Hewlett Packard
Enterprise

Optimizing the Cray Graph Engine for Performant Analytics on Cluster, SuperDome Flex, Shasta Systems and Cloud Deployment

Christopher D. Rickett, Kristyn J. Maschhoff and Sreenivas R. Sukumar

Cray User Group, May 3-5, 2021

Outline

- **Background**
  - Cray Graph Engine (CGE)
- **Enabling execution on multiple platforms**
  - Performant execution across architectures
- **Recent Optimizations**
  - Database build
  - Query Engine
- **Deployment Options**
  - Native and cloud deployments
- **Benchmarks**
  - Dictionary build
  - Query execution
- **Summary**

Background : The Cray Graph Engine

- CGE is a scalable parallel graph analytics framework
  - An in-memory semantic graph database
    - Basic graph pattern search
    - Graph-theoretic (whole graph) algorithms
  - A W3C Standards inspired triplestore
    - Uses RDF Data model
    - Uses SPARQL as query language
- Scales-out and scales-up, i.e. built for “vertical scaling” based on parallel and distributed computing principles
  - Most market competitors are scale-out databases.
  - Scale-up => faster with more compute resources
- Fast turn-around on queries, brings interactivity to graph-based discovery
  - Scaling and performance enables interactive analysis of very large datasets (1 – 100s of TBs)

2012: YarcData is formed to build Big Data solutions

2014: Urika-GD – an appliance for graph analytics using the XMT architecture

2016: Urika-GX – graph engine code ported to work on Aries interconnect and x86 architecture.

2017: Urika-XC – graph engine code scales on supercomputers.

2018: Demonstrated scaling and showed CGE as the best performing database on a trillion triples.

2020 : Ongoing work on porting Graph engine ported for HPE hardware architectures such as ProLiant, SuperDomeFlex, InfiniBand, Shasta and HPE Container Platform.

Enabling CGE on Multiple Architectures

- **CGE for XC based on Cray Coarray-C++**
  - Coarray-C++ built on Cray PGAS which leverages XPMEM/DMAPP and Aries interconnect
- **Changes required for multi-platform support:**
  - Developed simplified PGAS library to remove dependence on Cray hardware and software stack
    - Based upon POSIX shared memory and one-sided MPI communication
    - Implements only functionality required by CGE, such as:
      - Creation of symmetric virtual address space across images and symmetric memory allocations/frees
      - Blocking and non-blocking puts/gets
      - Barriers
      - Collectives
      - Direct memory access amongst images on same physical node
  - Modified Coarray-C++ to leverage simplified PGAS library
- **Advantages of using POSIX shared memory and MPI**
  - Portability
  - Performance across architectures

CGE PGAS Library

- **POSIX shared memory used for symmetric heap**
  - All images use shared memory to allocate symmetric heap
    - Used for all allocations in CGE, including memory for other allocators CGE maintains
  - Enables images on same node to directly read/write to each other's memory without XPMEM

- **MPI used for PMI, synchronization and RMA**
  - MPI window feature used to enable RMA access to symmetric heap
  - CGE put/get functions translated into memcpy if on same node or MPI_Put/MPI_Get if to remote node
    - If non-blocking put/get, CGE will track outstanding RMA until next atomic_image_fence()
  - atomic_image_fence() implemented using MPI_Win_flush_local for necessary ranks
  - sync_all() implemented as an atomic_image_fence() and MPI_Barrier()

- **Using MPI enables containerization with Singularity**
  - Container, cloud, Infiniband clusters, SuperDome Flex, and Shasta Cray EX supercomputers
  - Without significant loss in performance across architectures

CGE Coarray-C++ Template File

- **Only implemented features used by CGE**
  - Removed features such as cofuture and coatomic
- **Replaced calls to Cray PGAS library with new versions to CGE PGAS**
  - New API mimics Cray PGAS which simplified replacement
- **New functionality:**
  - coexchange(): templated utility function provided that performs all-to-all communication of single data elements in a group-aggregated manner
  - Separate image fence functions for puts/gets:
    - atomic_image_put_fence()
    - atomic_image_get_fence()
  - coexchange and separate put/get fences required corresponding support be added to CGE PGAS library

