 9, and for a pinned host memory we use equation 10. In both equations, BW_{eff} is the effective bandwidth of either PCIe or NVLink which we show how to calculate in Equation 2. In Equation 9, we multiply n by two to incorporate the effect of reading/writing in a temporary pinned buffer. We measure $(L+O)$ as a single attribute by measuring the transfer time of a single byte. $Memory\_BW$ is the theoretical peak bandwidth of the host memory calculated as follows: $Memory\_BW = Bus\_Width * Frequency * Transfers\_per\_cycle$

$$T = L + O + 2n/Memory\_BW + n/BW_{eff} \tag{9}$$

$$T = L + O + n/BW_{eff} \tag{10}$$

It is worth mentioning that there is an additional type of memory transfer present in nearly all GPU-accelerated applications, this transfer moves data from a specific GPU memory location to another. It can be invoked by specifying the device-to-device flag in the CUDA memory copy API. We model this type of transfer as a simple linear relationship where the time taken to transfer n bytes is calculated as given in equation 10. However, instead of using BW_eff, we use the theoretical GPU memory bandwidth. We measure $(L+O)$ for this type of memory copy operation by transferring a single byte.

Moreover, in multi-GPU systems, one can use our NVLink or PCIe models presented in this section to estimate the data transfer time in GPU-GPU communication.

One of our design goals for the model, is to estimate the transfer time on a future node we do not have access to. Basically, a node that is proposed by a vendor (e.g. Summit) to replace an old one (e.g. Titan). Hence, all the information we know about this node is the interconnect technology (PCIe, NVLink), memory bandwidth, CPU cores count and frequency, etc.

Although our model uses just a single measurement for single byte transfer time (L+O) and the effective interface bandwidth to estimate the transfer time for different transaction sizes, ignoring (L+O) reduces the estimation accuracy when we cannot measure it on a future node.

To overcome this, we found that we can use the measured overhead value from the old node (Titan) when calculating the transfer time on the new node (Summit). This overhead represents a constraint that makes it difficult to project data-transfers on future nodes with high accuracy. Nonetheless, we noticed that the changes in the fixed overhead between different machines, varies slightly compared to changes in the link bandwidth. The exact values used in this study for the parameters of the models discussed in this section are included in AHEAD's source code repository.

V. Implementation

In this section, we detail our implementation of AHEAD. It consists of three main components: Analyzer, Filter and Estimator. Assuming that we have two nodes A and B, our tool can be used to project the impact on runtime of operations on node B using only application traces from a run on node A, along with node B characteristics.

The first step in using AHEAD is to run the application using a profiler on node A to record the application operations in an SQLITE database file. Then, the *Analyzer* extracts all GPU-related operations from the trace file. Next, the *Filter* extracts only the relevant operations' details (data transfer in our case) to feed them to the Estimator. The *Estimator* applies a user-pluggable model to estimate the runtime of the operations on a node B for which we have hardware characteristics. In our case, we extract data transfer operations using AHEAD, then we plug our data-transfer model, and estimate the transfer time on Node B. Fig. 6 shows AHEAD components, workflow and the evaluation steps that we discuss later in Section VI.

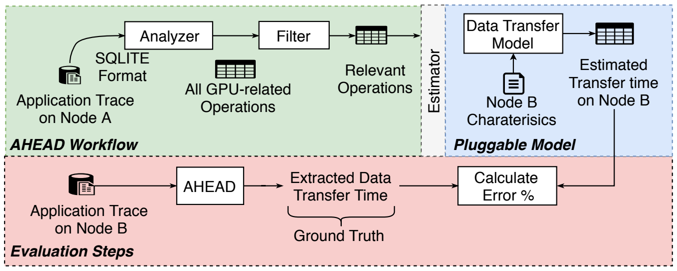


Fig. 6. AHEAD components, workflow and evaluation steps

VI. Experimental Methodology

This section presents the applications and hardware we use in the evaluation of the model. We also discuss our evaluation procedure and alternative approaches we compare against.

