Oncilla: A GAS Runtime for Efficient Resource Allocation and Data Movement in Accelerated Clusters

Jeff Young*, Se Hoon Shon*, Sudhakar Yalamanchili*, Alex Merritt†, Karsten Schwan†
*School of Electrical and Computer Engineering, †College of Computing
Georgia Institute of Technology
{*jyoung9,sehoon*}*@gatech.edu, sudha@ece.gatech.edu, merritt.alex@gatech.edu,schwan@cc.gatech.edu*

Holger Fröning
Institute of Computer Engineering
University of Heidelberg, Germany
holger.froening@ziti.uni-heidelberg.de

***Abstract*—Accelerated and in-core implementations of Big Data applications typically require large amounts of host and accelerator memory as well as efficient mechanisms for transferring data to and from accelerators in heterogeneous clusters. Scheduling for heterogeneous CPU and GPU clusters has been investigated in depth in the high-performance computing (HPC) and cloud computing arenas, but there has been less emphasis on the management of cluster resource that is required to schedule applications across multiple nodes and devices. Previous approaches to address this resource management problem have focused on either using low-performance software layers or on adapting complex data movement techniques from the HPC arena, which reduces performance and creates barriers for migrating applications to new heterogeneous cluster architectures.**

This work proposes a new system architecture for cluster resource allocation and data movement built around the concept of managed Global Address Spaces (GAS), or dynamically aggregated memory regions that span multiple nodes. We propose a software layer called Oncilla that uses a simple runtime and API to take advantage of non-coherent hardware support for GAS. The Oncilla runtime is evaluated using two different high-performance networks for microkernels representative of the TPC-H data warehousing benchmark, and this runtime enables a reduction in runtime of up to 81%, on average, when compared with standard disk-based data storage techniques. The use of the Oncilla API is also evaluated for a simple breadth-first search (BFS) benchmark to demonstrate how existing applications can incorporate support for managed GAS.

I. Introduction

Recent growth in the size of data sets in the business world [1] has led to an increased need for computational power and large amounts of memory to support high-performance processing of what has become known as Big Data. Graphics processing units (GPUs) can provide added computational power, but data must first be transferred to these devices for processing. In addition, data center applications cannot easily benefit from expensive hardware and specialized software (e.g., message passing) that enables data movement for other domains, such as high-performance computing (HPC), because using these frameworks would require significant changes to the application's design. These constraints illustrate a fundamental problem for memory management in data centers: Efficient memory management for large data center workloads is currently limited by both the difficulty of data movement between nodes and the difficulty of programming business applications to support high-performance data movement.

Furthermore, many business applications, such as the read-only databases used in data warehousing, do not actually require complex solutions to ensure memory coherence between nodes; in some cases they can take advantage of relaxed consistency requirements such as eventual consistency [2] or on-demand consistency [3]. For this reason, it makes sense to implement a memory management scheme that supports simplified application programming and that provides high-performance data movement using non-coherent remote memory accesses. The global address space (GAS) model [4] is proposed as a suitable memory management scheme because it provides a simple mechanism for accessing remote memory (one-sided put/get operations), and because it scales well as the size of a cluster grows. Previous implementations of the GAS model have typically required custom interconnects [5], which can be expensive in terms of cost, or complicated software layers, which can reduce the performance of data movement between nodes [6]. Thus, the ideal memory management solution for these types of Big Data applications would be low-cost, with minimal software interference and coherence protocols, and would be easy to use with applications like data warehousing.

This work proposes such a memory management solution called Oncilla that consists of two pieces: 1) a system architecture built around non-coherent, managed Global Address Spaces and the use of hardware-supported GAS to move data between host memory and GPUs and 2) a runtime abstraction that assists programmers in using the put/get model for efficient data transfer. This work provides a demonstration of the system model as well as performance analysis for both typical data warehousing queries and a breadth-first search

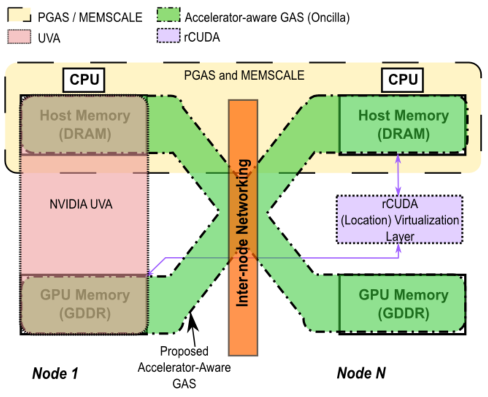

Fig. 1. Address Space Models for Accelerator Clusters

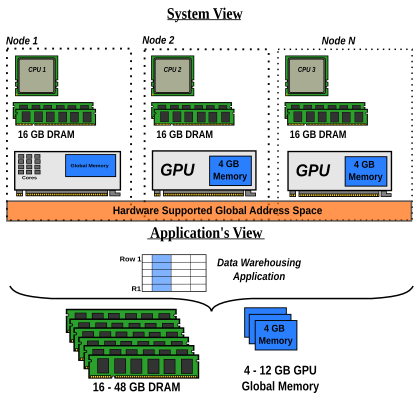

Fig. 2. GAS GPU Cluster System Model

graph algorithm.