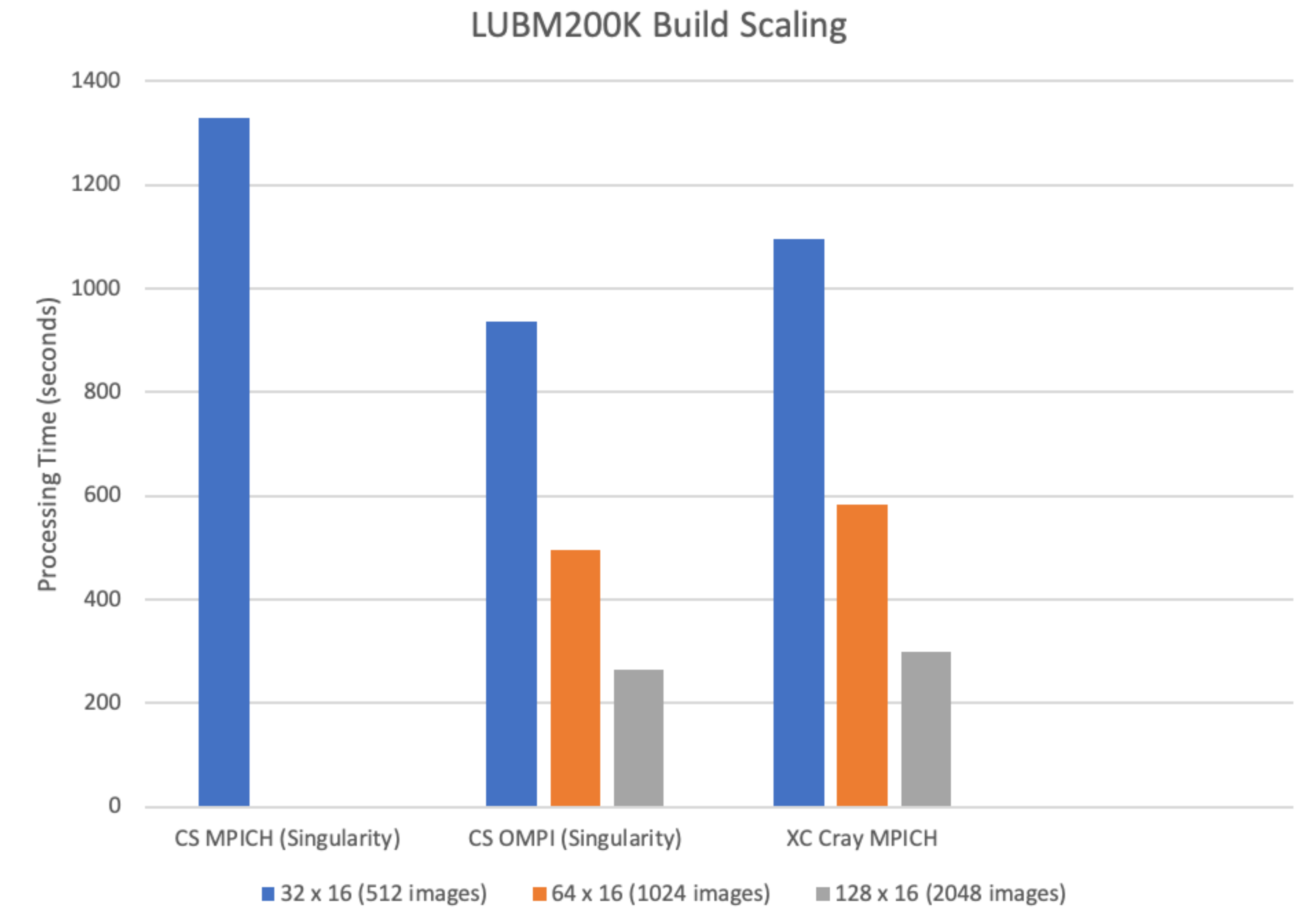
Recent Optimizations

- **Dictionary updated to support block-fetching of strings**
  - Given an array of IDs dictionary uses group-aggregated block fetching of strings
  - Used by query results writing, FILTER, Inferencing, etc.
- **Inferencer updated to use dictionary block fetching of strings**
  - Strings used to verify quads being inferred
  - Reduced inferencing time of LUBM100K on 512 images from 1796.6 seconds to 158.8 seconds
- **Scan/Join operators**
  - Optimized for shared memory machines to avoid data marshalling
  - Modified group-aggregated messaging so images hold data for consecutive images
    - Previously aggregated data from group based on one partner image per group – O(num_groups) messages
    - Reduces number of groups an image communicates with -- O(num_groups / group_size) messages
  - Reduced message sizes by returning bit-flag for 1/0 statuses rather than full words