A. Applications

We evaluate our model using one bandwidth micro-benchmark, five CORAL-2 applications [15], and twelve Rodinia benchmark [16] applications.

Bandwidth Micro-benchmark: To compare our model to existing models and to better understand its shortcomings, we develop a bandwidth micro-benchmark that initiates data-transfers with different sizes and records the timing of each transfer. We run each experiment 100 times and calculate the average time to reduce the variation.

CORAL-2 Applications: To evaluate our model's capability to estimate data-transfers time, we evaluate it against five CORAL-2 applications. These applications are:

- *Lammps:* An open source large-scale parallel simulator for Atomic/Molecular dynamics. It can run on a single processor or multiple processors using message-passing techniques. It also supports several accelerators such as GPUs and Intel Xeon Phis.
- *QMCPACK:* An open source ab initio quantum Monte Carlo package for electronic structure of atoms, molecules and solids. It is memory bandwidth sensitive when run in production.
- *Pennant:* A mini-app for hydrodynamics on unstructured meshes in 2D. It exhibits irregular memory access patterns.
- *MiniFE:* A proxy application for unstructured implicit finite element codes. It supports GPUs.

- *Laghos:* A mini-app for solving time-dependent Euler equations in a moving Lagrangian frame for compressible gas.

Rodinia Benchmarks: To further illustrate the limitations and strengths of our model, we extend our evaluation to include Rodinia heterogeneous benchmarks [16]. We select 12 applications to cover several application categories (two applications from each category). The applications we choose are: Hotspot3D, Heart Wall, Back propagation, CFD solver, B+ Tree, Breadth-first search, Lower-upper decomposition, Needleman-Wunsch, DWT2d, huffman, LavaMD, and SRAD.

B. *Hardware*

Supercomputer Nodes: We evaluate our model using five CORAL-2 applications on two nodes from two different supercomputers at Oak Ridge National Lab (ORNL). The first one is Titan [17] and the second is Summit [18] (Titan's successor and currently world's fastest supercomputer). Titan has a single 16-core AMD Opteron CPU and a single NVIDIA Kepler GPU (K20x). The main memory is 32 GB in size and the GPU memory is 6 GB with 250 GB/s aggregate bandwidth. The CPU-GPU interface is PCIe Gen 2 with x16 lane width, bringing the total bandwidth to 8 GB/s.

Summit has a more complicated node topology with two IBM POWER9 CPUs, six NVIDIA Volta V100 GPUs. Each processor is connected to a 256 GB of DRAM, bringing the total memory per node to 512 GB. Each GPU has 16 GB of HBM2 memory with aggregate bandwidth of 900 GB/s. Moreover, the main node inter-connect is NVLink V2.0. Fig. 7, shows a full Titan node topology and only half-node of Summit.

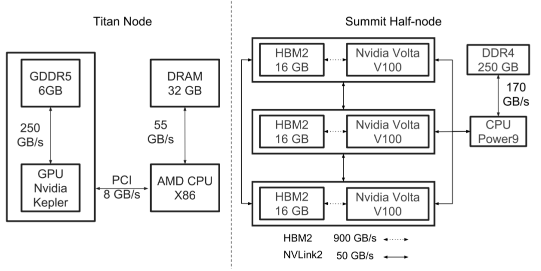


Fig. 7. Titan and Summit architectures

Commodity Hardware: We evaluate our model using Rodinia benchmark applications on two data-center class servers. In the rest of the paper we refer to these servers as Node-1 and Node-2 respectively. Node-1 has 2x Intel Xeon E5-2670 v2 (Ivy Bridge), 10 cores each with 512 GB of DDR3 memory. Additionally, it has 2x NVIDIA Tesla K20C (GK110) GPUs, each with 5 GB device memory. Node-2 has 2x Intel Xeon E5-2650 (Sandy Bridge), 10 cores each with 256 GB DDR3 memory. It also has 2x NVIDIA GeForce GTX TITAN (GK110) GPUs each with 6 GB device memory. Both nodes have a Gen2.0 PCIe interface.

