Oncilla - a Managed GAS Runtime for Accelerating Data Warehousing Queries

Jeffrey Young, Alex Merritt, Se Hoon Shon

Advisor: Sudhakar Yalamanchili

4/16/13

Sponsors: Intel, NVIDIA, NSF

The Problem – Big Data in Data Centers

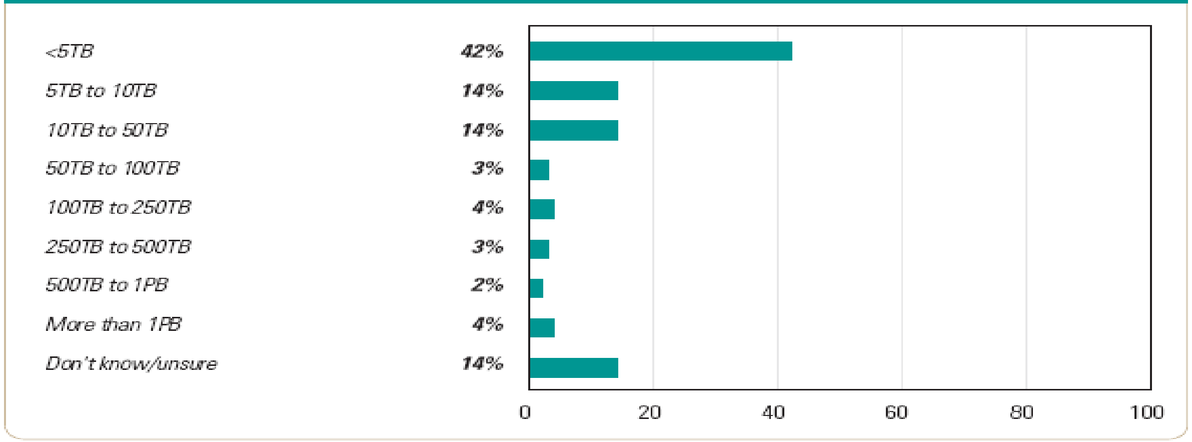


Figure 10: Total Amount of Data Warehouse Data (Histogram)

- Current data warehouse applications process anywhere from 1 to 50 TB of data
  - Expected to grow at a rate of up to 25% per year [1]
- Accelerators like GPUs can be used to accelerate queries for data warehousing applications with large amounts of data
  - Co-processing with GPUs provides 2-27x speedup [2]
  - Our group has recently implemented all TPC-H queries for GPU [3]

[1] Independent Oracle Users Group. A New Dimension to Data Warehousing: 2011 IOUG Data Warehousing Survey.
[2] B. He, et. al, "Relational query coprocessing on graphics processors," ACM TODS, 2009
*[3] H. Wu, et al, "*Red Fox: An Execution Environment for Data Warehousing Applications on GPUs" (under review)

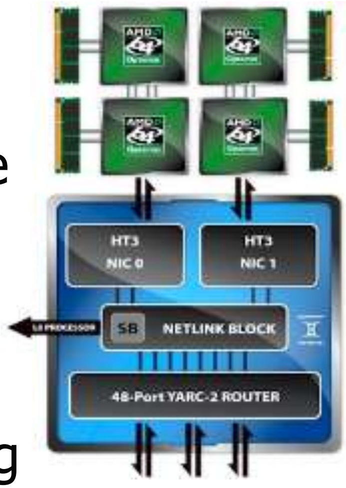
Big Data in Data Centers - 2

- However, GPU processing is limited by the size of on-board memory, typically 4 – 8 GB
- SSD disks have access latencies of 30 or more microseconds compared to DRAM latencies in low nanoseconds (SSD bandwidths are 1 – 1.5 GB/s) [4]
- 75% of typical data warehousing users surveyed feel that in-memory is needed for competitiveness and real-time analytics [5]
- However…
  - Data movement to supply high-bandwidth accelerators is currently difficult for large clusters
  - Interconnects tend to be expensive (Cray) while software has limited performance (TCP/IP)

[4] Independent Oracle Users Group. Accelerating Enterprise Insights: 2013 IOUG In-Memory Strategies Survey
[5] Lystro Warpdrive specs: http://www.supermicro.com/products/nfo/PCI-E_SSD.cfm?show=LSI

Why Can't HPC Techniques Apply Directly to Datacenters?

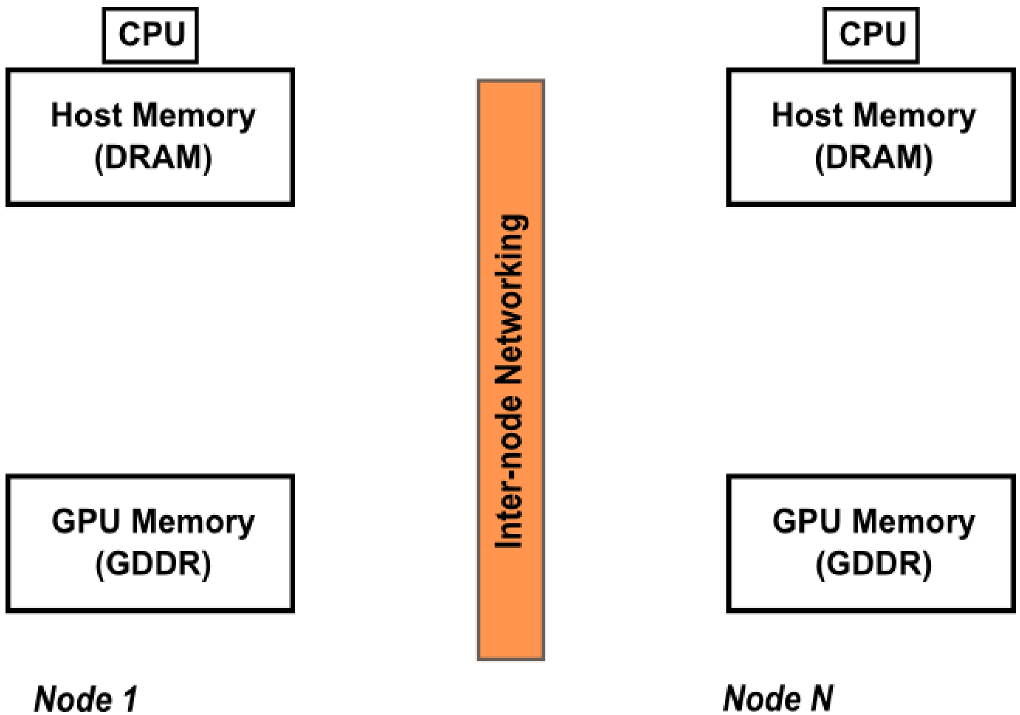
- HPC:
  - Interconnects:
    - Gemini interconnect: 6.6 GB/s [6]
  - Programming
    - Optimized for multi-node clusters with GAS, MPI
  - Accelerators
    - Heavily embraced in newest systems including aggregating multiple types of accelerators



- Data Centers:
  - Interconnects:
    - DDR and QDR IB still most common; 40 Gb/s common for switching
    - 10 Gb/s Ethernet common for switching
  - Programming
    - Based around multi-core architectures and one type of accelerator
    - Limited sharing between nodes

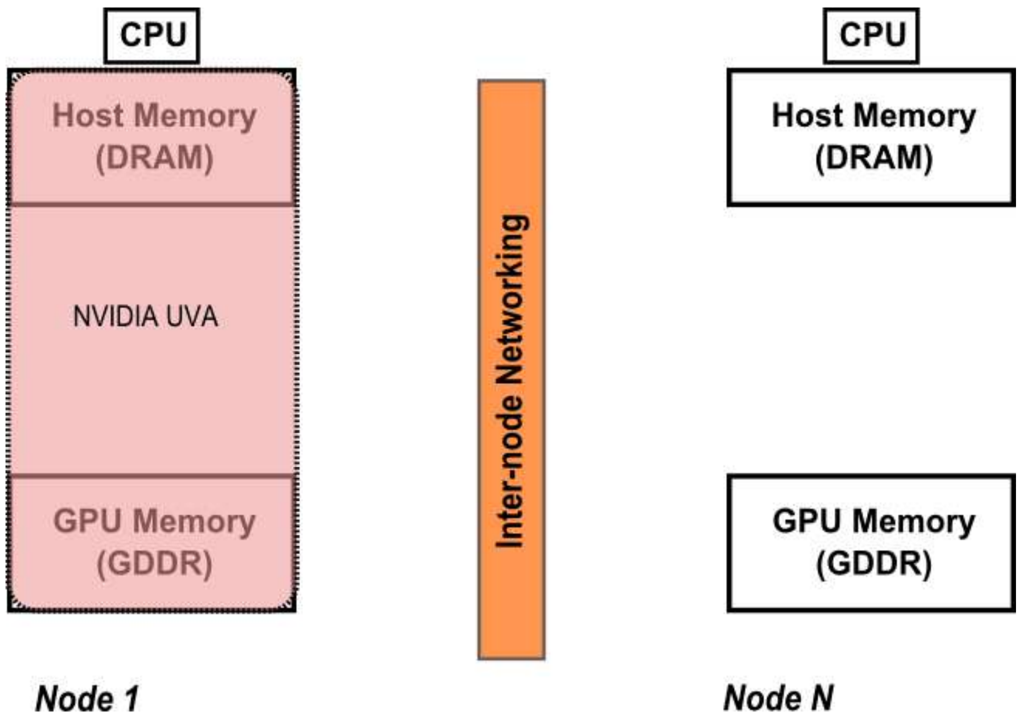
[6] A. Vishnu, Evaluating the Potential of Cray Gemini Interconnect for PGAS Communication Runtime Systems, HOTI 2011
image: http://www.theregister.co.uk/2010/05/25/cray_xe6_baker_gemini/page2.html