II. Motivation for the Oncilla runtime

Other current frameworks, such as GASNet [7], MEMSCALE [8], and rCUDA [9] support some form of GAS-based data movement, so a new runtime should be designed to complement this existing work and also to solve a specific problem. NVIDIA's Unified Virtual Architecture, or UVA, shown in Figure 1, currently has some GAS-like characteristics in that it supports a common virtual address space across all the host and device memory on one node. However, this addressing is currently limited to one node.

GASNet supports the partitioned global address space (PGAS) model of programming with distinct shared and private memory regions (as drawn in yellow in Figure 1) and is primarily focused on improved programmability across a wide variety of cluster interconnects. However, GASNet currently does not focus on data movement between accelerator and host memory. Research into the Phalanx model [10] has extended GASNet with a programming model approach for using GPUs and doing data transfer with GAS, but it is unclear if this GPU support will be merged back into the core GASNet API. Also, both GASNet and Phalanx are limited by their programming model that requires the use of UPC-style primitives to fully take advantage of the underlying GAS model. The use of these primitives may require substantial rewrites for business applications.

From a virtualization perspective, the rCUDA project (represented as a software virtualization layer in Figure 1) implements a solution for virtualizing remote GPUs and handling data transfers using TCP/IP, InfiniBand, and NVIDIA's GPUDirect [11], but it focuses more on the computation aspect of GPU virtualization rather than a cluster-wide aggregation of resources. A closer analogue to our ideal GAS model for host memory and accelerator memory can be found in the data transfer support found in MEMSCALE (also shown in yellow in Figure 1), which shares some characteristics of PGAS models. The MEMSCALE project includes support for using GAS across host memory for standard transactional databases, and the hardware-supported GAS provided by the EXTOLL network is used both by MEMSCALE and by Oncilla.

All of these projects provide different solutions for supporting data movement for multi-node applications, but they are typically focused on the high-performance community and developers who already have a detailed understanding of data movement APIs (e.g., InfiniBand Verbs, MPI, TCP/IP sockets). The ideal model would enable the coordination of cluster I/O, memory, and network resources with an API that provides the simplicity of NVIDIA's UVA, as currently supported on one node.

III. System Model

As shown in Figure 2, the generalized system model used with an Oncilla-supported cluster combines the host memory (DRAM resources) and accelerator memory (GDDR) into a large, logical partition that can be used by one or more applications. Access to this aggregated memory in a high-performance manner is supported by low-latency and high-bandwidth hardware that supports the one-sided put/get model. While overall application performance may be affected by the location of remotely allocated memory, application scaling is not restricted by where the memory is physically located.

Our implementation of this system model combines several different hardware and software components to create a high-performance and commodity solution for allocating remote memory and sharing it more efficiently between cluster nodes. The hardware supported GAS in our model is built around EXTOLL network interface cards (NICs) or InfiniBand HCAs while the software layer is composed of the Oncilla runtime for data allocation and movement and the Red Fox compiler for compilation and optimization of data warehousing application kernels for the GPU.

A. *Red Fox compiler framework*

The Red Fox compiler is designed to run complex queries with large amounts of data for a heterogeneous cluster and is focused on optimizing computation kernels that run well on highly parallel architectures, such as GPUs. As Figure 3 shows, Red Fox is comprised of: 1) A front-end to parse a descriptive language such as Datalog and to create an optimized query plan in the form of a graph of relational algebra (RA) primitives, such as select, project, join, etc. 2)

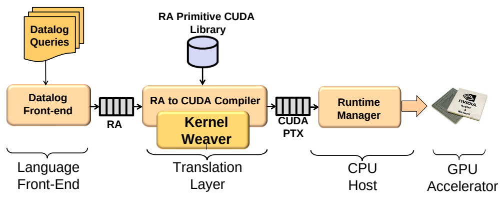

Fig. 3. Red Fox Compiler Flow

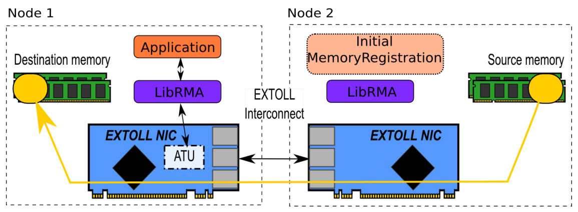

Fig. 4. EXTOLL Remote Memory Access Dataflow

A compiler to map these RA primitives to their corresponding GPU kernels using NVIDIA's CUDA language [12] 3) A kernel optimization component called Kernel Weaver that applies Kernel Fusion/Fission (KFF) optimizations to reduce the amount of data transferred to the GPU and to overlap data movement with computation [13] [14].

Previous work has shown that database computation maps well to GPUs due the parallel nature of many operations [15], and more recent work with Red Fox's Kernel Fusion optimization has helped to reduce the overhead of transferring data over the local PCI Express (PCIe) bus by combining computation that operates over the same input set. While Kernel Fusion can help to mitigate some of the effects of Big Data by limiting the number of PCIe data transfers, the transfer of large in-core data sets to smaller GPU global memory is still limited by the efficiency of data transfer between cluster nodes. The Oncilla runtime discussed in Section IV aims to provide this efficient inter-node data movement mechanism.