  - Uses separate fence for remote get to prevent blocking on put of status bits
  - LUBM100K Query 9 time on 32 nodes x 16 images per node reduced from 8.6 to 7.0 seconds

Deployment Options

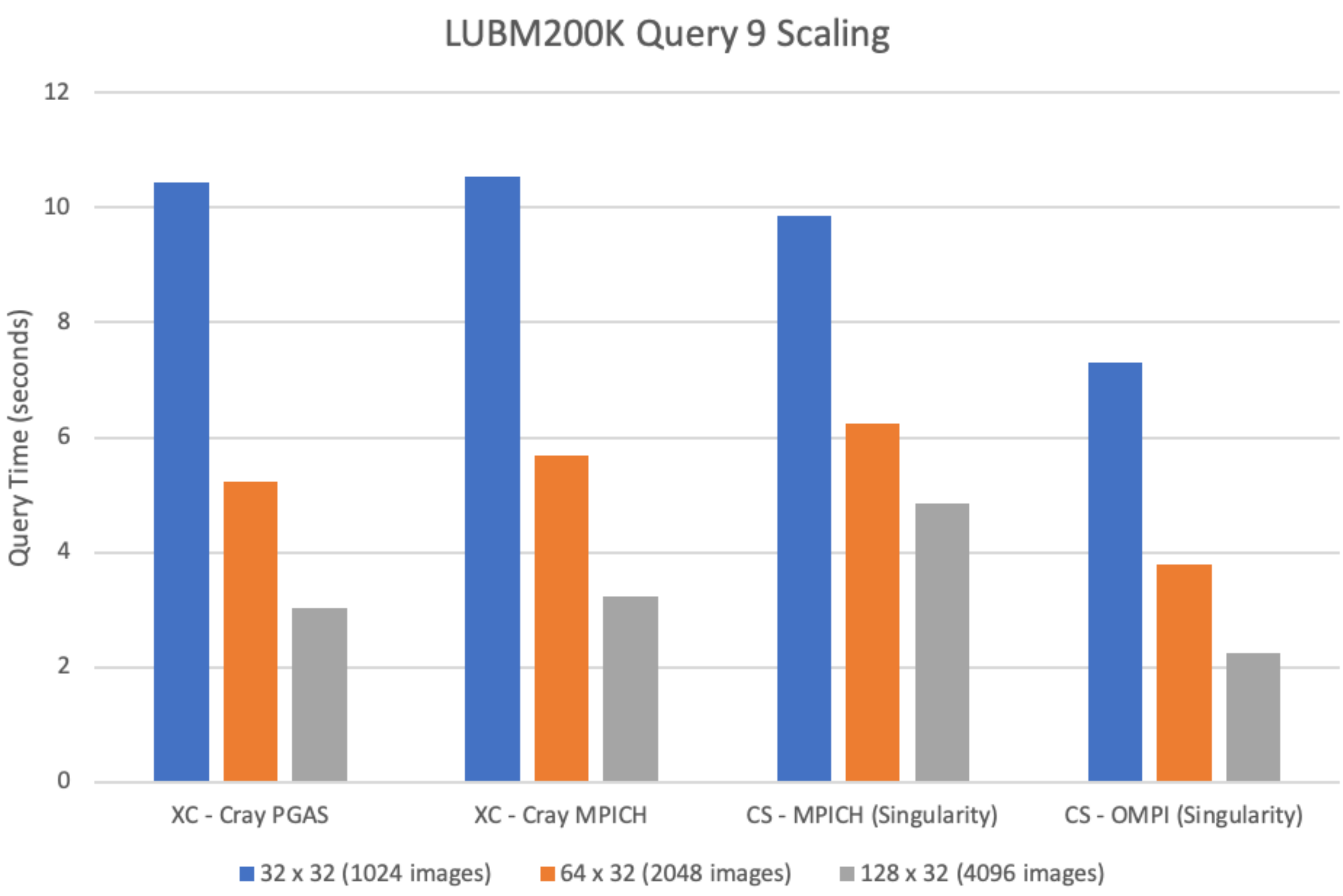
- **Build and execute natively using Cray MPICH**
  - Method used for executing on XC and EX
  - Enables optimal performance for given hardware using Cray MPI libraries
- **Containerization using Singularity**
  - Two models for MPI applications:
    - Bind – host MPI bind-mounted into container
    - Hybrid – MPI built into container and host MPI interacts with container MPI
  - CGE uses hybrid model for portability but requires container MPI to be configured for performance
    - Container includes: Mellanox OFED, UCX/OFI and MPI
  - Containers created for both OpenMPI and MPICH
    - OpenMPI 4.1 using UCX
      - Requires host OpenMPI install
    - MPICH 3.4 using OFI
      - Easy execution using slurm
      - Sometimes fails with OFI timeouts