Current State of the Art



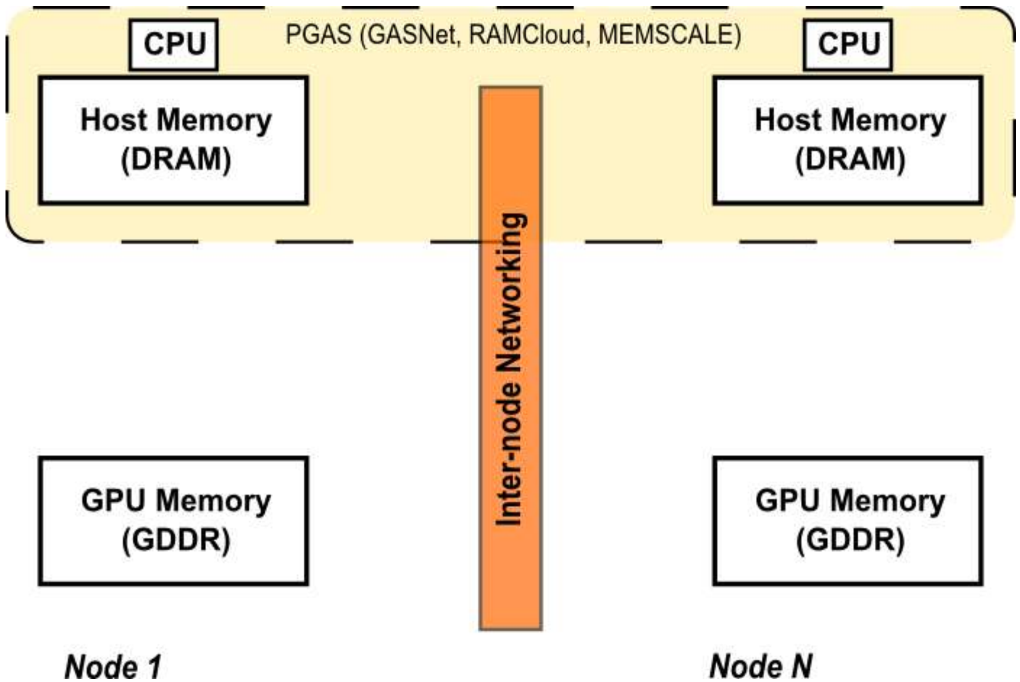
If we want fast, aggregated memory why don't we use existing techniques from the HPC world?

Current State of the Art



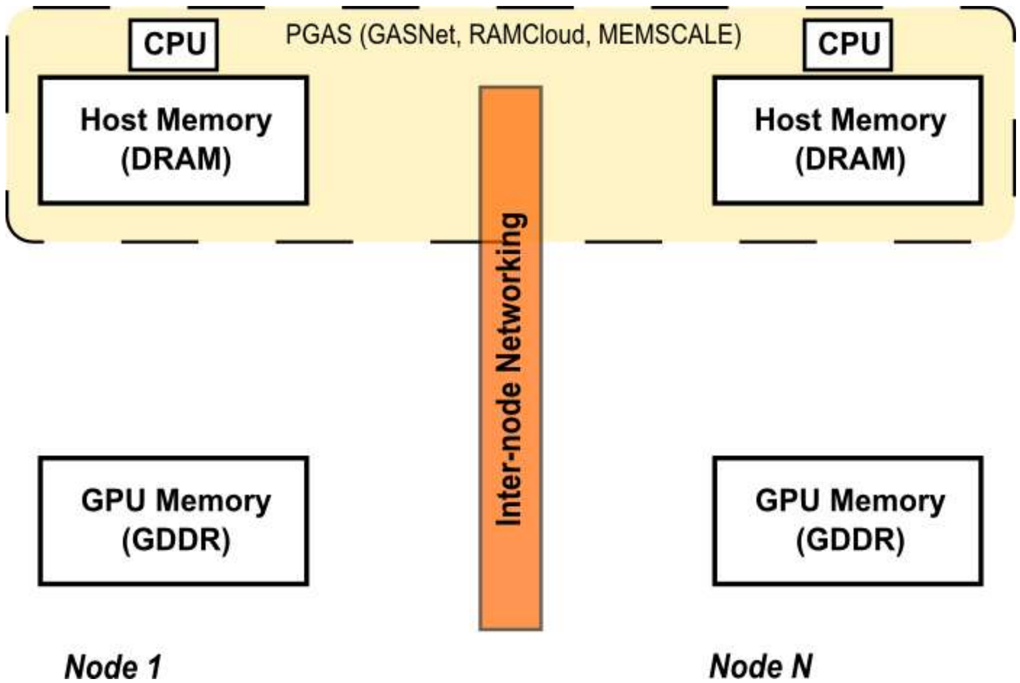
- NVIDIA UVA/UVM + MPI + GPUDirect
  - **Pros:** High-performance, allows managed access to remote memory and accelerator memory
  - **Cons:** Business applications rarely fit neatly into message passing framework; no explicit aggregation; programming complexity

Current State of the Art



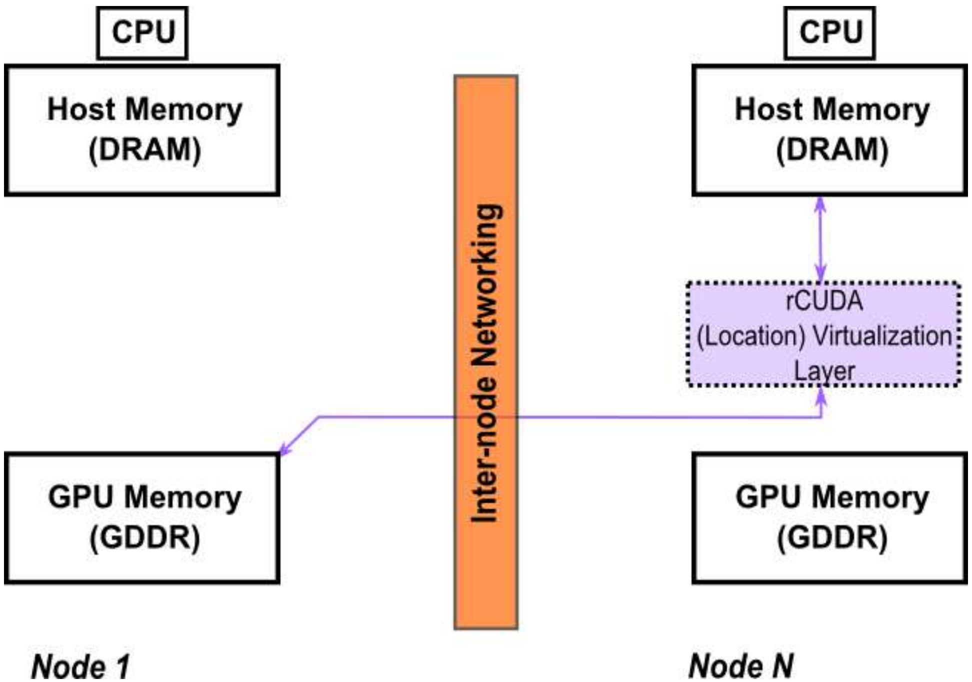
- RAMCloud
  - **Pros:** Scalability; future plans for consistency support; simple support for in-core applications
  - **Cons:** No current plans for accelerator memory support

Current State of the Art



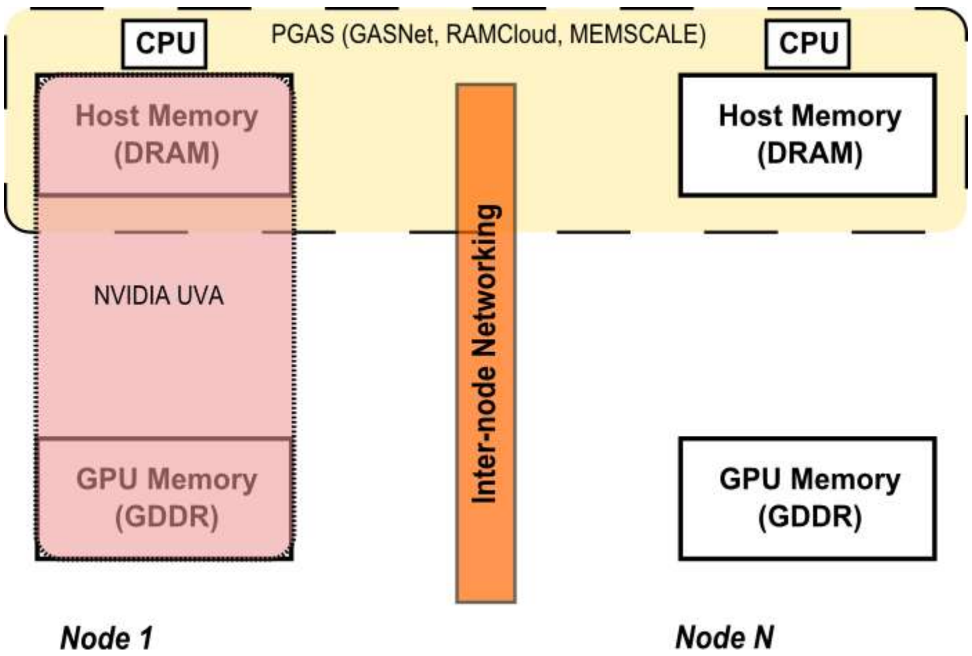
- GASNet
  - **Pros:** Good support for GAS; built-in support for a variety of "conduits"; established user base in the HPC community
  - **Cons:** Built around the usage of UPC compiler and language; GPU support is currently piecemeal or UPC dependent. No real support for aggregation.

