SDC 18
September 24-27, 2018
Santa Clara, CA
www.storagedeveloper.org

Accelerated Erasure Coding: The New Frontier of Software Defined Storage

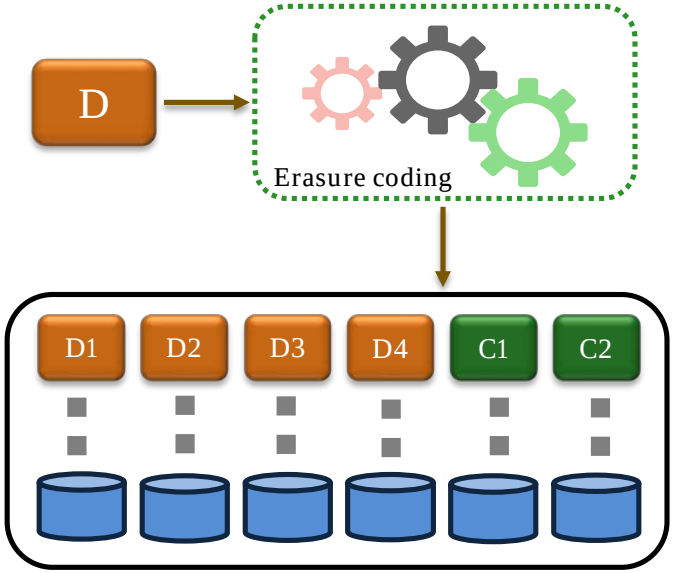
Dineshkumar Bhaskaran
Aricent – Altran Group

Introduction to Data Resiliency

- Traditional RAID and Mirroring
  - Multiple disks are used for data placement thereby improving performance and resiliency
    - High storage overhead; high rebuild times
    - Difficult to recover from co-related disk failures
- Erasure coding
  - Erasure coding is data protection method in which data is encoded to data blocks and parity blocks. These are then stored across locations or storage nodes
    - Compute intensive

Erasure Coding : A primer

- A traditional erasure code is represented as (k, m) where it encodes k data blocks with m parity blocks writes them to k+m storage nodes
- An optimal (or MDS) code can recover from any ‘m’ node failures
- A popular code is Reed-Solomon (RS). It has been successfully used in several solutions like Linux RAID-6, Google file system II, Hadoop, Facebook, etc.