C. *Evaluation Metrics*

Data-transfer Time: In GPU accelerated applications, there are two distinct times consumed by GPU operations:

- *CPU time:* The time spent by the CPU to launch the operation on the GPU or waiting for GPU operation to finish (synchronous operations).
- *GPU time:* The time spent by the GPU to fulfill the operation.

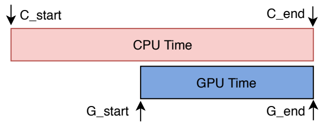


Fig. 8. CPU and GPU timing relations

Fig. 8 shows an example timeline of a simple CUDA memory copy operation on both the CPU and the GPU. Generally, the following rules apply: 1) $C\_start < C\_end$, 2) $G\_start < G\_end$, and 3) $C\_start < G\_start$. Other than these timing rules, any combination is possible. For example, G_start can be $<, =, or > C\_end$. Based on this discussion, CPU time can be calculated as $C\_end - C\_start$ and GPU time as $G\_end - G\_start$. To measure transfer bandwidth, we use the GPU time as such: $BW = \frac{BytesTransferred}{GPUTime}$.
Our decision to use the GPU time is based on three observations: 1) in asynchronous operations, CPU time does not reveal any insight about what is happening on the GPU because the API call returns immediately, 2) CPU time exhibits high variability depending on several factors such as, CUDA runtime API version, driver implementation, operating system, warm/cold caches and CPU utilization, and 3) GPU time is the metric that is impacted directly by hardware characteristics (PCIe or NVLink, memory bandwidth, etc.), and it shows consistent behavior over several iterations and between different applications.

Error Calculation: Since we are evaluating a predictive model, our main evaluation metric is the Weighted Mean Absolute Percentage Error (WMAPE). WMAPE is basically a weighted relative error. The reason why we choose WMAPE is the fact that each data transfer operation can have a different contribution to the overall transfer time, thus, reporting only the relative error might over-signify the error from small-sized transactions (which exhibit a highly variable error). In mathematical notations, weighted absolute percentage error for a single transfer operation can be calculated as follows: $100 * W_i * \frac{|A_i - P_i|}{A_i}$ where A_i and P_i are the actual and projected transfer times for transaction i, $W_i = \frac{A_i}{\sum_{i=1}^{n} A_i}$ where n is the total number of data transfer operations. Finally, WMAPE is calculated as the average over all transactions. Fig. 6 shows a high-level view of our evaluation steps.

D. *Alternative Approaches for Comparison*

The current approach used to estimate the impact of hardware changes on data transfer performance is using back-of-the-envelope calculations (basically, comparing the theoretical peak bandwidth values to estimate how data-transfer operations will behave on new hardware). Although simple and fast, this approach does not always provide accurate estimations, especially for latency-sensitive applications. A different approach is to leverage existing work such as the ones presented by Boyer et al. [12] or Werkhoven et al. [9] to estimate data transfer time in heterogeneous applications. However, both approaches require direct access to the target hardware to collect characterization measurements to fill the model parameters. To compare our work against existing approaches, we use the peak bandwidth instead of measuring the effective bandwidth at a relatively large transaction size (e.g. 512 MB). We refer to this approach in the Results section as Peak Bandwidth, and we also compare against Back-of-the-envelope calculations.

VII. Evaluation Results

A. Bandwidth Micro-benchmark

To understand the sources of error in our approach, we use a bandwidth micro-benchmark for different data sizes. Figs. 9 and 10 show the projected data transfer bandwidth on a Summit node and Node-2 for different data sizes, host memory types (pageable, pinned), and transfer directions (host-to-device, device-to-host).