Current State of the Art



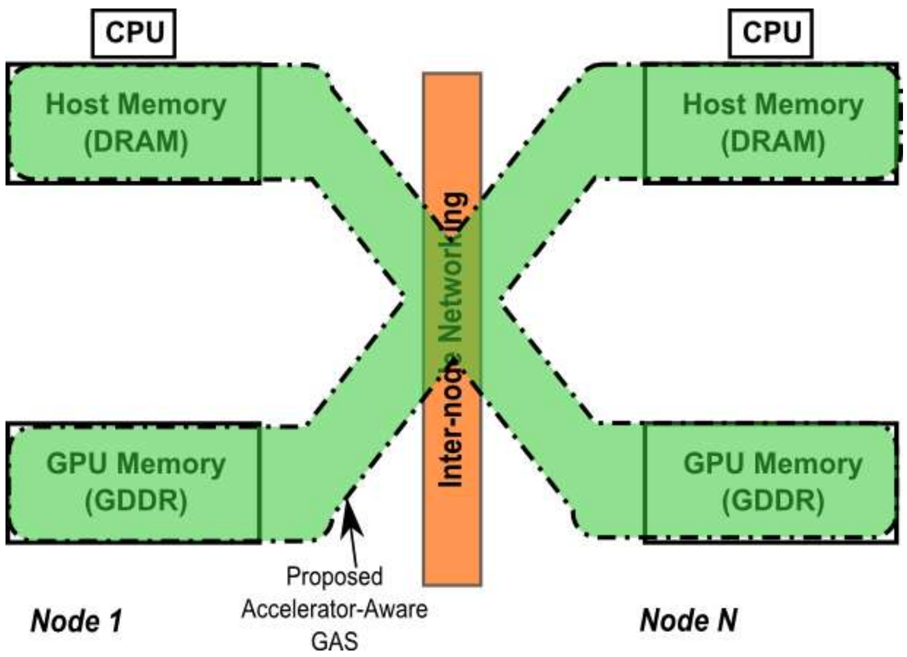
- rCUDA
  - **Pros:** Supports virtualized access to remote GPUs and remote resources. Supports high-performance data transfer.
  - **Cons:** Not focused on aggregating both host and GPU memory resources for a single application

Current State of the Art



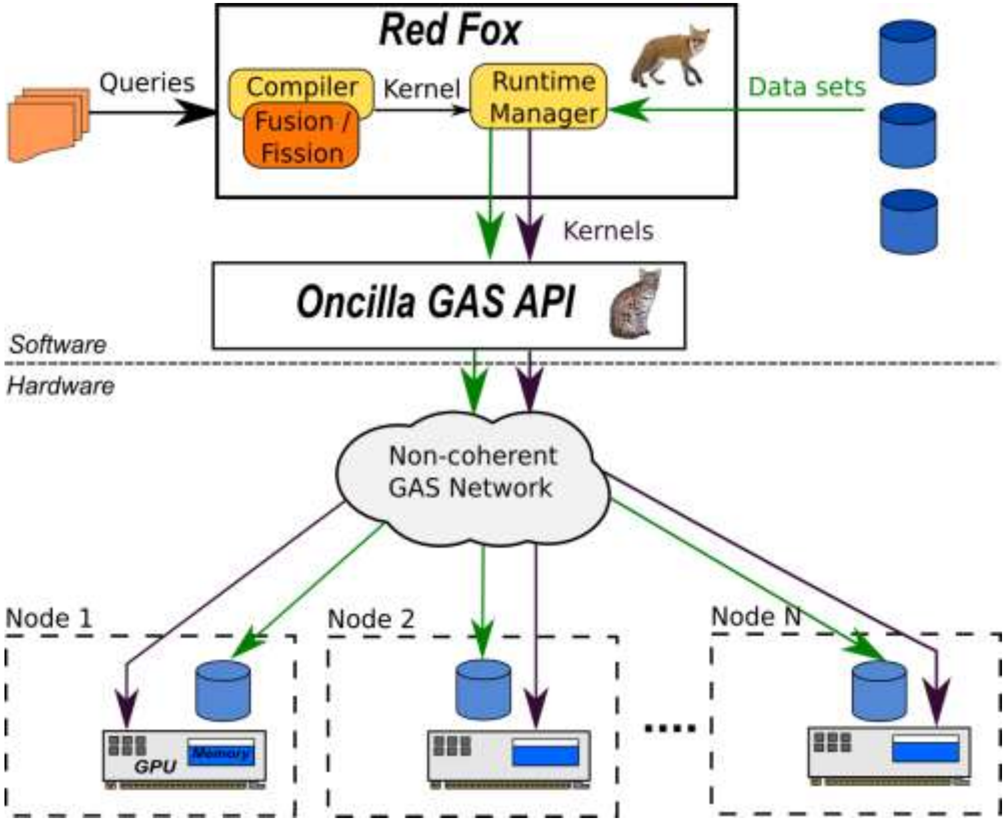
- Phalanx (SC 2012)
  - **Pros:** Uses GASNet and UVA to provide data movement between remote host memory and GPUs. Pointer-based addressing of remote memory. Good scheduling for remote GPUs
  - **Cons:** Requires the use of GASNet for multi-node applications. May be too complex for business applications.

Current State of the Art



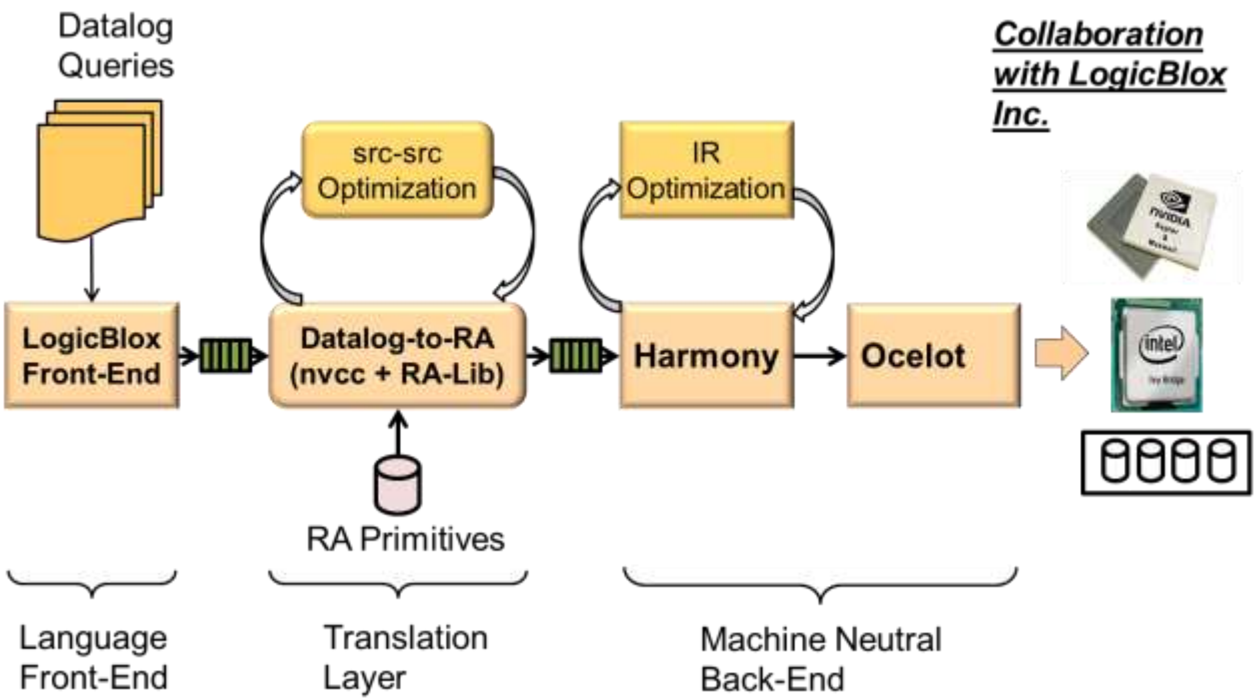
- We would like a simple method for aggregating host and GPU memory that is **user-friendly** and **high-performance.**
  - Focus on applications that can use non-coherent get/put operations

Oncilla Runtime and API



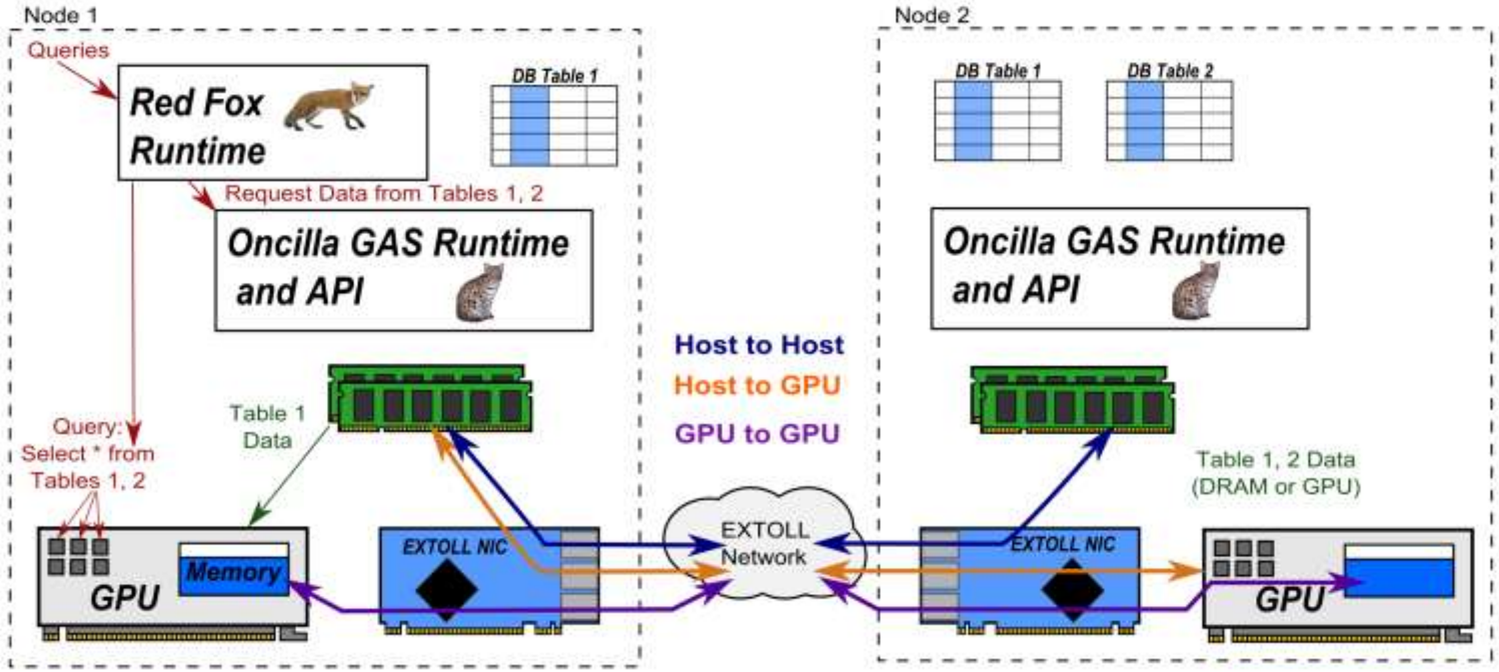
Our Big Data application – Red Fox

- Compiler that generates optimized CUDA queries from Datalog primitives
  - Translation layer uses optimized RA primitives and fusion/fission to combine multiple queries into one kernel