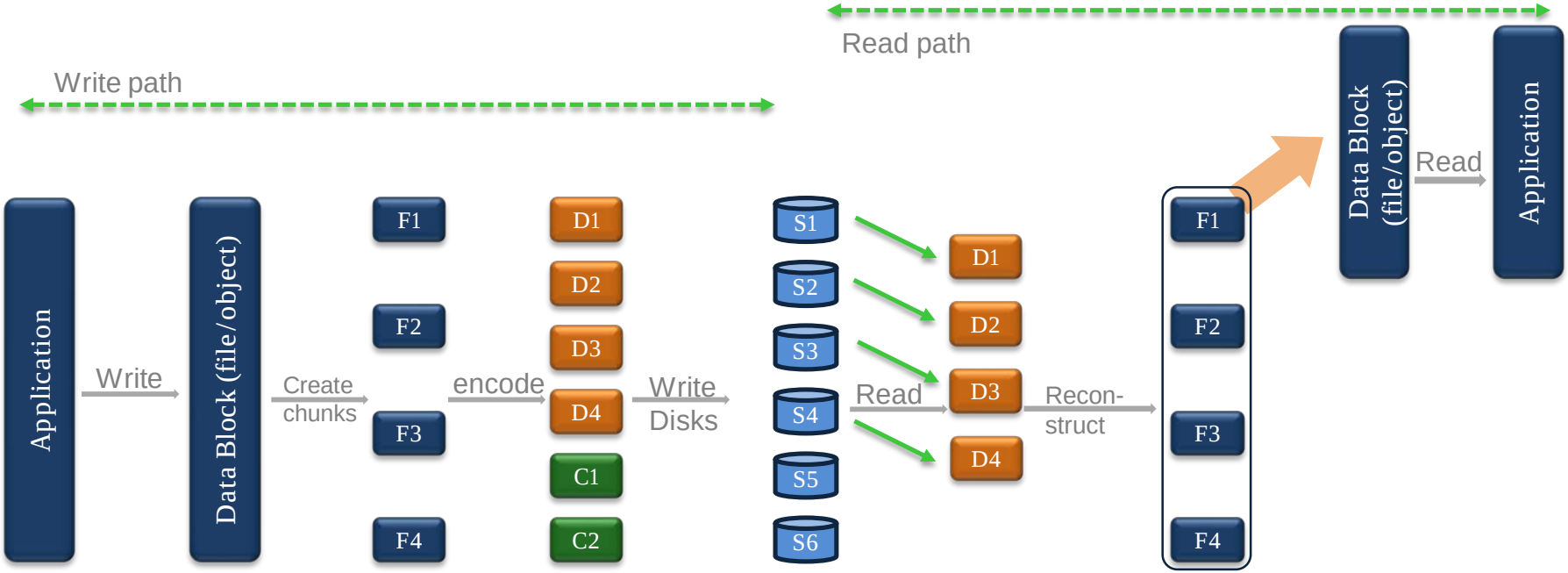


A (4, 2) erasure code

Erasure Coding : Read and Write

Traditional erasure code

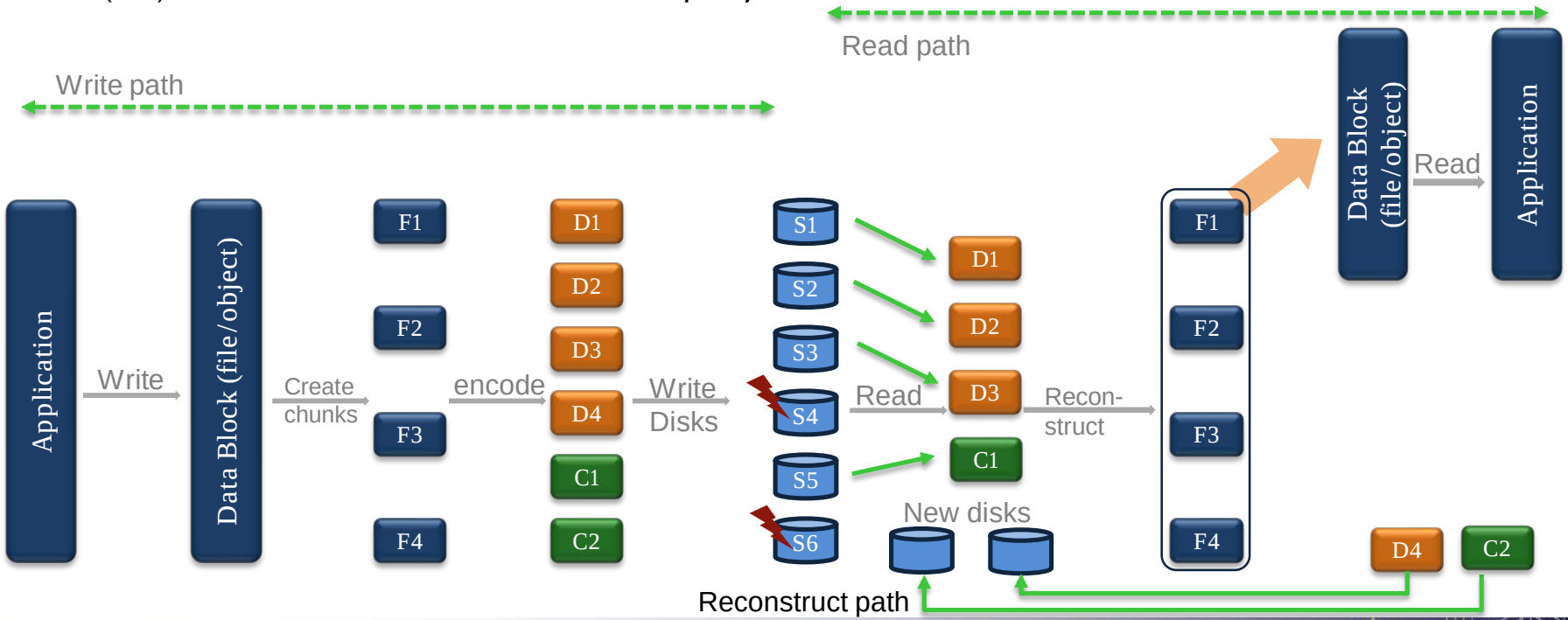
- A (4, 2) erasure code has 4 data chunks and 2 parity chunks



Erasure Coding : Read and Write

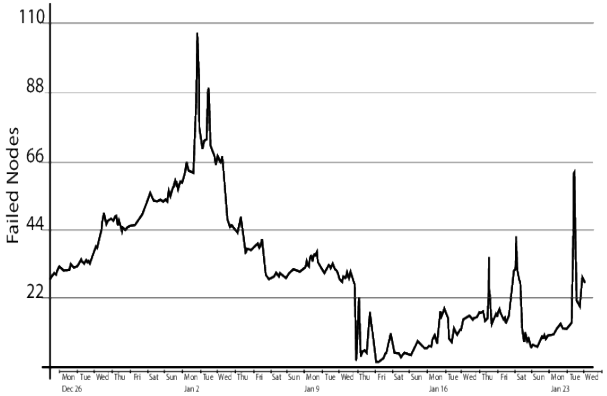
Traditional erasure code

- A (4, 2) erasure code has 4 data chunks and 2 parity chunks



Erasure Coding : Shortcomings

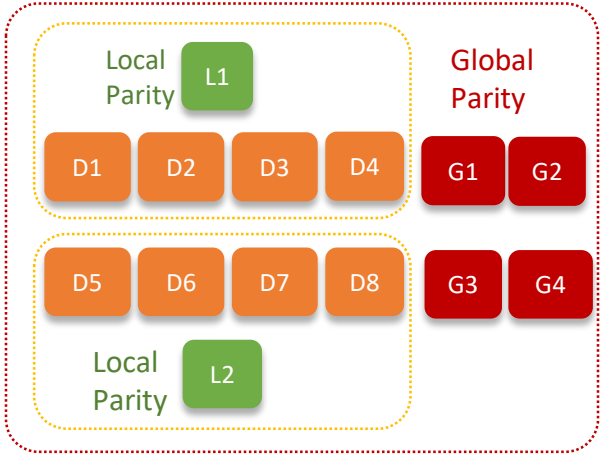
- Encode is compute intensive
  - In case of Reed Solomon a generator matrix of dimension (k+m, k) is used to create code chunks from data chunks
- Reconstruction is costly. It is triggered in case of
  - Degraded Read : This issue is caused when application receives read exception while reading a data block in a node due to software errors (hot spot effect or system updates) or hardware errors
  - Node repair : The whole node is down