B. The EXTOLL Networking Fabric

EXTOLL [16] is a network fabric that allows for the optimization of network characteristics to support specialized networking tasks, including the consolidation of resources and low-latency data transfer between nodes. The two most relevant aspects of EXTOLL are its support of global address spaces (GAS) for consolidation purposes and low-overhead put and get semantics.

By using a direct low-latency hardware path and one-sided communication with remote resources, global address spaces can be spanned up across any number of nodes. From an application's perspective, local and remote accesses cannot be distinguished based on location, but such details can be exposed in the programming model for locality optimizations. These global address spaces can be used for dynamic resource aggregation, thus breaking up the standard static resource partitioning associated with cluster computing. In particular in-memory, data-centric applications like data warehousing can benefit from the vast amount of memory provided by such aggregated resources [8]. In addition, both CPUs and other components like accelerators can access global address spaces for resource aggregation. As disaggregated memory resources cannot easily provide the capacity required for data-warehousing applications, here we use global address spaces for resource consolidation purposes to keep data as close to the processing units as possible.

While put/get operations are provided by many different networks, the one-sided communication operations in EXTOLL are optimized for fine-grained communication schemes and high-core counts. First, the overhead associated with registration/de-registration and initial communication costs are minimized. Secondly, in-hardware virtualization supports multi-core environments without intermediate multiplexing software layers. Features that have proven to be beneficial for HPC applications also are useful for resource consolidation purposes: the need to keep memory ranges registered is low, as the registration process is very fast (2 μs vs. 50 μs initial overhead for small memory ranges when compared to IB [17]). As registration requires pinning pages in main memory, too many registrations can reduce the number of page swap candidates and significantly impact demand paging and system-level performance. Also, while many put/get semantics provide few possibilities for notification about completed operations, EXTOLL provides notifications on both sides (origin and target) that can signal completion to supervising processes with very little overhead. In this work, this feature helps to maximize overlap among the various communication flows.

The EXTOLL NIC currently supports three styles of data transfer: 1) The Shared Memory Functional Unit (SMFU) [18] exposes remote memory via Linux's mmap mechanism and allows for direct manipulation of remote memory using put/get operations on pointers. 2) The Remote Memory Access (RMA) unit supports RDMA-style operations using low-latency pinning of pages as shown in Figure 4. The low-overhead Put/Get operations used with RMA are based on a single-instruction dispatch of work requests [17] and are helpful to minimize communication overhead for fine-grained messaging applications, thus maximizing overlap between computation and communication. 3) The VELO unit can be used to send small messages using hardware "mailboxes" and is used to support MPI over the EXTOLL network. This work makes use of the RMA unit to implement a GAS-supported accelerator clusters, but SMFU or VELO could also be used to perform low-latency put/get operations across the GAS.

Due to the API-based nature of one-sided transfers using the EXTOLL RMA unit as well as the availability of the InfiniBand Verbs API for data transfer we refer to our model as a "managed" global address space, since memory regions are indirectly accessed. This indirect access is mostly due to the limitations of current hardware and software stacks, but the aggregation of multiple, non-coherent regions across the cluster and the use of put/get operations both provides similar functionality to traditional global address spaces.

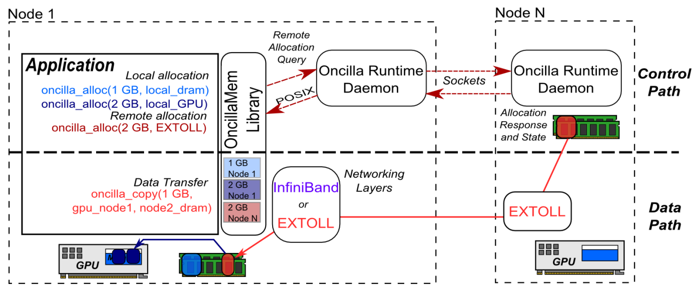

Fig. 5. Oncilla Runtime and use of API

IV. Oncilla Runtime and API

The Oncilla runtime is designed to abstract away the complexity of remote memory allocation and data movement while also providing optimized data transfer between remote and local host memory and I/O memory (on GPUs). As shown in Figure 5, the runtime consists of a control path and a separate data path. The control path performs the request, allocation, and freeing of local and remote memory and is accessed by an application through the "OncillaMem" library. In this example, a local allocation is serviced easily through standard memory allocation techniques (malloc, or new), and a remote allocation initiates the allocation of pinned DRAM pages on the remote node. Once the allocation takes place, the runtime keeps track of the state of the allocation and also any processes needed on each node to set up and tear down memory buffers. This information is made available to the application via the library, and subsequent "oncilla copy" (aka, *ocm_copy*) calls use this state to transfer data over the desired network layer.

The data path could consist of any high-performance network, but this work focuses on the EXTOLL RMA network interface due to its low memory registration costs, high bandwidth, and low latency for small messages. InfiniBand RDMA is a similar high-performance networking fabric that is evaluated alongside EXTOLL in Section VI-A. Regardless of the selected fabric, once the setup phase is complete, the application relies solely on the library and network layer to transfer data from a remote memory to local host memory or local GPU global memory. This abstraction exposes a large memory to the application developer without having to optimize for the specific network underneath.