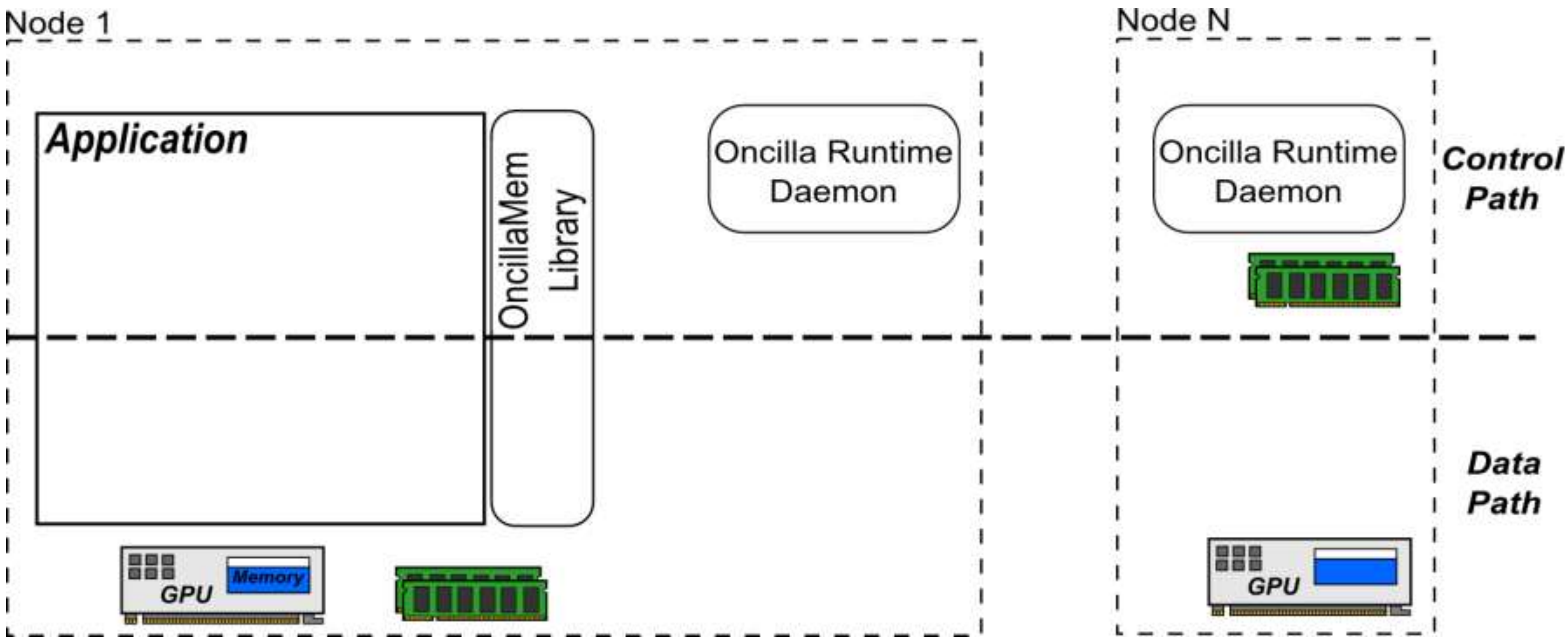
Collaboration with LogicBlox Inc.

Oncilla – A (managed) GAS Runtime for Accelerator Clusters



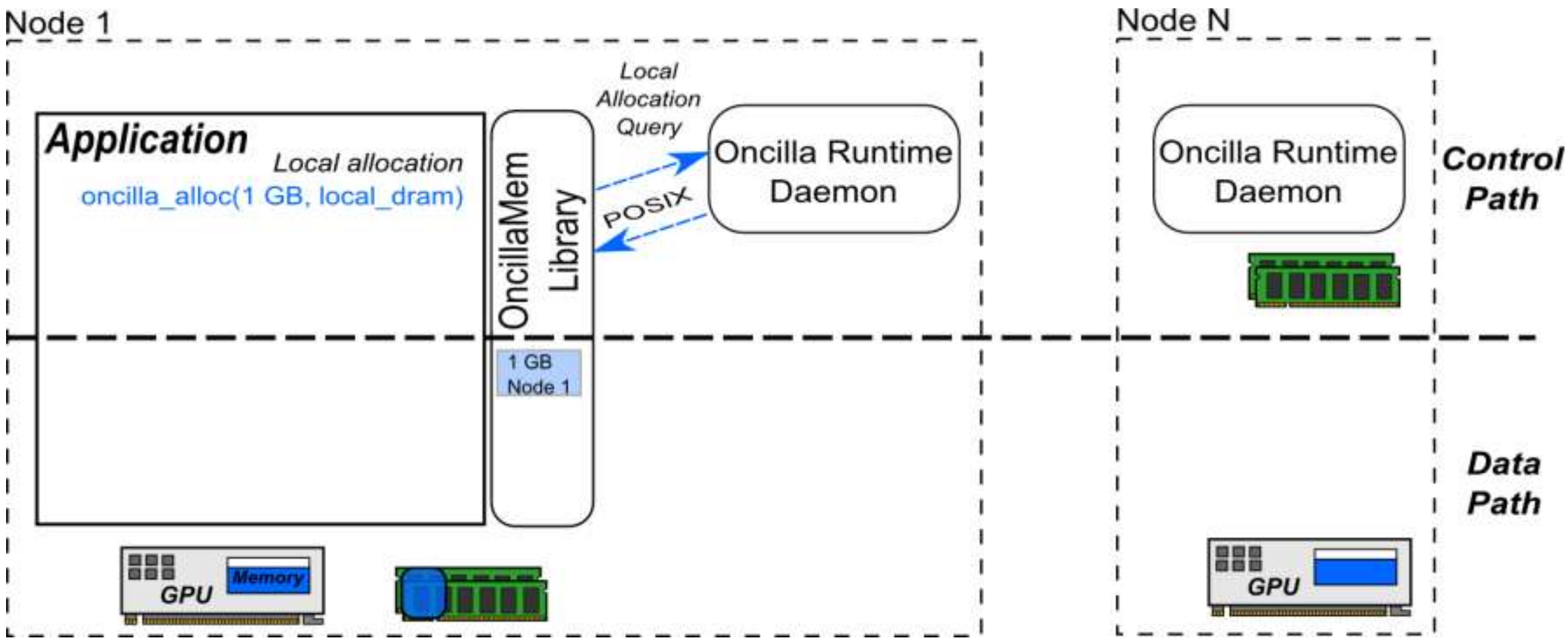
- Oncilla provides high-performance memory aggregation and data movement for applications such as the Red Fox compiler (GPU optimizations for Datalog queries)
  - Consists of a **runtime for allocation** and a **library** that can be linked with gcc or nvcc

Oncilla – Runtime Allocation and Copy Example



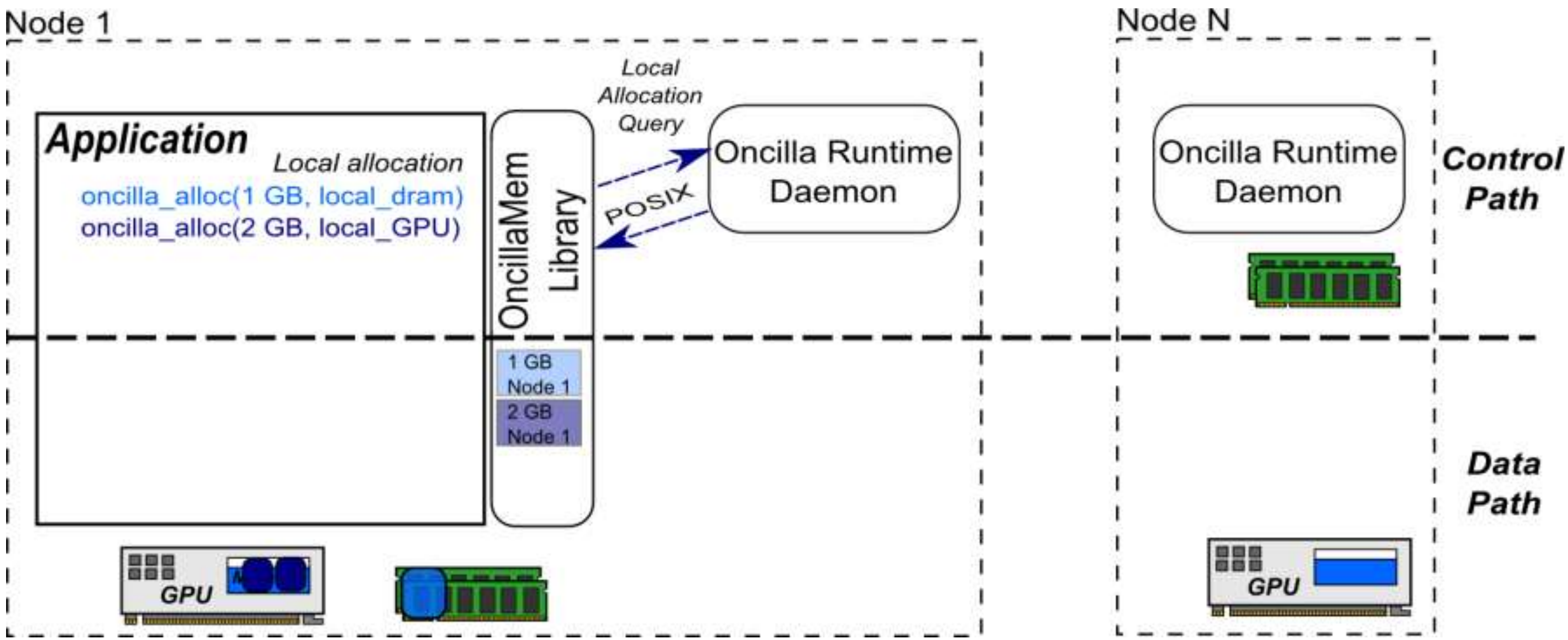
- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla – Runtime Allocation and Copy Example