Number of failed nodes in a Facebook cluster of 3,000 nodes for a month [4]

Erasure Coding : Modern approaches

- **Locally recoverable code (LRC)**
  - LRCs trade storage efficiency for speeding up the recovery process
  - LRCs use MDS code in a hierarchical manner by performing the encoding at multiple levels



Erasure Coding : Modern approaches

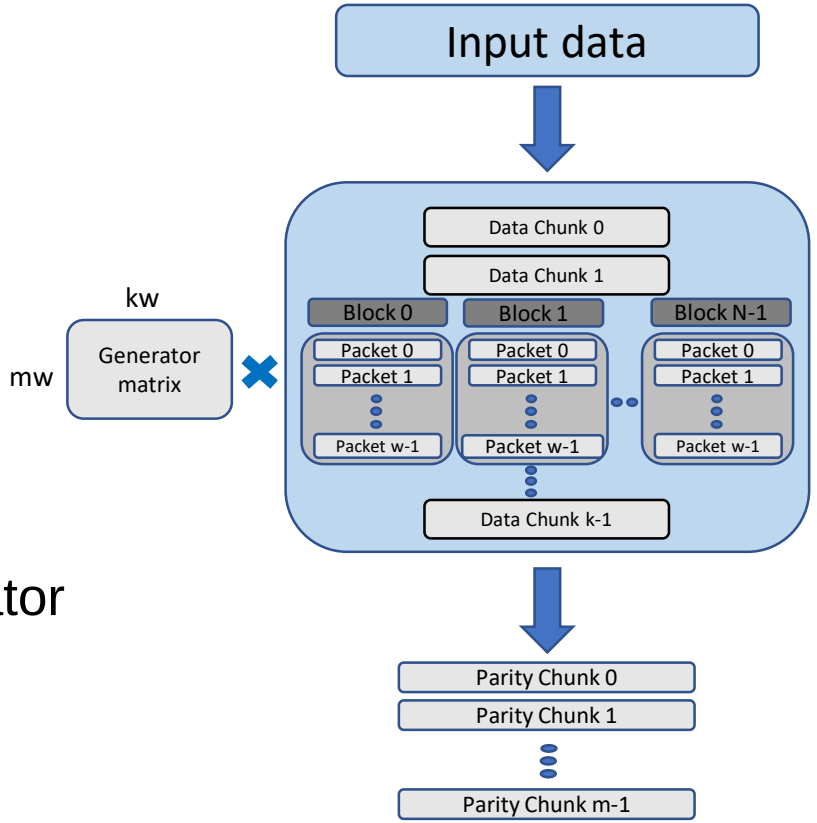
- **Regenerating codes**
  - These are mostly MDS codes represented as (n, k, d, α, β) which divides the chunks into smaller sub-chunks during the encoding process
  - Reduce the bandwidth for the repair by reducing the amount of data read from each node
  - Further classified as minimum storage regenerating codes and minimum bandwidth regenerating codes
    - Highly compute intensive

Erasure code @ Aricent – Altran Group

- Improvement in storage efficiency, Latency.
  - Employ new generation Clay Code[2]. Clay Code has
    - Least possible storage overhead
    - Least possible repair bandwidth and disk read
    - Shown 3x repair time reduction and up to 30% and 106% improvement in degraded read and write with CEPH
- Acceleration of Erasure Coding
  - Offloading the computation to GPU
    - Accelerated Cauchy RS (CRS) from Jerasure library and Clay Code
- Integrate the accelerated erasure code algorithms to CEPH.