This idea of abstraction is further defined by the Oncilla API which has the concept of "opaque" and "transparent" operators that support either naïve or explicit hints about remote memory allocation and data placement. These library interfaces allow a developer to specify whether just a large chunk of memory is needed or whether a chunk of memory on a specific device on a certain node is needed. For example, a naïve allocation might specify that 20 GB of host memory is needed but not the location of this memory.

oncilla_alloc(20 GB, DRAM)

This type of operation could allow for 20 GB of DRAM to be allocated on the local node, across two nodes, or across four nodes. The developer in this case accepts the access penalty for remote DRAM rather than using disk-based operations. Alternatively, a developer could specify allocation in a fashion that is closer to traditional NUMA commands.

oncilla_alloc(20 GB, node1_ddr, node2_ddr, equal_alloc)

This type of "transparent" operator can be extended to GPU memory, and the Oncilla runtime helps to allocate and keep track of more advanced allocation and data movement operations. While scheduling of computation and fair memory allocation decisions can also be incorporated into the runtime, this work does not currently attempt to solve this problem. Future work includes evaluating the Oncilla resource management framework alongside existing scheduling frameworks, such as those found in [19] and [20].

A. *Using Oncilla for Existing Single-node Applications*

As Section VI will show, Oncilla can be used to enable multi-node resource allocation and data movement for existing single-node applications. This work has demonstrated a few key steps that are useful to consider when using Oncilla:

- **Plan data partitioning:** The most important step for modifying applications to work with remote memory is to consider how input data should be partitioned. The applications discussed in this paper depend on aggregation of large amounts of remote memory and the use of one accelerator, but the use of multiple accelerators could inform what segments of data should be placed on each node.
- **Support non-coherent memory accesses (or not):** Applications that take advantage of Oncilla must either conform to a model that supports non-coherent put/get operations or include additional software layers to provide consistency between shared regions. TPC-H is an application that performs few data updates, so it conforms well to this standard.
- **Use managed allocations** Oncilla's *ocm_alloc* can currently be used to replace GPU allocations, local DRAM allocations and remote host memory allocations with EXTOLL or InfiniBand. Besides having simplified copy semantics with the *ocm_copy* API call, allocation with Oncilla allows the master node to keep track of existing allocations for more informed resource management across the cluster. An *ocm_localbuf* API call provides a standard C pointer to the Oncilla allocated buffer, minimizing code modifications.
- **Define the data transfer model** For aggregated data workloads like data warehousing, data transfer with Oncilla is performed using multiple *ocm_copy* calls to stream inputs from remote host memory to local accelerators. Other applications may require small reads and writes that could affect data placement strategies and the placement of data transfer calls.

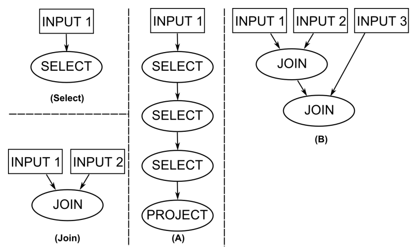

Fig. 6. Relational Algebra Complex Benchmarks

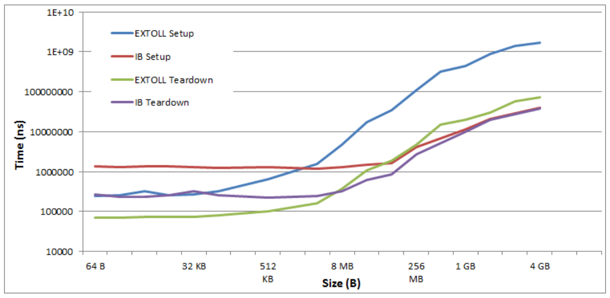

Fig. 7. EXTOLL and IB Server Allocation

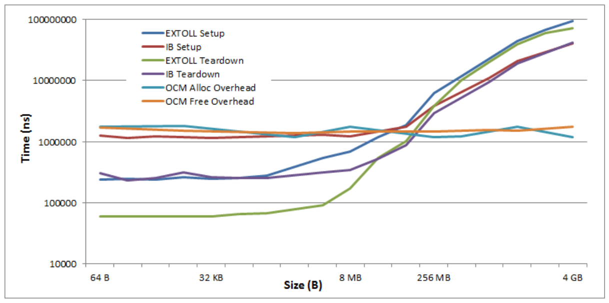

Fig. 8. EXTOLL and IB Client Allocation

V. Experimental Setup

Two different two-node systems are used to evaluate the Oncilla framework. Each blade in the EXTOLL cluster includes a single-socket AMD CPU, a solid-state drive (SSD), 16 GB of DRAM, an EXTOLL2 network interface, and a NVIDIA GTX 480 GPU with 1.5 GB of GDDR. Each node in the InfiniBand cluster contains dual-socket Intel CPUs, a standard 5400 RPM hard drive, 12 GB of DRAM, an InfiniBand ConnectX-2 VPI QDR adapter, and a GTX 670 GPU with 4 GB of GDDR.

The main application workload used to simulate large data warehousing applications is based on relational algebra primitives like those that compose the queries represented in the TPC-H benchmark suite [21] These queries are also representative of the types of operations that Red Fox aims to support for GPU acceleration. These primitives, including *project*, *select*, and *join*, have been optimized for the GPU to perform efficiently when compared to CPU-only primitives [12]. However, the performance of these GPU primitives is limited by two factors: 1) the size of GPU global memory required for inputs, outputs, and temporary data and 2) the cost of data transfer across the PCI Express bus.