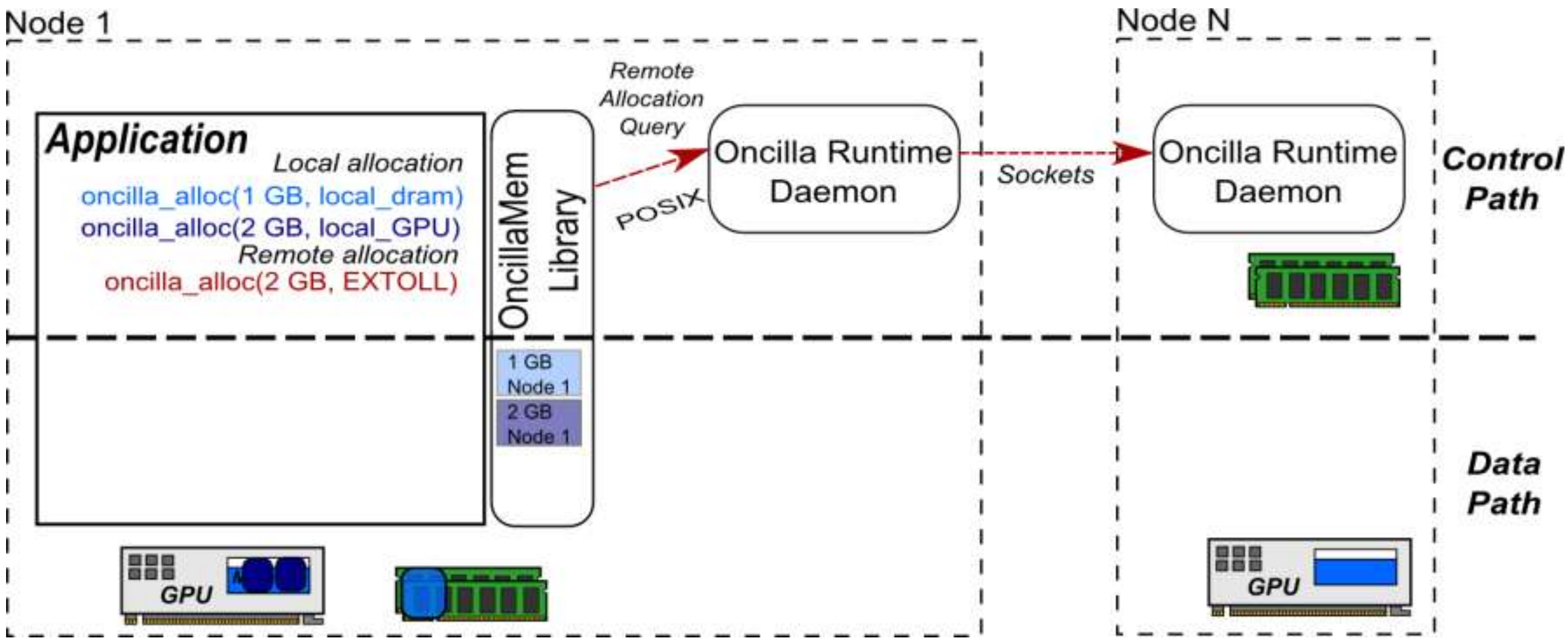
- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla – Runtime Allocation and Copy Example



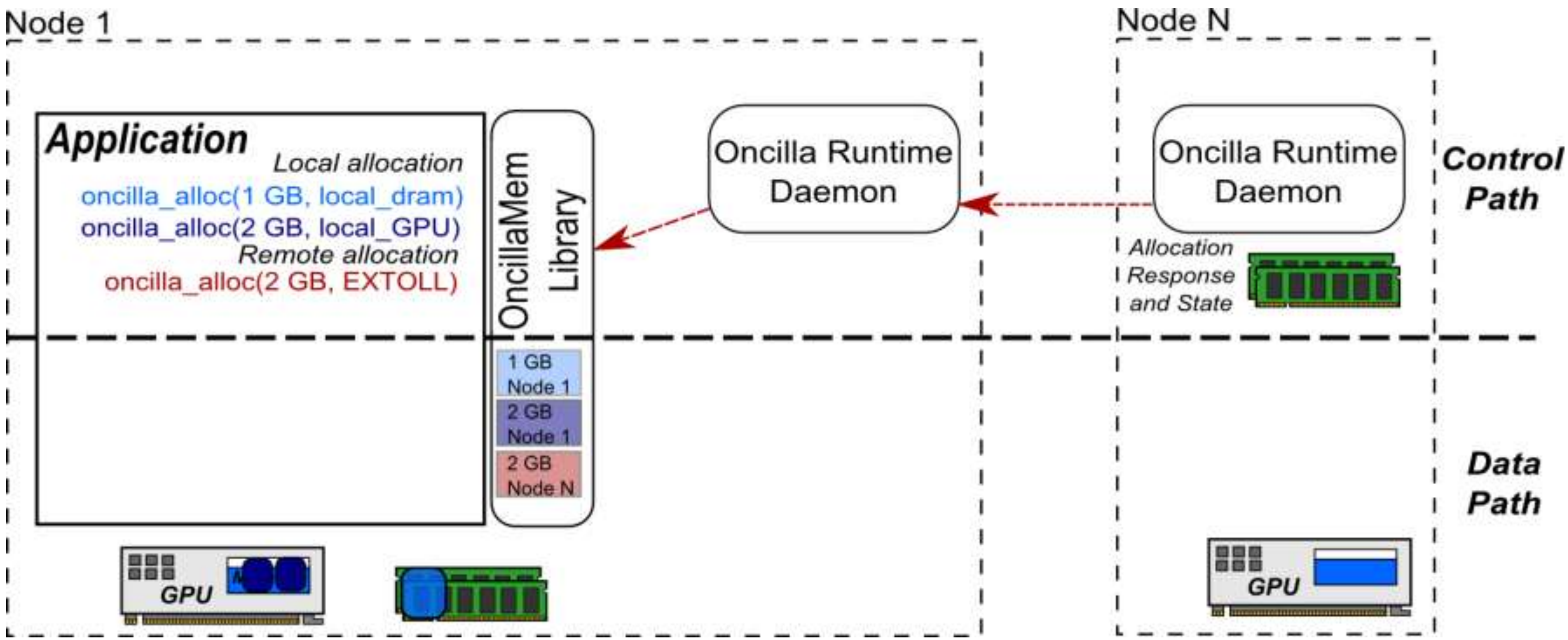
- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla – Runtime Allocation and Copy Example



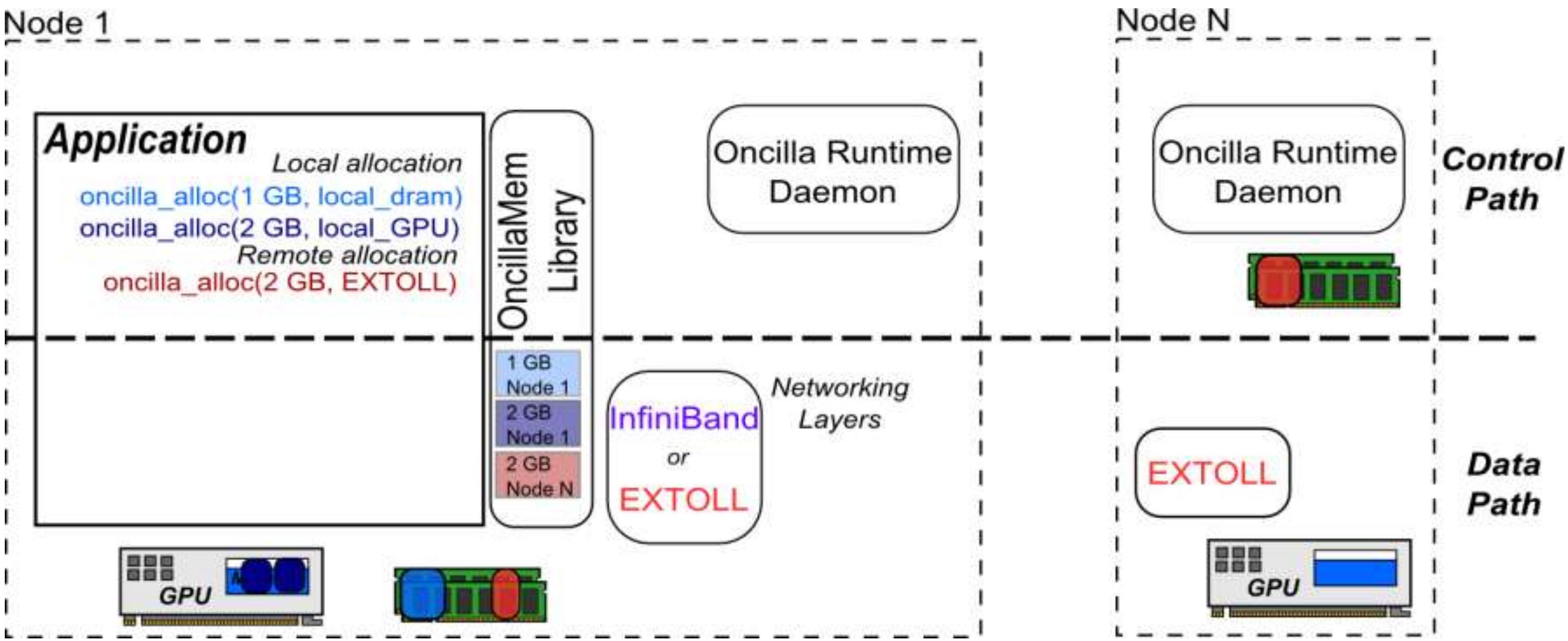
- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla – Runtime Allocation and Copy Example