Dictionary Build Step Scaling



- **Database build time excluding file I/O**
  - Parse, Sync, Sort, Update and Inference
- **Scales well by node count**
- **CGE using OMPI in Singularity is fastest**
- **CGE using MPICH fails to build using > 32 nodes**
  - OFI timeout errors

LUBM200K Query 9 Scaling



- **Strong scaling for all deployments except CS-MPICH**
  - 64 to 128 nodes had reduced scaling
- **Performance with Cray MPICH matches Cray PGAS**
- **CGE on CS using OMPI is fastest**
- **CGE on CS using MPICH required new coexchange**
  - OFI timeouts for small all-to-all puts/gets

Query Performance on EX, AWS and SDF

- **Benchmarked on AWS and Cray EX using LUBM100K (~18.2 billion quads)**
  - AWS parallel cluster with c5n.18xlarge nodes
  - Cray EX dual socket nodes with AMD 64-core EPYC processors
  - 32 nodes for both with either 16 or 32 images per node

Computing Platform	MPI	Query 9 time in seconds	
		512 images	1024 images
AWS efa cluster (singularity)	OpenMPI	11.60	8.40
Cray EX with Slingshot	Cray MPICH	6.17	4.24

- **SuperDome Flex (SDF) with LUBM25K (~4.6 billion quads)**

Computing Platform	MPI	Query 9 time in seconds	
		96 images	192 images
8 socket SDF (singularity)	MPICH	16.45	12.81

Summary

- **Described effort for enabling CGE to execute on multiple platforms**
  - Simplified PGAS and Coarray-C++ built on top of POSIX shared memory and one-sided MPI
  - Portability and performance using POSIX and MPI

- **Recent optimizations to CGE**
  - Dictionary build, inferencing and variable bindings

- **Demonstrated dictionary build and query performance and scaling**
  - CGE MPI based PGAS performance matches Cray PGAS
  - Containerized CGE performance on CS can match or exceed native XC performance

- **CGE now deployable either natively or in a Singularity container**
  - Native version for XC/EX enables leveraging Cray MPICH
  - Container enables easy deployment across platforms as well as in the cloud

THANK YOU

- **Questions?**