As shown in Figure 6, the tests include two simple operations, select and join, and three more complex operations that incorporate select, project, and join primitives. The more complex microbenchmarks also have two variations that can take advantage of Kernel Fusion [13] to reduce the impact of the PCI Express data transfer costs and to reduce each kernel's GPU global memory footprint. These variations are represented as nonfused or fused (not pictured) microbenchmarks in the results.

The input sets for each of these microbenchmarks are transferred to the GPU either directly from a hard drive using standard C++ file operators (fread) or from local and remote memory using the RMA or RDMA hardware. Due to the limitations of the hardware setup, input files are limited to 24 GB, and each node can allocate up to 12 GB of data to be accessed from local memory, RMA, or RMA. The IB cluster is limited to 11 GB, so inputs are pulled from an 11 GB allocation, where 1 GB of input is reused. Data files are represented using binary format and are read sequentially from disk or remote memory.

As a secondary application, the SHOC BFS benchmark [22] is used to represent a simple graph algorithm and to demonstrate how Oncilla can be extended for HPC applications. The benchmark implementation is currently limited to the size of the workload that fits on one GPU, so 1 million vertices are used as input for each test. Two tests are conducted to demonstrate the overhead of using Oncilla for remote memory allocation and data transfer - one where the input graph resides solely in local memory and one where the input graph resides in remote memory. An ideal implementation of this benchmark would use a multi-node, partitioned graph with Oncilla for data movement instead of software layers, such as MPI, but the standard for distributed graph implementations, Graph 500 [23], currently lacks suitable support for GPU-based algorithms. For this reason, we focus on a simple evaluation of the performance aspects and programmability of using the Oncilla model for this type of application.

VI. Results

Before evaluating the applications, we use the Oncilla framework to perform a characterization of the allocation and deallocation characteristics of the EXTOLL and InfiniBand networks. These allocation and deallocation tests look at timing statistics for the creation of symmetric local and remote buffers, and they give insight into which fabric might be best suited for a certain type of allocation.

A. Oncilla Control Path

As Figure 7 shows, the EXTOLL network excels at small allocations while the InfiniBand adapter provides quicker allocations of large memory regions. For the smallest allocation, 64 B EXTOLL takes 250.75 μs versus 1.35 ms in the IB case. However, for a 4 GB allocation, EXTOLL takes 1.67 s while InfiniBand takes 40.54 ms. Since both adapters must register or pin pages in the OS to provide non-coherent buffers for one-sided put/get operations, the difference in performance can be

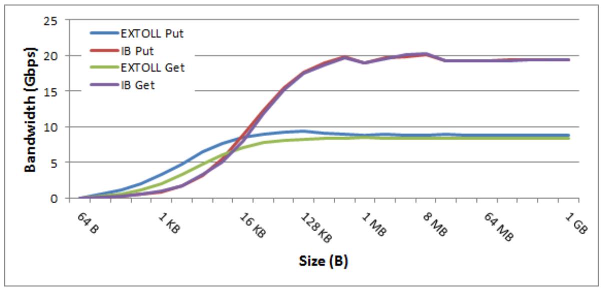

Fig. 9. IB and EXTOLL Bandwidth Comparison

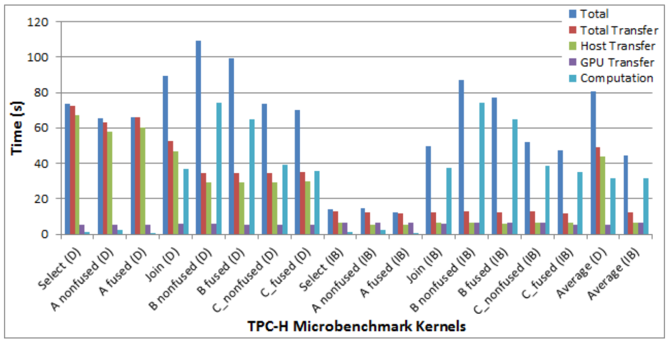

Fig. 10. TPC-H with Oncilla (IB)

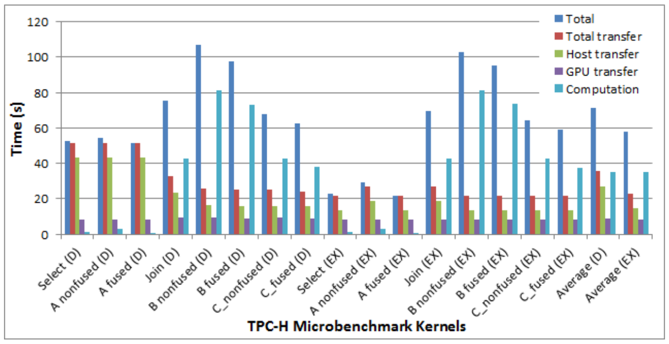

Fig. 11. TPC-H with Oncilla (EXTOLL)

found in the design and optimization of the respective kernel drivers, including support for Linux features like huge pages. For instance, the EXTOLL 4 GB allocation takes 94.89 ms to be registered with the OS while the IB allocation completes the registration process in 38.88 ms. However, for a 64 B allocation EXTOLL requires just 3.10 μs as compared to 37.68 μs for IB. These results indicate that EXTOLL is faster for allocations smaller than 4-8 MB while IB is faster for larger allocations.