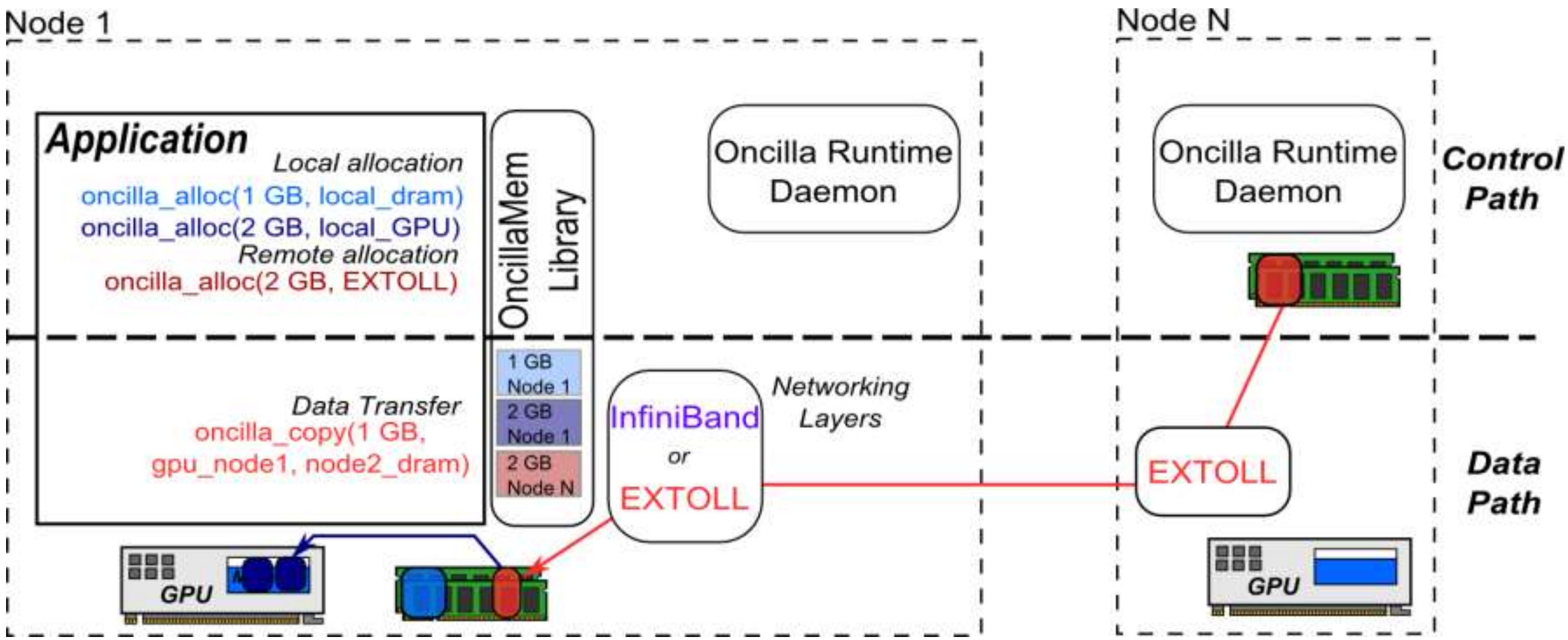
- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla – Runtime Allocation and Copy Example



- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla – Runtime Allocation and Copy Example



- The runtime is currently built around POSIX messages (to/from the application) and sockets between nodes
- The user makes a library call to allocate memory; remote memory also spawns a local and remote thread to handle data movement between the nodes
- The library keeps track of an allocation's source and destination buffer sizes and relevant information to perform oncilla_copy.

Oncilla Infrastructure

- Two node cluster prototypes
  - 12-16 GB of DRAM
  - NVIDIA C2070 GPUs
- EXTOLL cluster
  - Network adapters and fabric developed by University of Heidelberg, Germany
  - AIC custom blades
  - Galibier Virtex 6 prototypes
- IB cluster based on KIDS
  - Mellanox QDR IB adapter
  - Dual-socket Intel Xeon X5660

Oncilla Networking Support – EXTOLL RMA vs. InfiniBand

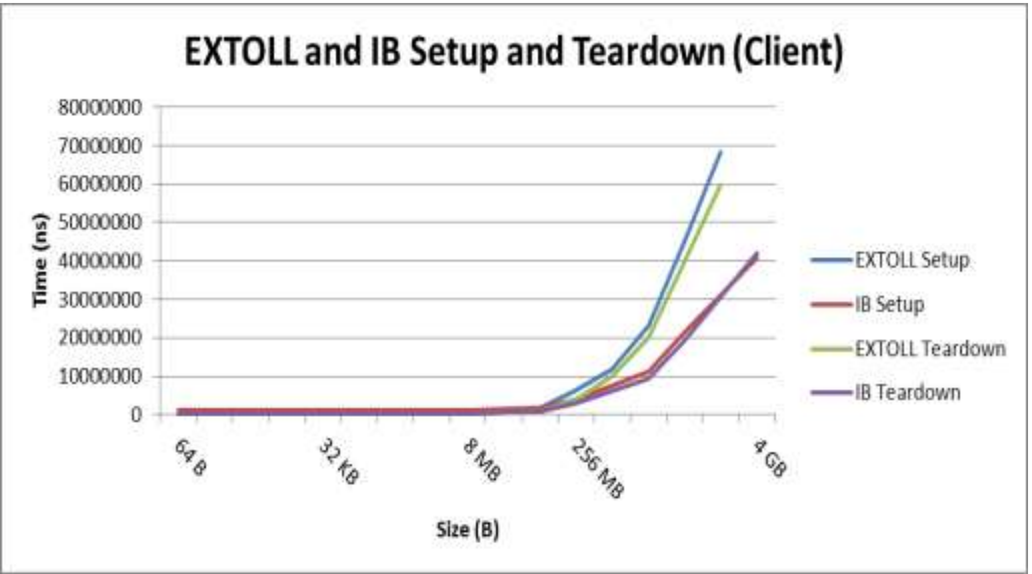
EXTOLL RMA (Remote Memory Access)

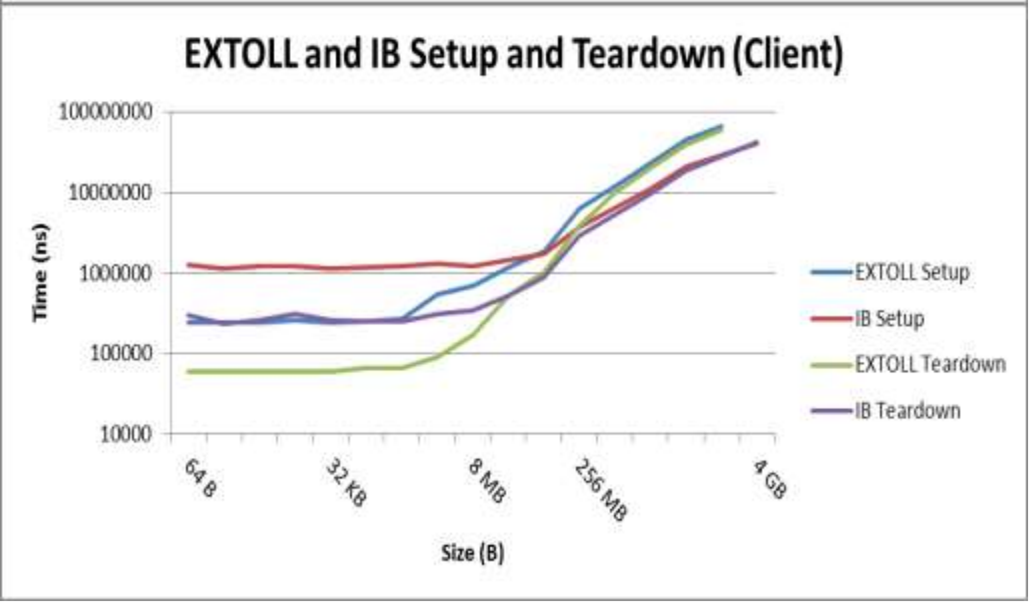
- Advantages:
  - Lower registration costs for small messages
  - Lower latency for small messages
- Disadvantages:
  - FPGA prototype (limited BW)
  - Small ecosystem of users
  - Point-to-point only (mesh or torus)

InfiniBand RDMA (Remote Direct Memory Access)

- Advantages:
  - High bandwidth, especially for large messages (up to 120 Gbps for FDR)
  - Defaault HPC interconnect; large user community
- Disadvantages:
  - High setup costs for small messages and frequent registrations
  - Highest performance comes from using IB verbs stack