Cauchy Reed-Solomon

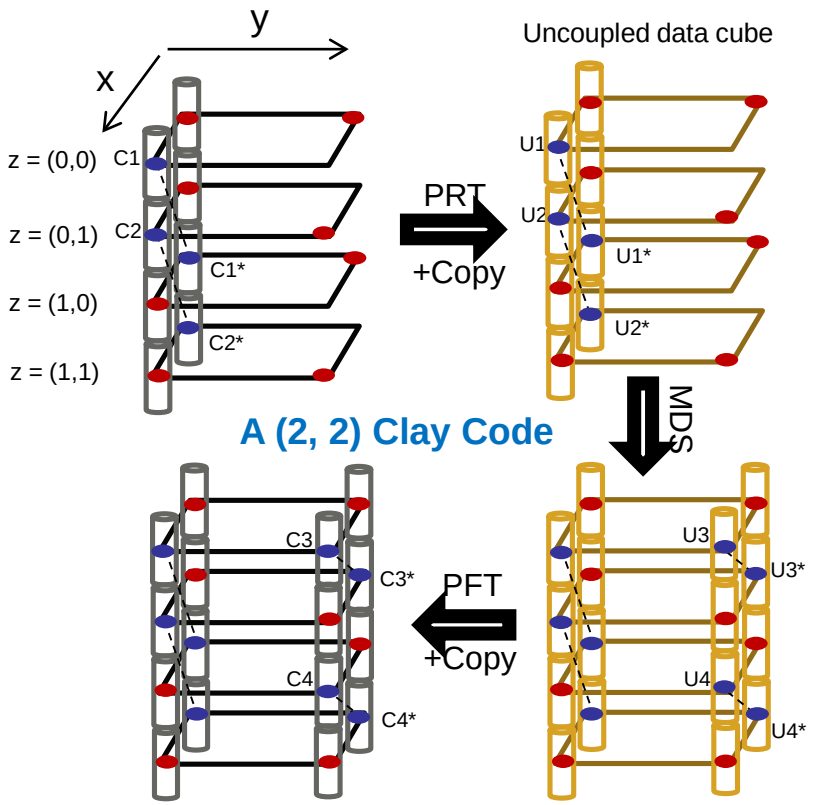
- Cauchy Reed Solomon
  - Uses Cauchy generator matrices
  - Multiplication is reduced to XOR operation
- Accelerated Cauchy Reed Solomon
  - Use of constant memory of generator matrix in GPU
  - Use of shared memory to optimize access to data in global memory



Cauchy RS erasure code[3]

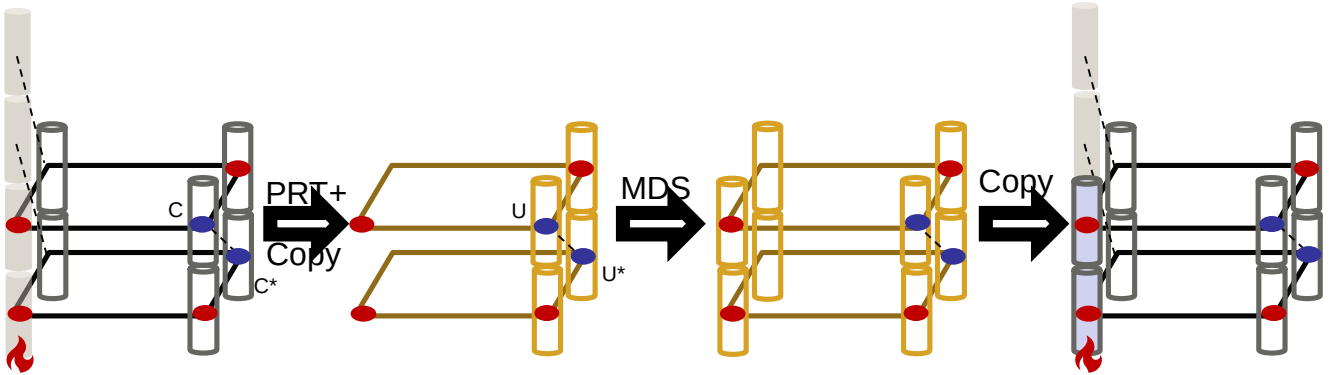
Clay Code : Construction

- Consider the (2, 2) encoding, each sub-chunk is represented using a point in plane. Sub-chunks are further classified as coupled (blue dots) and uncoupled (red-dots)
  - Using uncoupled pairs copied as is, a Pairwise Reverse Transform (PRT) is used on paired sub-chunks to obtain elements of uncoupled data cube (cube on RHS). A MDS code is used to get rest of uncoupled data cube
  - Using newly constructed uncoupled data cube a Pairwise Forward Transform (PFT) is applied to obtain the code chunks. Both PRT and PFT are (2, 2) MDS codes



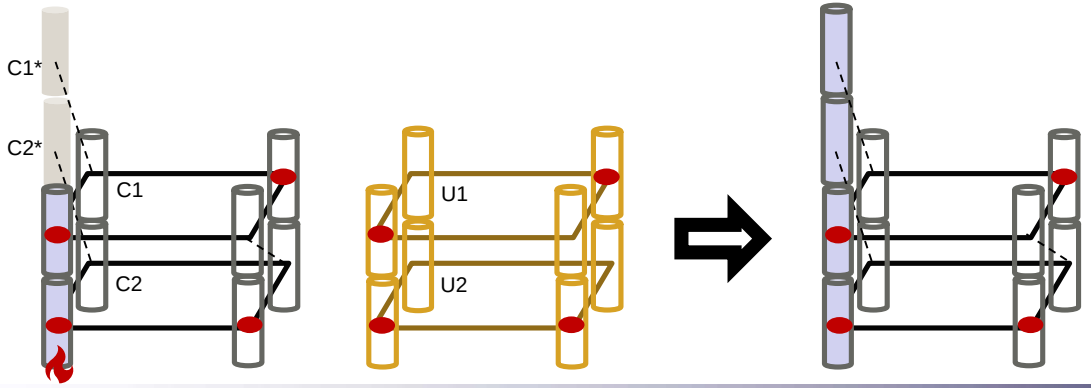
Clay Code : Decode/Recovery Process

- Consider the following single data node erasure case
  - Uncoupled data cube is created using PRT and copying the unpaired sub-chunks
  - MDS decode is performed on the planes selected for recovery and uncoupled sub-chunks are copied