Figure 8 shows timing for EXTOLL and IB allocate and free operations as well as the overhead for making an Oncilla allocation for varying sizes of memory regions. EXTOLL takes from 242.66 μs to 95.47 ms to allocate a local buffer and connect to a remote allocation while InfiniBand takes between 1.26 ms and 40.80 ms for the same memory region sizes. Likewise, teardown for EXTOLL takes between 59.12 μs and 73.28 ms to tear down a 64B or 4GB region while InfiniBand takes between 301.42 μs and 42.15 ms.

Oncilla overhead for the allocation of remote memory regions is consistently around 1.18 ms for the two-node case while deallocation of memory regions takes from 1.45 to 1.55 ms. This overhead is due to the time taken for transmission of POSIX and TCP/IP messages that are sent between the Oncilla daemons, so it is not affected by the one-sided setup and teardown that takes place for a server and client allocation. While this overhead is substantial for very small allocations, the use of memory regions larger than 8 MB greatly amortizes the Oncilla setup and teardown overhead. Local host and GPU allocations that use the Oncilla framework also incur a small overhead penalty, on the order of 1 ms, but this overhead allows for tracking of cluster-wide resource allocations by a master daemon that can then make informed decisions about allocation based on the available resources in the cluster.

The use of the Oncilla framework also allows for easy comparison of hardware bandwidths for put and get operations, as shown in Figure 9. The EXTOLL network adapter is based on an FPGA running at 156 MHz, so its realized bandwidth is limited to 9 Gbps (9-10 Gbps, theoretical) while the ASIC-based IB QDR adapter has a maximum realized bandwidth of 20 Gbps (32 Gbps, theoretical). Despite the frequency limitations of the EXTOLL adapter, this diagram also illustrates its suitability for transmission of messages smaller than 8 KB, which is discussed in more detail in [16].

B. Oncilla as an Aggregation Fabric - TPC-H Microbenchmarks

Figures 10 and 11 show results for using Oncilla to aggregate host memory for a data warehousing application, such as TPC-H. Computation time for the kernels on the GTX 670 ranges from 0.17 seconds (A_Fused) up to 74.2 seconds (B_Nonfused), and the fused variants require between 2.21 (A) and 9.36 seconds (C) less computation time to process 24 GB of input data. The use of remote IB memory as shown in Figure 10 for input data transfer is between 4.6 to 10.4 times faster than reading input data from disk, and the use of EXTOLL is 1.2 to 3.2 times faster than transfer from an SSD disk. The use of InfiniBand decreases the total runtime on average by 81% over standard disk (80.88 s vs. 44.20 s) while the use of EXTOLL decreases runtime on average by 22% (71.06 s vs. 58.02 s) when compared to reading from an SSD.

Both figures demonstrate the computational difficulty of doing join operations on a GPU - this is due to the small input size that can be used for inner join on the GPU, and it results in more iterations for the join, B, and C kernels. For the B nonfused kernel, performing two joins requires 74.2 seconds (GTX 670) to 81.3 seconds (GTX 480) while the data transfer requires just 29.0 seconds and 15.9 seconds, respectively. Benchmarks like A_Fused greatly benefit from the use of GPU shared memory and small output sizes and can be processed with input sizes of 368 MB (GTX 480) or 1000 MB (GTX 670), but complex join-based operations

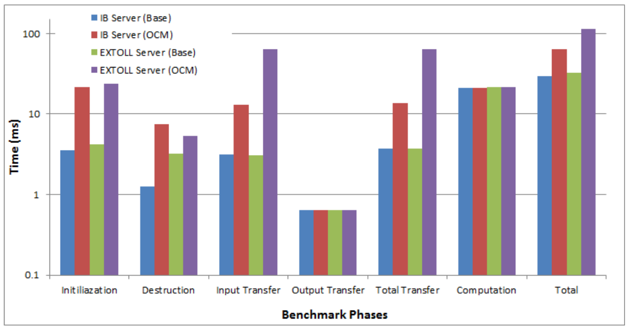

Fig. 12. SHOC BFS - Oncilla Overhead

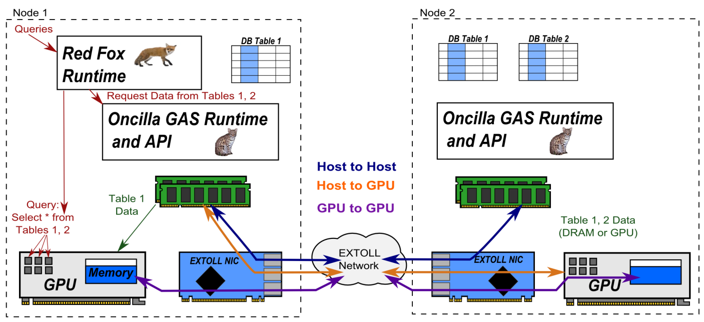

Fig. 13. Oncilla Optimized Data Transfer for Accelerators

(B_Nonfused) require small input sizes of either 10 or 28 MB.