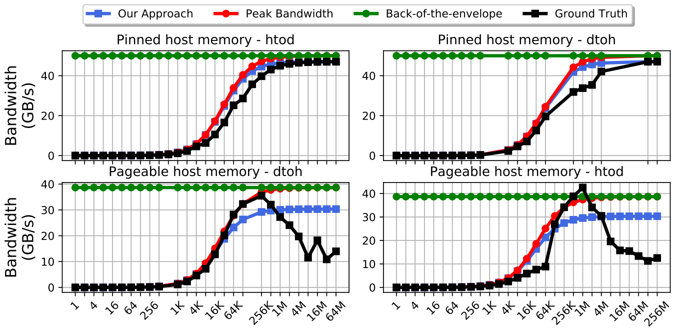


Fig. 9. Summit node projected data transfer bandwidth (y-axis) for different data sizes (x-axis) for pinned host memory (top row), and pageable host memory (bottom row).

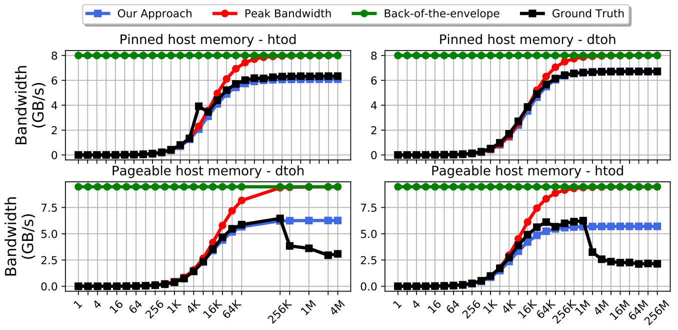


Fig. 10. Node-2 estimated data transfer bandwidth % (y-axis) for different data sizes (x-axis) for pinned host memory (top row), and pageable host memory (bottom row).

In both figures, our model achieves closer projection to the ground truth (measured values) compared to the other alternative techniques. On the bottom half of the figures, we plot the projected bandwidth for pageable host memory transfers. One can notice an irregular behavior at 256KB and 1 MB transfer sizes. This observation is consistent on all nodes. Assuming that there is an overlap between pageable to page-locked transfers and page-locked to GPU memory transfers, we can relate this irregular behavior to different factors such as GPU driver implementation, cache size, CPU utilization, etc.

B. CORAL-2 Applications

We evaluate our modeling approach against five CORAL-2 applications.

Fig. 11 shows the BoxPlot for WMAPE of data transfer time estimation on Summit node for two approaches. The first is our approach, it leverages the NVLink model we introduced in Section IV. The second approach uses the theoretical peak bandwidth.

The third approach uses the theoretical peak bandwidth without the single byte overhead measurement, which is the Back-of-envelope approach. However, we excluded it from the plot as it exhibits high errors and diminishes the plots of the other approaches.

Our approach performs similarly (QMCPACK, Pennant, and Minife) or better (Lammps and Laghos) compared to using the Peak bandwidth in the model. In QMCPACK, the transfer time is dominated by latency/overhead-sensitive transactions, and therefore, both approaches have similar relative estimation errors. Moreover, the variability is high due to the difference in Overhead values we use from the actual values on the Summit node. For Minife, there is a dominant transaction of size greater than 512 MB, therefore, using the peak bandwidth approximates what our NVLink model estimates (NVLink V2.0 has almost 95% efficiency at large transfers). For Pennant, most transactions are from pageable host memory buffers, and as we mentioned earlier, our model does not capture the intricate operational details of the pageable memory transfers, therefore, both our approach and using peak bandwidth yield similar errors and variability.