Clay Code : Decode/Recovery Process

- With clay code construction any two sub-chunks in the set {U, U*, C, C*} can be recovered from the remaining two sub-chunks using PFT. Here C1* is computed from C1, U1 and C2* from C2, U2
- The repair bandwidth is reduced in this method since data from only 2(half) Z-planes are used for the recovery process

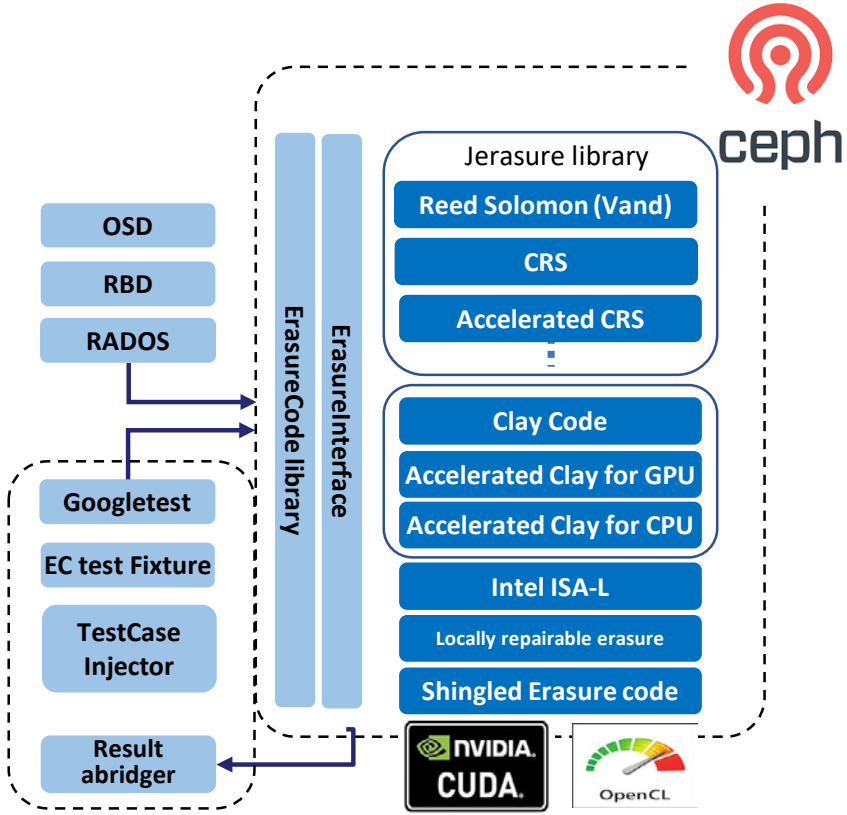


Clay Code : Enhancements

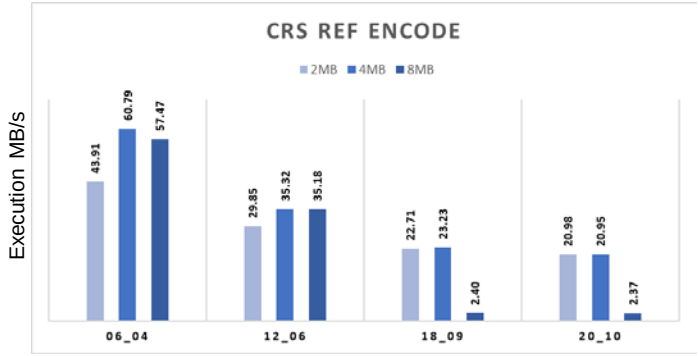
- Use of accelerated Cauchy RS
  - Clay code uses MDS codes for performing PFT and PRT through existing Erasure code infrastructure in CEPH. A version of earlier accelerated Cauchy RS is used for PFT and PRT
  - Multiple memory allocation (both CPU and GPU side) and related copying were involved with CEPH erasure code infrastructure. These were optimized by removing redundant operations
- Optimized memory access and separate GPU kernel for PFT and PRT
  - Clay code construction uses data copy and various transforms to create intermediate and final results. Complete clay operations were moved to GPU space while using CUDA/OpenCL primitives to achieve the copy operations
  - An optimized and independent (2, 2) erasure code CUDA/OpenCL implementation is used for PFT and PRT

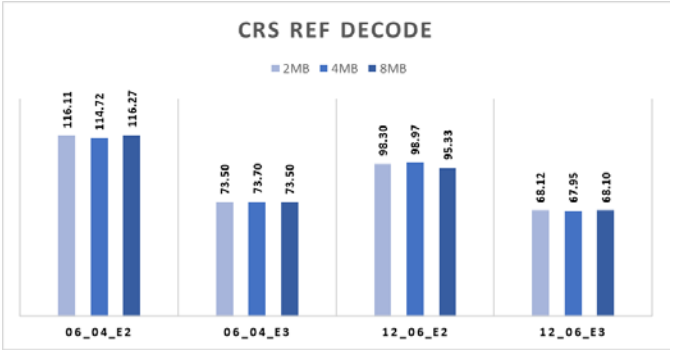
Environment

- Hardware
  - 16 core Intel(R) Xeon(R) CPU E5-2660 @ 2.20GHz with 64GB ram
  - NVIDIA GTX 1080
- Software
  - CEPH 13.1.0 (mimic)
  - CUDA 8.0 (driver 384.111)
  - Intel OpenCL 2.1 for CPU