The realized bandwidth for each of the networking protocols ranges from 5.15 to 7.2 Gbps for EXTOLL and 14.8 to 17.5 Gbps for InfiniBand while disk transfer bandwidths range from 1.5 (HDD) to 5.84 Gbps (SSD), depending on the input chunk size for each benchmark. The high bandwidth performance in the SSD case is representative of the sequential nature of these microbenchmarks where large input sets, representing large data tables, are transferred first to host memory and then to the GPU. The use of random access patterns to select small pieces of a database table would likely favor the in-core implementations using IB and EXTOLL, despite SSD's benefits over standard rotational HDDs. GPU transfer bandwidths are relatively consistent at between 2.5 to 3 GB/s (GTX 480) and 3.6 to 4.5 GB/s (GTX 670).

C. Oncilla for HPC Applications - BFS

As a simple example of how Oncilla can be used to enable multi-node resource support for existing applications, the SHOC BFS benchmark (based on the algorithm described in [24]) was migrated to use inputs from remote memory. This migration required just three steps: 1) the replacement of local GPU allocations with Oncilla-managed GPU allocations, 2) the addition of two remote allocations using the Oncilla API, 3) and the replacement of cudaMemcpy calls with calls to *ocm_copy*.

Figure 12 shows the performance overhead of the Oncilla framework when it is used to hold an in-core data store for a standard graph of one million vertices for the SHOC BFS benchmark. The allocation of the edge array and initial adjacency list on a remote node takes about 5.7x (EXTOLL) to 6.1x (IB) longer for the initialization phase and 1.7 to 5.8x longer for teardown. Input data transfer takes 4.2x longer than the baseline case with IB due to its high bandwidth, but EXTOLL in this case takes 20.9x longer due to its lower bandwidth - this uncharacteristic slowdown is likely due to the use of one large EXTOLL buffer holding both inputs as opposed to separate smaller buffers used with IB. Overall, BFS with IB and Oncilla is 2.2x slower than the baseline, and EXTOLL is 3.5x slower than its relative baseline.

Currently, this benchmark is limited to one iteration with input sizes of 4 MB and 8 MB respectively for the edge array and adjacency list. This small input size makes it difficult to recommend using Oncilla for this particular BFS implementation, but the ease of implementation with the Oncilla framework and reasonable performance characteristics would likely make it more suitable for larger graphs.

VII. Related Work

Other recent work has also focused on memory management for in-core business applications. For example, the MVAPICH group has pushed forward in implementing RDMA support for core applications, including HDFS [25]. The Oncilla framework differs from this work in that it focuses on providing a low overhead but network agnostic path for doing remote memory allocations and data transfer. This approach has performance penalties with respect to a straight-forward use of the IB Verbs API, but it also offers a simpler conversion path for using a variety of networking substrates and additional support for remote resource management.

As mentioned previously in Section II, other projects that have focused on using global address spaces to share cluster resources include MEMSCALE [8], rCUDA [9], and APENET+ [26]. Much of the work for each of these projects is focused on providing support for high-performance computing applications, which means that potential applications can be rewritten to take advantage of the high-performance aspects of InfiniBand [27] and CUDA optimizations [28]. Related projects in the GPU virtualization space include DS-CUDA [29], which uses a middleware to launch HPC jobs across many GPUs in a cluster. Oncilla aims to provide similar high-performance data movement while also focusing on a better allocation of resources and better support for a wide variety of accelerator and network resources, as opposed to optimizing for one specific networking fabric.

Finally, projects like RAMCloud [30] are focused on providing large, in-core memory allocations using standardized protocols. Currently, RAMCloud focuses only on host memory and does not incorporate accelerator resources.

VIII. Further optimizations and cluster scalability

Also important to the discussion of resource management is the use of Oncilla as a portability framework. The same application code that is used to enable multi-node memory aggregation on an IB-based cluster can then be moved to an

EXTOLL-based cluster and run without any further modifications. Similar to how open-source frameworks like OpenStack [31] focus on running virtualized applications on a variety of heterogeneous clusters, Oncilla provides a path for application developers to design high-performance applications that can operate on clusters with heterogeneous network architectures. One example would be to use the Oncilla stack to choose, at run-time, the desired networking fabric on a per-allocation basis, dependent on application characteristics and on available networking fabrics. For instance, small allocations and put/get operations might work best with the EXTOLL RMA or VELO protocols, while large, bandwidth intensive transfers might be more suited for InfiniBand.

To this end, the Oncilla software stack must also be capable of optimizing data movement across host-to-host, host-to-GPU, and GPU-to-GPU data transfers as shown in Figure 13. Future work with Oncilla will support CUDA 5 and related devices, which can take advantage of the Peer-to-Peer API to transfer data directly from a NIC to the GPU and reduce inter-node overheads.

Experimental results demonstrate that Oncilla provides a high-performance abstraction for allocating memory on remote nodes and transferring it to accelerators, but many data warehousing applications may scale to several terabytes of data. While the test infrastructure is limited in its installed memory, typical servers can be provisioned with between 256 and 512 GB of host memory. For clusters that are built around a 3D torus interconnect, a node might have up to six nodes one hop away and up to 30 nodes one to two hops away. This translates to potentially having 3 - 15 TB of DRAM that can be incorporated into the global address space and that could contain many data warehousing applications in-core [1]. Oncilla further improves the performance and ease of use of these clusters through low-overhead allocations and high-performance data movement.