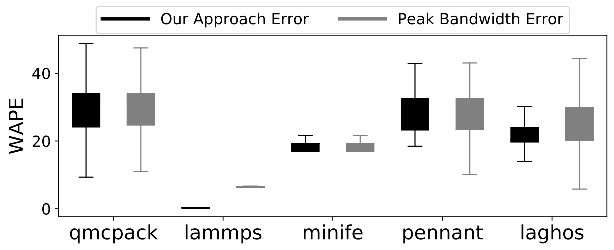


Fig. 11. CORAL-2 data-transfer time estimation error. For each box, the top and bottom sides represent the first and third quartiles (Q1 and Q3). The bottom whisker is calculated as (Q1 - 1.5*IRQ) and top whisker is calculated as (Q3+1.5*IRQ), where IRQ is the inter-quartile range.

C. Rodinia Applications

To further illustrate the strengths and weaknesses of our model, we evaluate it against several applications from Rodinia benchmark. Tables III and IV, show the average and standard deviation values of WMAPE for a default-size run and 4X the default size run respectively. This covers a wide-range of data transfer operations and exposes differences between the evaluated approaches.

For both problem sizes, on average our approach does consistently better than alternative approaches. We investigated the source code and application trace of those applications.

TABLE III
RODINIA APPLICATIONS WMAPE WITH DEFAULT DATA SIZE, EACH CELL REPRESENTS THE AVG/STD

	backprop	hotspot3D	heartwall	cfd	b+tree	bfs	lud	nw	dwt2d	huffman	lavaMD	srad
Our Approach	16.5/18.2	73.5/32.7	1.1/1.4	2.1 / 5.9	8.2/22.1	4.5/17.1	4.1/1.8	68.4/59.1	0.5/0.3	0.2/0.5	20.8/18.6	0.0/0.1
Peak BW	25.4/27.3	105.8/40.3	2.2/2.7	4.8/12.9	12.9/34.4	7.0/26.3	15.1/3.2	99.1/74.7	0.8/0.7	0.5/1.2	32.7/28.6	0.1/0.3
BackOfEnv.	25.6/27.5	105.9/40.3	2.2/2.7	4.9/13.1	13.0/34.6	7.0/26.5	18.1/4.7	99.2/74.7	4.7/2.8	0.7/1.4	32.9/28.7	0.9/0.9

TABLE IV
RODINIA APPLICATIONS WMAPE WITH 4X DEFAULT DATA SIZE, EACH CELL REPRESENTS THE AVG/STD

	backprop	cfd	bfs	lud	nw	dwt2d	lavaMD
Our Approach	15.9/18.0	17.5/40.2	9.3/34.5	69.4 / 4.6	74.9/59.5	0.3/0.3	29.2/19.0
Peak BW	24.7/26.8	25.4/57.6	13.3/49.3	103.0/10.1	107.5/75.1	1.1/1.6	44.3/28.6
BackOfEnv.	24.7/26.8	25.6/57.9	13.4/49.5	103.5/10.1	107.5/75.1	1.1/1.9	44.3/28.6

We found that nearly all of them use pageable host memory allocations except Huffman which has few pinned memory allocations. As shown in Fig. 10, The estimation error varies with the transaction size for the pageable-based host memory transactions. This explains the variability in errors.

VIII. DISCUSSION AND LESSONS LEARNED

Based on the evaluation, we list a couple of lessons learned.

Lesson-1: ***Bandwidth intensive applications.*** We noticed that pageable host memory allocations are frequent in CUDA applications. This is supported by statistics from both CORAL-2 and Rodinia. We suspect developers use pageable memory allocations as pinned memory allocations *(i)* take more time to allocate and *(ii)* they may slow down the overall application if the allocated memory size is too large compared to the total available memory. However, we argue that nowadays, computing systems, especially in HPC environments, contain large amounts of system memory in comparison to GPU memory. For instance, each Summit node has a total of 512 GB of system memory and an aggregate of 96 GB of GPU memory. We recommend that application developers follow a simple trade-off analysis to choose between pageable and pinned memory allocations. This analysis [19] simply compares the time needed to allocate and transfer pinned memory to/from GPU, versus a pageable allocation. At a certain data size (system dependent), there is a break-even point, after which pinned memory allocation overhead is usually amortized by its faster throughput. Additionally, the allocation overhead is amortized over the number of iterations in which data is transferred back and forth between the host and the accelerator.