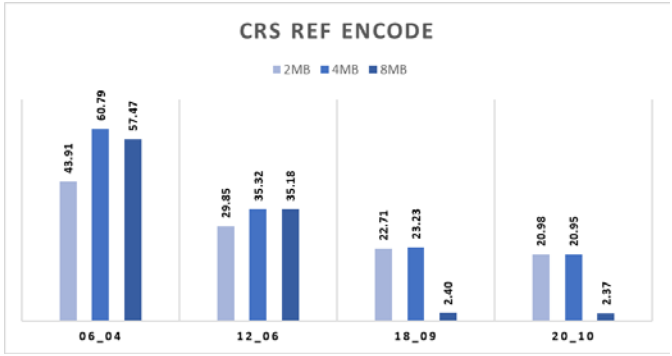
Results – CRS REF Performance

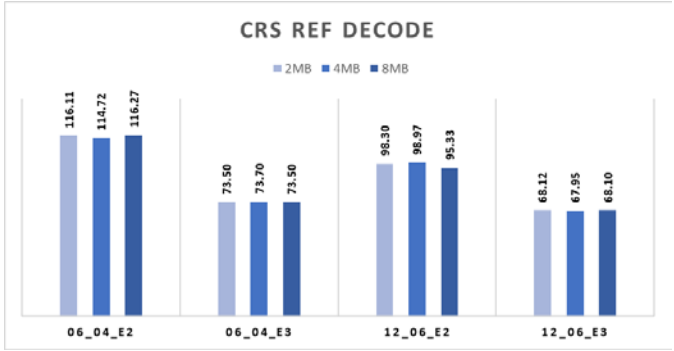




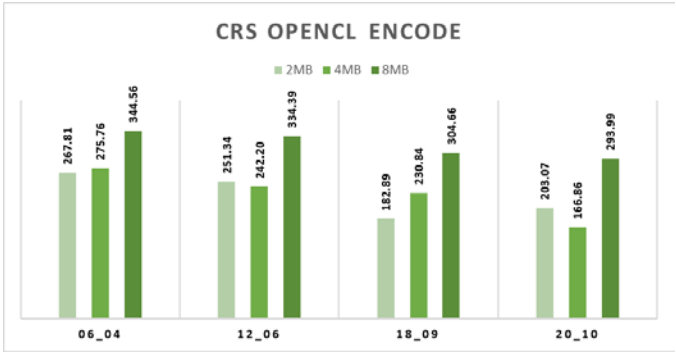
Encode and decode performances for various (k, m) values with different chunk sizes for CRS algorithm.

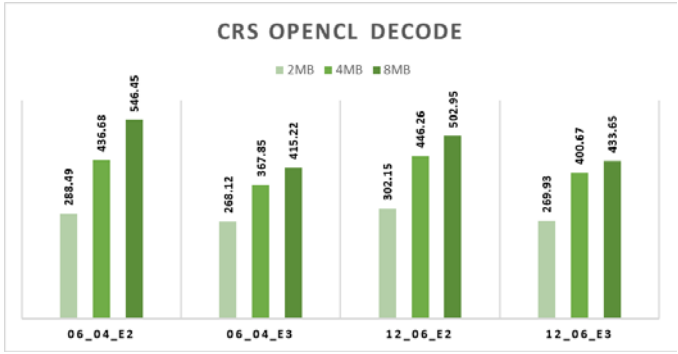
Results – CRS REF with OpenCL Performance





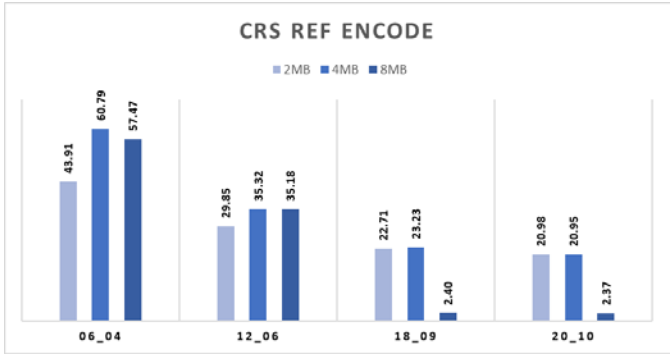
Encode and decode performances for various (k, m) values with different chunk sizes for CRS algorithm.