IX. Conclusion

The Oncilla framework and associated managed GAS system model provide an important addition to existing research into scheduling and data placement for emerging heterogeneous clusters. Due to the growth of large data sets for Big Data applications like data warehousing and the increasing availability of accelerator clouds, such as EC2, it is important to be able to manage both host and accelerator resources across a variety of cluster infrastructures. This work focuses on two such networking infrastructures, EXTOLL and InfiniBand, and the use of CUDA-based accelerators, but future heterogeneous clusters might include multiple networks and OpenCL, CUDA, or custom accelerators. In addition to future research into extending Oncilla for use with the full Red Fox framework, future work will likely focus on the convergence of different accelerators in a single cluster as well as further research into how to cooperatively manage the allocation of cluster-wide resources with scheduling for both standard and virtualized applications.

X. Acknowledgements

We would like to acknowledge the EXTOLL team for their expert assistance with hardware debugging and cluster support as well as Advanced Industrial Computer (AIC) for the donation of HTX-based server blades.

References

[1] IOUG, "A new dimension to data warehousing: 2011 IOUG data warehousing survey," 2012.
[2] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, Jan. 2009.
[3] H. Fröning, H. Montaner, F. Silla, and J. Duato, "On memory relaxations for extreme manycore system scalability," *NDCA-2 Workshop*, 2011.
[4] J. S. Chase *et al.*, "Sharing and protection in a single-address-space operating system," *ACM Trans. Comput. Syst.*, vol. 12, no. 4, Nov. 1994.
[5] C. Vaughan *et al.*, "Investigating the impact of the Cielo Cray XE6 architecture on scientific application codes," in *IPDPSW*, May 2011.
[6] L. A. Giannini and others., "A software architecture for global address space communication on clusters: Put/Get on Fast Messages," in *HPDC*, 1998.
[7] D. Bonachea, "GASNet specification, v1.1," Tech. Rep., 2002.
[8] H. Montaner *et al.*, "MEMSCALE: A scalable environment for databases," in *HPCC 2011*, 2011.
[9] J. Duato *et al.*, "Enabling CUDA acceleration within virtual machines using rCUDA," *HiPC*, 2011.
[10] M. Garland, M. Kudlur, and Y. Zheng, "Designing a unified programming model for heterogeneous machines," in *SC*, 2012.
[11] G. Shainer *et al.*, "The development of Mellanox/NVIDIA GPUDirect over InfiniBand - a new model for GPU to GPU communications," *Journal of Computer Science - Research and Development*, Jun. 2011.
[12] G. Diamos *et al.*, "Relational algorithms for multi-bulk-synchronous processors," *PPoPP*, 2013.
[13] H. Wu *et al.*, "Kernel Weaver: Automatically fusing database primitives for efficient GPU computation," *MICRO*, 2012.
[14] H. Wu *et al.*, "Optimizing data warehousing applications for GPUs using kernel fusion/fission," *IPDPS-PLC*, 2012.
[15] B. He *et al.*, "Relational query coprocessing on graphics processors," *ACM Trans. Database Syst.*, 2009.
[16] H. Fröning, M. Nüssle, H. Litz, C. Leber, and U. Brüning, "On achieving high message rates," *CCGRID*, 2013.
[17] M. Nüssle *et al.*, "A resource optimized remote-memory-access architecture for low-latency communication," in *ICPP 2009*.
[18] H. Fröning *et al.*, "Efficient hardware support for the Partitioned Global Address Space," *IPDPS-CAC*, 2010.
[19] V. Ravi *et al.*, "Scheduling concurrent applications on a cluster of CPU-GPU nodes," in *CCGrid 2012*, 2012.
[20] A. M. Merritt *et al.*, "Shadowfax: Scaling in heterogeneous cluster systems via GPGPU assemblies," in *VTDC 2011*, 2011.
[21] "TCP-H benchmark specification," http://www.tpc.org/tpch/.
[22] A. Danalis *et al.*, "The Scalable HeterOgeneous Computing (SHOC) benchmark suite," in *GPGPU 3*, 2010.
[23] R. C. Murphy *et al.*, "Introducing the Graph 500," in *Cray User's Group (CUG)*, 2010.
[24] S. Hong *et al.*, "Accelerating CUDA graph algorithms at maximum warp," ser. PPoPP '11, 2011.
[25] N. S. Islam *et al.*, "High performance RDMA-based design of HDFS over InfiniBand," in *SC 2012*.
[26] M. Bernaschi, M. Bisson, and D. Rossetti, "Benchmarking of communication techniques for GPUs," *J. Parallel Distrib. Comput.*, 2013.
[27] C. Reano *et al.*, "Cu2rcu: Towards the complete rCUDA remote GPU virtualization and sharing solution," in *HiPC*, 2012.
[28] M. Bernaschi *et al.*, "Breadth first search on APEnet+," in *SC Companion (SCC)*, 2012.
[29] M. Oikawa *et al.*, "DS-CUDA: A middleware to use many GPUs in the cloud environment," in *SC Companion (SCC)*, 2012.
[30] J. Ousterhout *et al.*, "The case for RAMCloud," *Commun. ACM*, 2011.
[31] S. Crago *et al.*, "Heterogeneous cloud computing," in *CLUSTER*, 2011.