Lesson-2: ***Latency sensitive applications.*** Finally, if the application is latency sensitive, then there might not be a huge gain choosing pageable vs. pinned memory allocation as the application performance is limited by an almost fixed overhead and link latency. Although there are significant advances in the CPU-GPU interface bandwidth (525% increase from Titan to Summit), the latency sensitive applications might not fully benefit from such improvement. In fact, a one byte transfer time on Summit takes 42% more time than Titan. Hence, it is essential that vendors consider improving the link overhead and latency as much as improving the bandwidth

Lesson-3: ***Overheads related to CUDA API use.*** Considering the impact of CUDA API calls is essential. For example, our profiling reveals that in QMCPACK, there are several CUDA kernels that have several *Constant* input arguments. Constant variables have positive performance impact in some CUDA applications (as they are allocated in dedicated additional cache on the GPU - the constant memory). By inspecting QMCPACK we noticed that there are several *cudaCpyTo/FromSymbol* operations, prepare the values of these arguments, transferring them either from the host memory or from the GPU memory itself. The main problem here is the significant overhead associated with each transfer (range of few microseconds per transaction from application runtime). Moreover, they inefficiently utilize the GPU device bandwidth and the CPU-GPU interface bandwidth. Using our tool, we are able to determine that the Device-to-device Symbol copying operations represent ∼42% of the total data-transfer operations and ∼38% of the total data transfer time for this application.

Lesson-4: ***Transfer time from the CPU perspective has high variability.*** Usually, when the CPU launches a GPU-related operation, it invokes several system calls before the operation actually starts on the GPU. The time it takes to complete those calls is highly variable and depends on the CPU core utilization, on how busy the GPU is for synchronous operations, and cache occupancy. Moreover, for proprietary APIs, such as CUDA runtime APIs, it becomes even more challenging to model understand what is happening under the hood. However, we noticed that while modeling the operations time on the GPU helps us better understand the impact of hardware changes on those operations, it sometimes does not reflect the impact on the runtime from an application user perspective.

IX. LIMITATIONS AND FUTURE WORK

We presented a data-transfer model that uses data-sheet information to estimate the bandwidth of CPU-GPU connectivity. We summarize our model's limitations:

- Our model does not capture the non-linearity in pageable host memory-based data copying. It is accurate for pinned memory transactions but provides only rough estimates for pageable ones. This could be alleviated by scaling the measured pageable-based transfers bandwidth on old nodes using new bandwidth values of the new machines, or tracing CUDA driver low-level system calls to understand how pageable transfers work. One possible explanation for this non-linearity could be that there is a transaction size threshold below which the driver allocates memory in pinned buffers regardless of what the developer specified. Above this threshold, the driver follows the allocation scheme and transfer mode specified by the application developer (e.g synchronous transfer from a pageable buffer allocation). Based on our evaluation results, this threshold could be a value between 256 KB and 1 MB.

- While our model estimates the raw data transfer time in heterogeneous applications, additional information is needed to incorporate the effect of overlapping between transfer operations and other operations (e.g. compute kernels, other data copying, etc.). AHEAD can be extended to incorporate existing work attempts to model this overlapping.
- This study focused on estimating data transfer operations in isolation from any other interference. As a result it considered single-threaded, single-process application runs and excluded the impact of interference between several threads/processes on the effective data bandwidth. This is challenging because it depends on the run parameters of the application and implementation specific details (how many threads/processes, task or data parallelism, communication patterns between these threads/processes, etc.)
- For latency-sensitive applications, our model provides reasonable estimates of the data transfer time if a one-byte data transfer is relatively similar between the old and new node. This appears to be the case for the procurement scenarios we target.