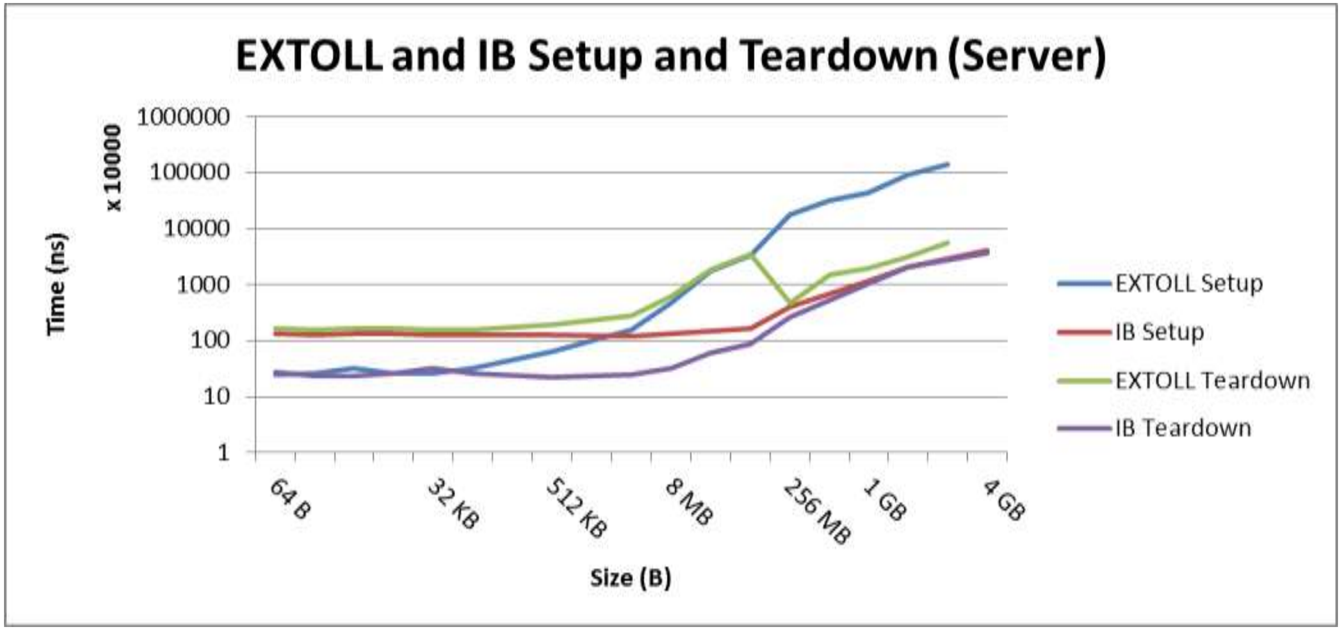
EXTOLL and InfiniBand Comparison – Client Setup





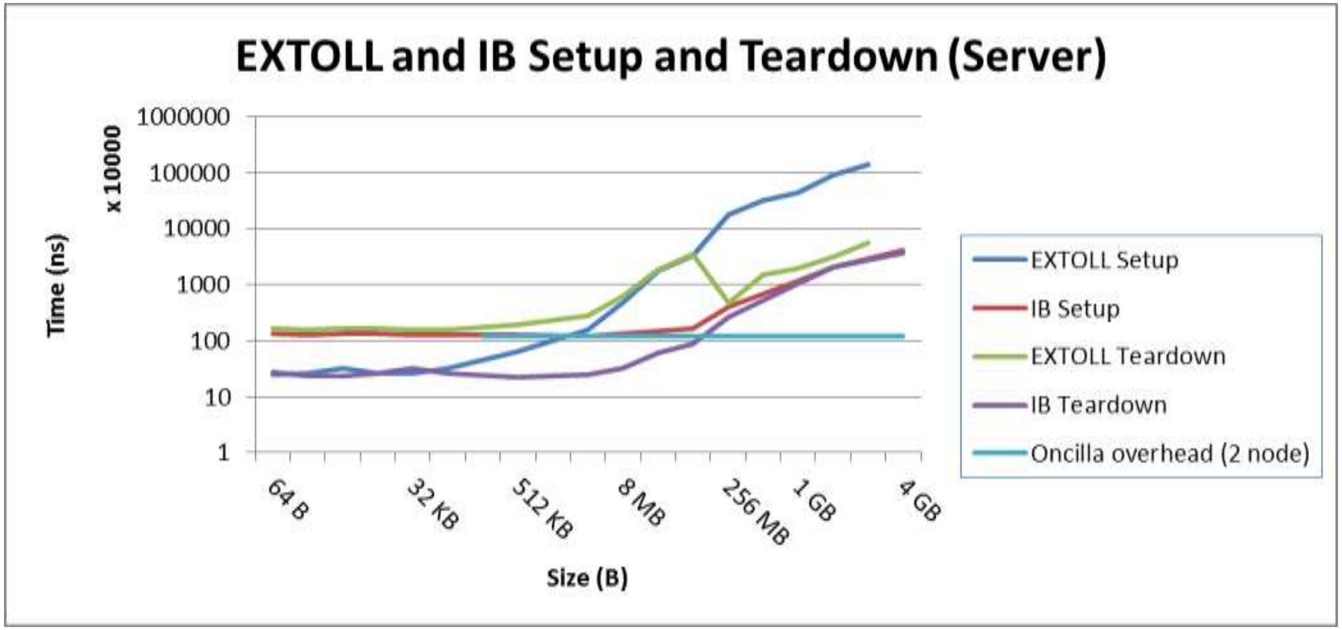
- EXTOLL client setup/teardown outperforms IB at sizes up to 32 MB
  - 242 µs up to 68.3 ms for setup (64 B – 3 GB)
  - 59.1 µs to 59.7 ms for teardown
- IB scales better for larger sizes of allocation and deallocation
  - 1.3 ms up to 40.8 ms for setup (64 B – 4 GB)
  - 301.4 µs to 42.2 ms for teardown

EXTOLL and InfiniBand Comparison – Server Setup



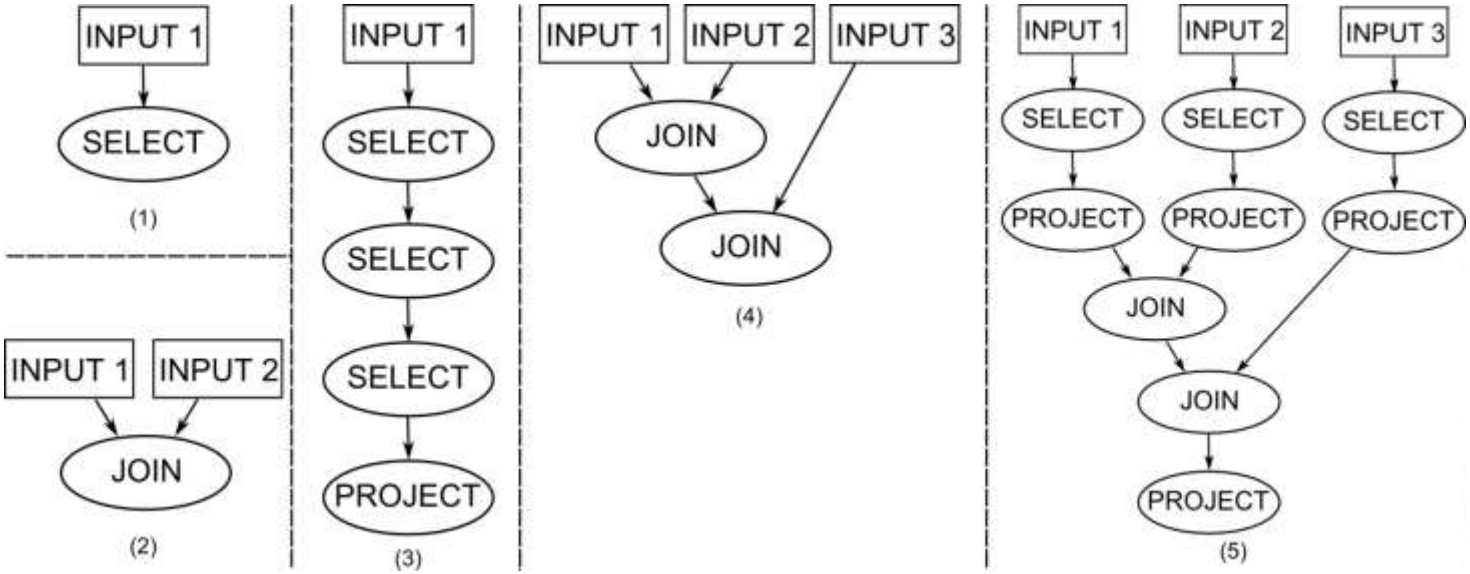
- EXTOLL is good at quick, small allocations
  - 250 μs vs. 1.6 ms (IB) for 64 B allocation
- Again, IB is faster for larger allocations and deallocations
  - 2.12 ms for IB vs. 4.5 s for EXTOLL (2 GB allocation)
  - IB stack includes huge page support while EXTOLL does not
- Most overhead is tied up in pinning/unpinning pages

Server Setup – Oncilla Runtime Overhead (2 Node)



- Oncilla adds 1.1 to 1.2 ms of overhead to existing setup costs
  - Current model has simplistic allocation due to limited nodes
  - Overhead consists of POSIX messages from application to daemon and socket call to remote daemon
  - For comparison, IB server setup for 8 MB allocation: 1.3 ms; EXTOLL server setup: 4.7 ms