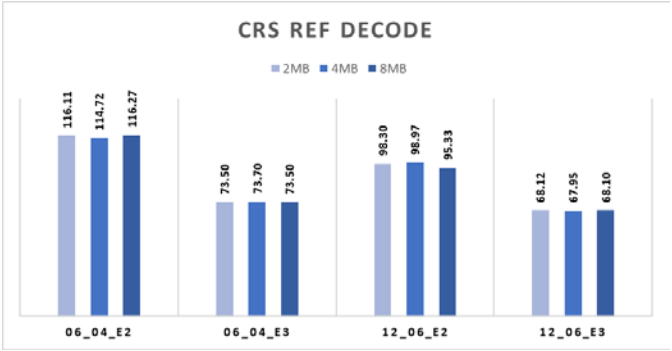




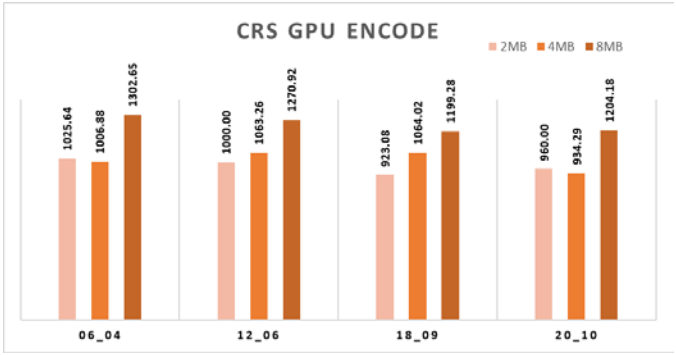
CRS: Encode and decode performance decrease with higher (k, m) values. In case of decode the performance declines with no. of erasures similar to REF.

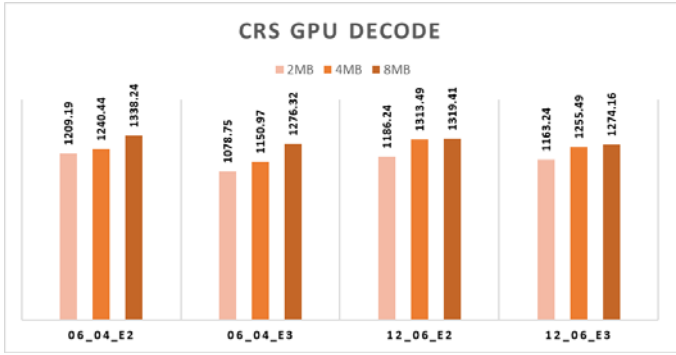
Results – CRS REF with GPU Performance





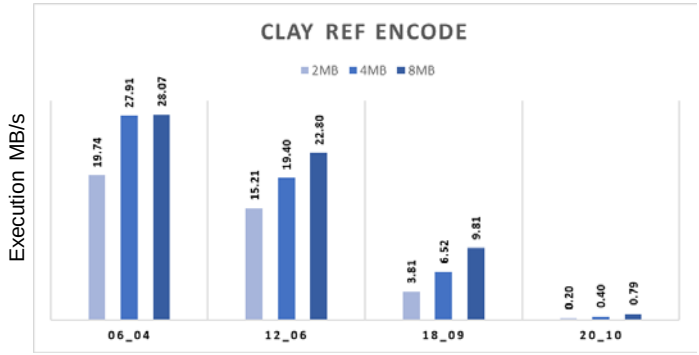
Encode and decode performances for various (k, m) values with different chunk sizes for CRS algorithm.

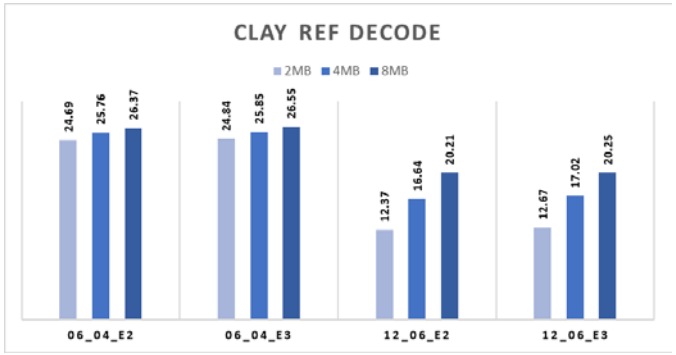




CRS: Encode and decode performance are fairly consistent with variation in (k,m) and number of erasures