We plan to tackle some of these limitations. For instance, using micro-benchmarks, one can understand how the pageable memory transfers interact with the underlying hardware (caches, registers, etc.), and then develop a detailed model. Incorporating multi-threaded/multi-process versions of those applications is desirable as current and next generation platforms comprise several inter-connected components (CPUs, GPUs, DRAMs, etc). We believe that a graph-based model representing each component as a vertex, and each data interface as a dynamic edge in this graph is appropriate. This graph-based model should help us capture the interference impact on the data links, and it should allow us to better model complex interactions within a compute node (e.g. GPU-to-GPU communication). To study the overlap between operations, we believe that an application time-line reconstruction after estimating how much time each operation would consume on a target node, will provide reasonable estimates of the overall runtime. For the compute time, as we discussed in section II, there have been several models on estimating the compute time of a GPU kernel. Therefore, incorporating any of these models in AHEAD should be feasible, and serves towards end-to-end performance estimation.

X. Summary

This study explores the feasibility of a tool that informs hardware procurement decisions by projecting the impact of hardware enhancements on heterogeneous applications runtime. To this end, (*i*) we build AHEAD, a tool to profile accelerated applications at the node level, (*ii*) characterize the accelerated applications of CORAL-2 benchmark suite, (*iii*) highlight that, for these applications, the data transfer between the host and the device represents a major part of the runtime, and (*iv*) devise on a methodology to project host to device data transfer times that assumes that only limited information on the target architecture is available (our methodology, requires one estimate - a one-byte data transfer cost estimate which can be obtained on an existing system - and the technical sheet specifications of the data transfer interface between the host and the accelerator).

References

[1] Top500 supercomputers. Last accessed Dec. 2018. [Online]. Available: https://www.top500.org/project/

[2] "New gpu-accelerated supercomputers change the balance of power on the top500," last accessed Feb. 2019. [Online]. Available: https://tinyurl.com/yxwy9f53

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: incorporating long messages into the logp modelone step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*. ACM, 1995, pp. 95–105.

[4] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 152–163.

[5] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu, "An adaptive performance modeling tool for gpu architectures," in *ACM Sigplan Notices*, vol. 45, no. 5. ACM, 2010, pp. 105–114.

[6] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling gpu-cpu workloads and systems," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 31–42.

[7] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "Grophecy: Gpu performance projection from cpu code skeletons," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 14.

[8] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in gpgpu applications," in *ACM SIGPLAN Notices*, vol. 47, no. 8. ACM, 2012, pp. 11–22.

[9] B. Van Werkhoven, J. Maassen, F. J. Seinstra, and H. E. Bal, "Performance models for cpu-gpu data transfers," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 11–20.

[10] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding pcie performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 327–341.

[11] M. R. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole, "Modeling and predicting performance of high performance computing applications on hardware accelerators," *The International Journal of High Performance Computing Applications*, vol. 27, no. 2, pp. 89–108, 2013.

[12] M. Boyer, J. Meng, and K. Kumaran, "Improving gpu performance prediction with data transfer modeling," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 1097–1106.

[13] Pci-sig. Last accessed Dec. 2018. [Online]. Available: www.pcisig.com/

[14] Nvlink interface. Last accessed Jan. 2019. [Online]. Available: www.en.wikichip.org/wiki/nvidia/nvlink

[15] Coral-2 benchmarks. Last accessed Dec. 2018. [Online]. Available: https://asc.llnl.gov/coral-2-benchmarks/

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.

[17] Olcf ornl - summit node overview. Last accessed Jan. 2019. [Online]. Available: www.olcf.ornl.gov/for-users/system-user-guides/titan/

[18] Olcf ornl - titan node overview. Last accessed Jan. 2019. [Online]. Available: www.olcf.ornl.gov/for-users/system-user-guides/summit/system-overview/

[19] Pinned versus pageable memory transfers. Last accessed Jan. 2019. [Online]. Available: www.cs.virginia.edu/~mwb7w/cuda_support/memory_management_overhead.html