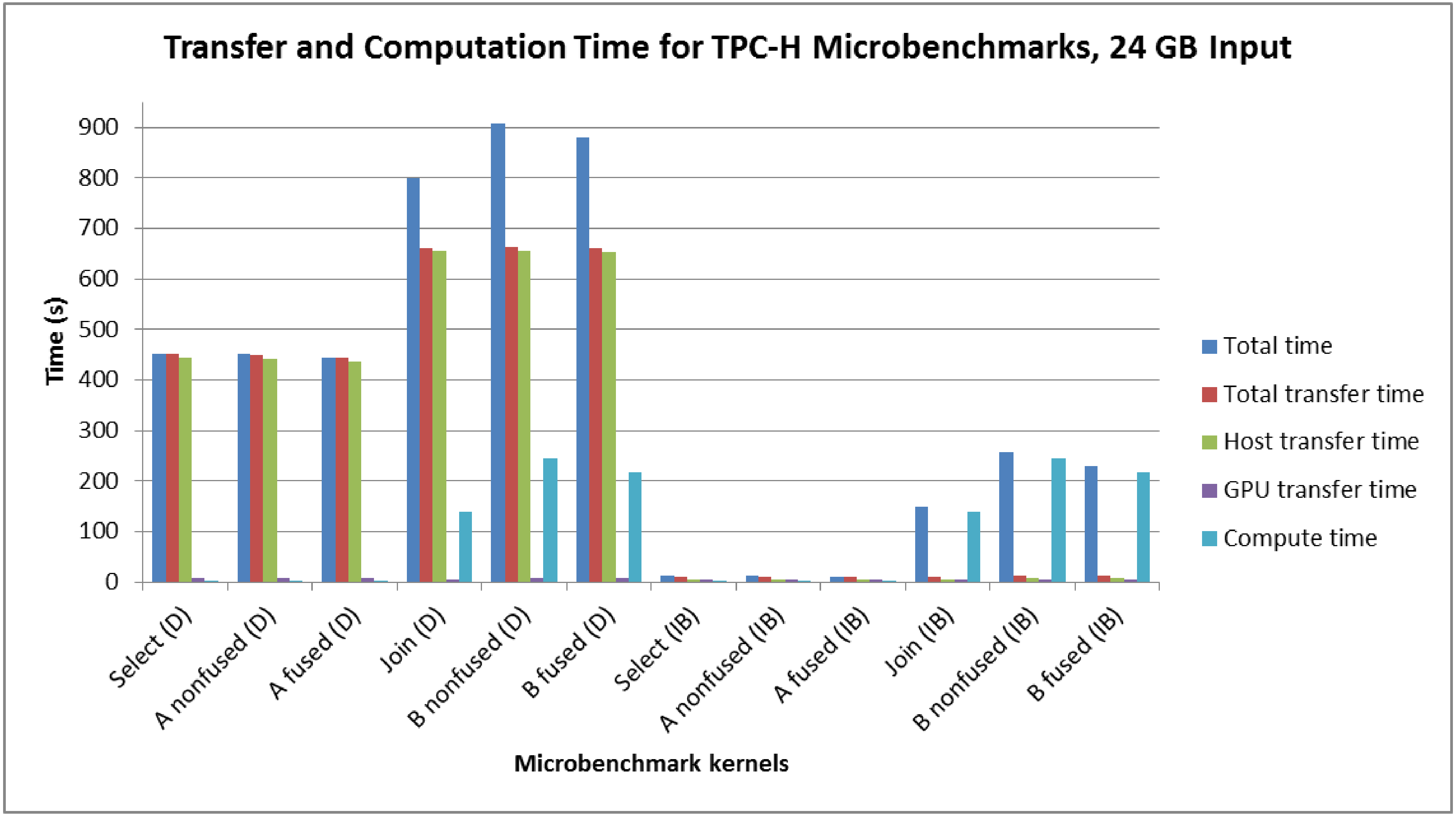
TPC-H Application Microbenchmarks



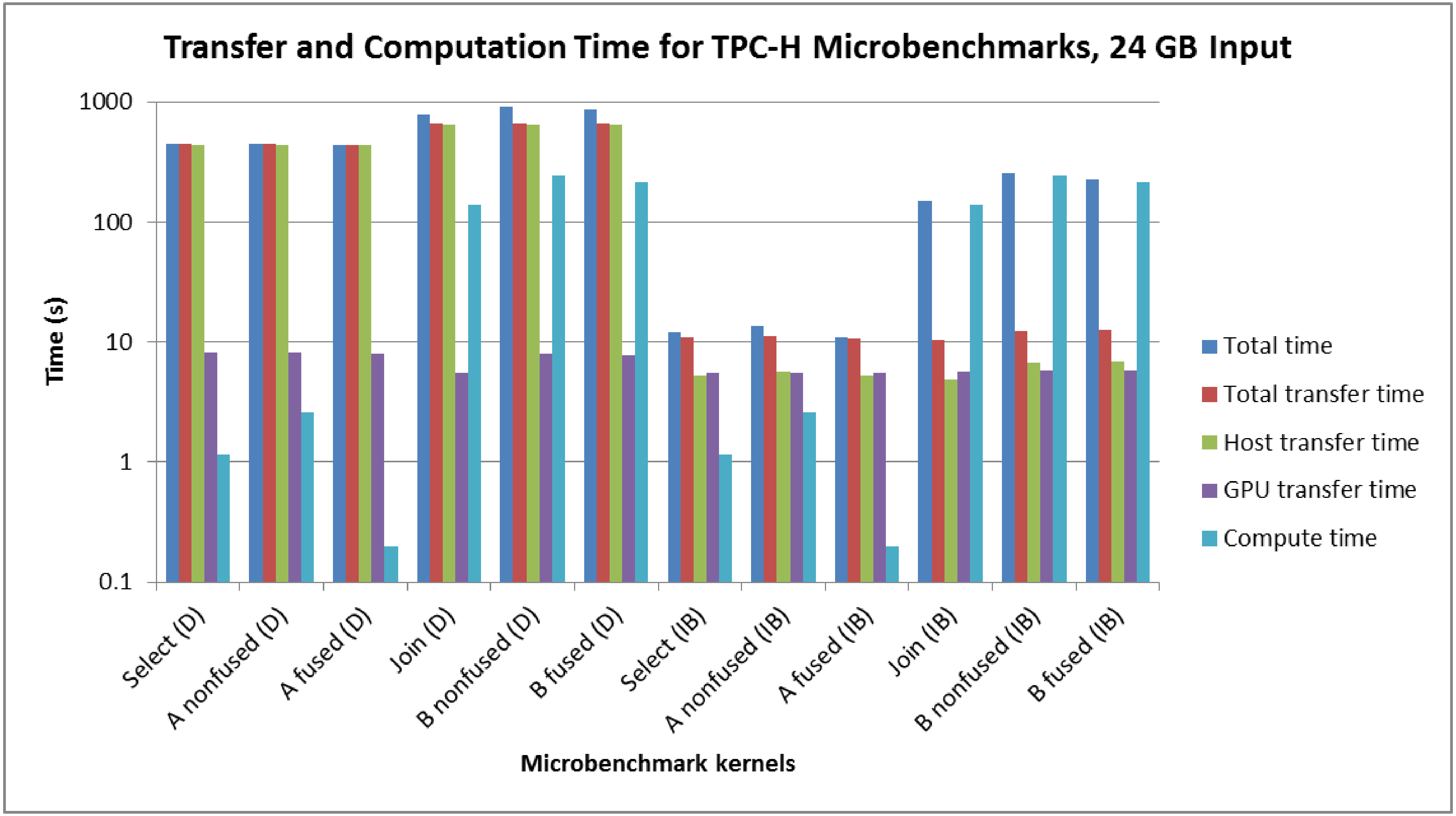
- Simple select and join along with combined operations representative of the TPC-H suite (and data warehousing operations)
  - Join operations are global memory-intensive since output can be 10-20x the input
  - Each benchmark has a normal and a "fused" version that runs faster and uses less global memory by combining operations [7]

[7] H. Wu, et al., Kernel Weaver: Automatically Fusing Database Primitives for Efficient GPU Computation. The 45th International Symposium on Microarchitecture (MICRO), 2012.

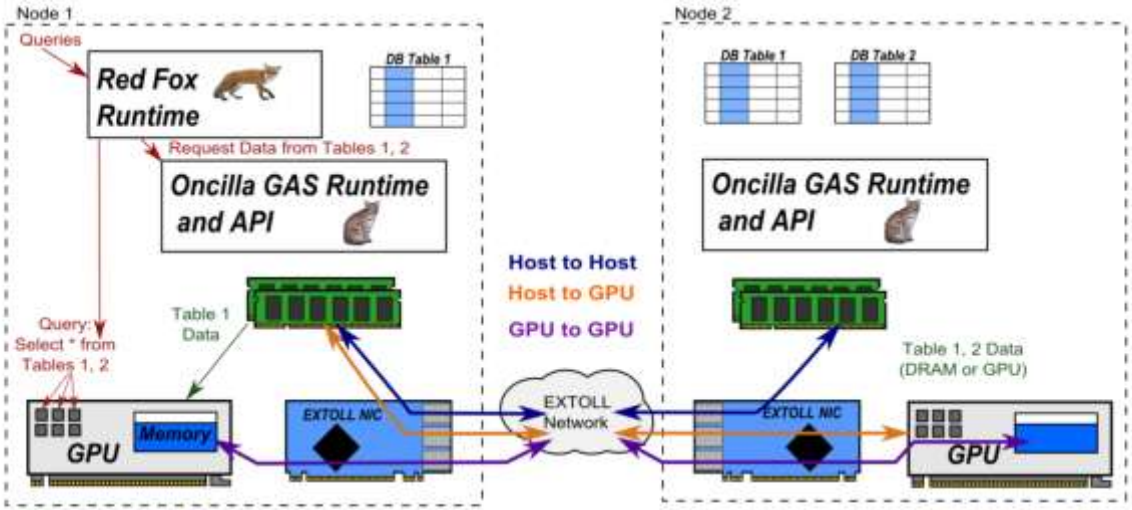
Oncilla – TPC-H Microbenchmarks (Preliminary Results)



Oncilla – TPC-H Microbenchmarks (Preliminary Results)



Questions?



Oncilla webpage: http://gpuocelot.gatech.edu/projects/oncilla-gas-infrastructure

Oncilla release: coming soon!