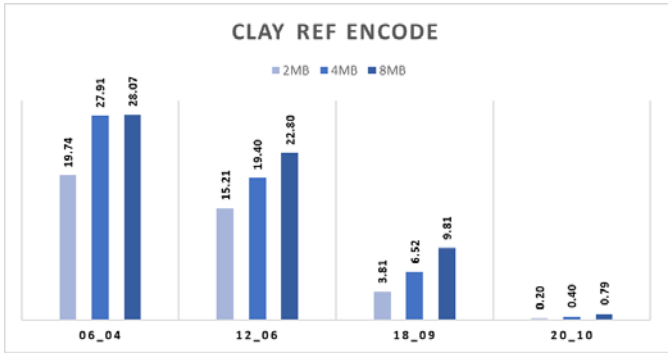
Results – CLAY REF Performance

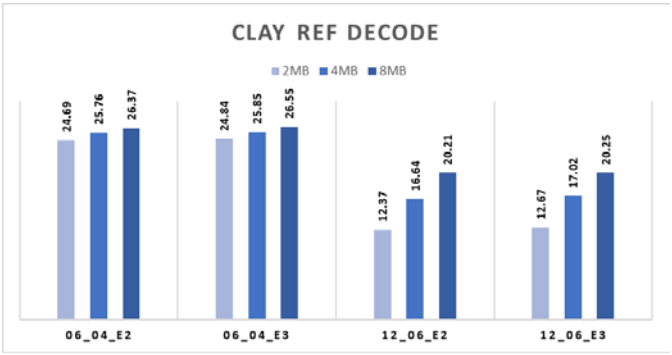




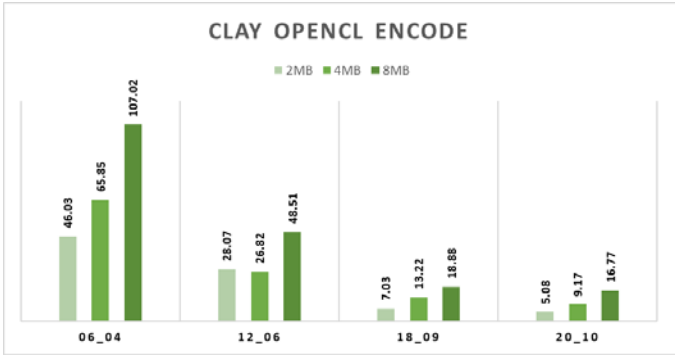
Encode and decode performances for various (k, m) values with different chunk sizes for CLAY algorithm.

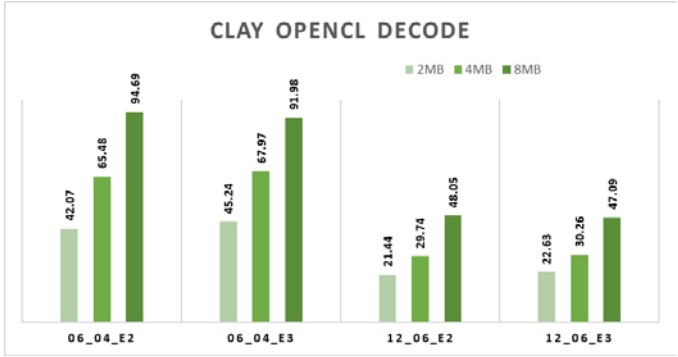
Results – CLAY REF with OpenCL Performance





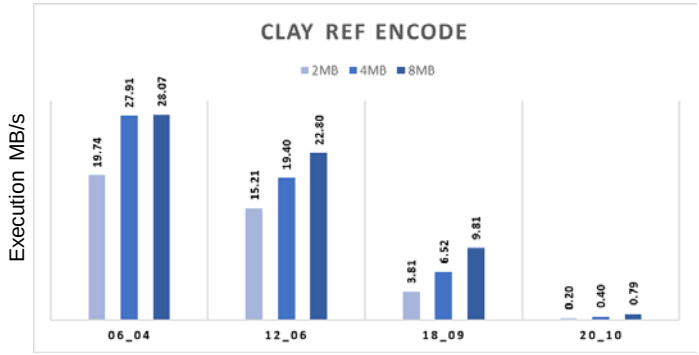
Encode and decode performances for various (k, m) values with different chunk sizes for CLAY algorithm.

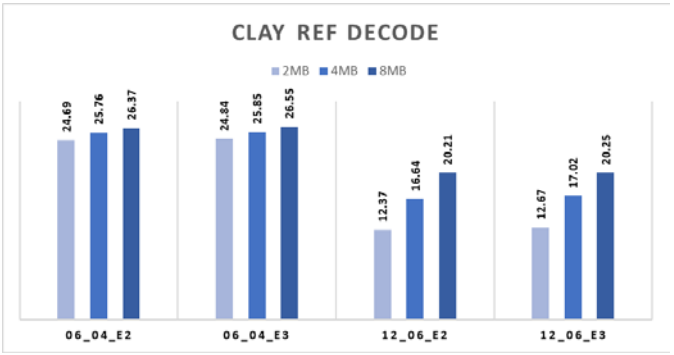




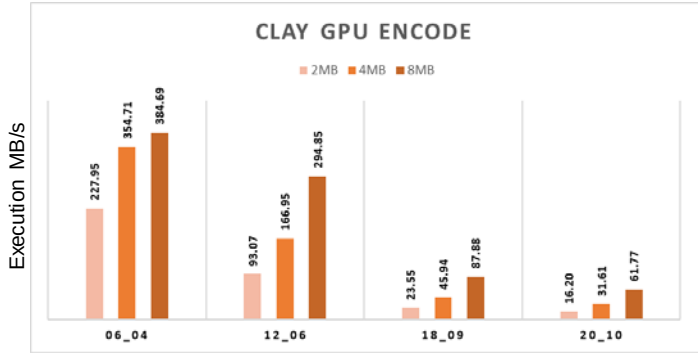
CLAY: Encode performance decrease with higher (k, m) values. In case of decode the performance is consistent with no. of erasures similar to REF.

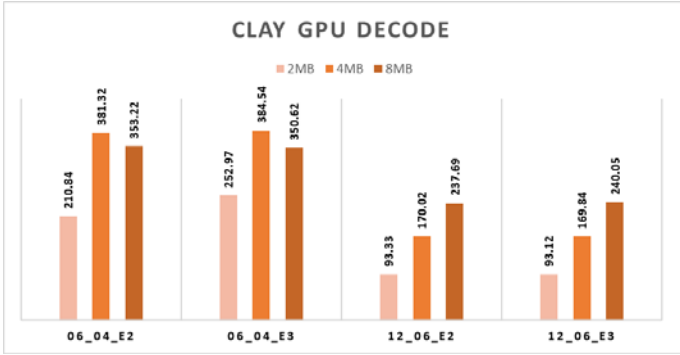
Results – CLAY REF with GPU Performance





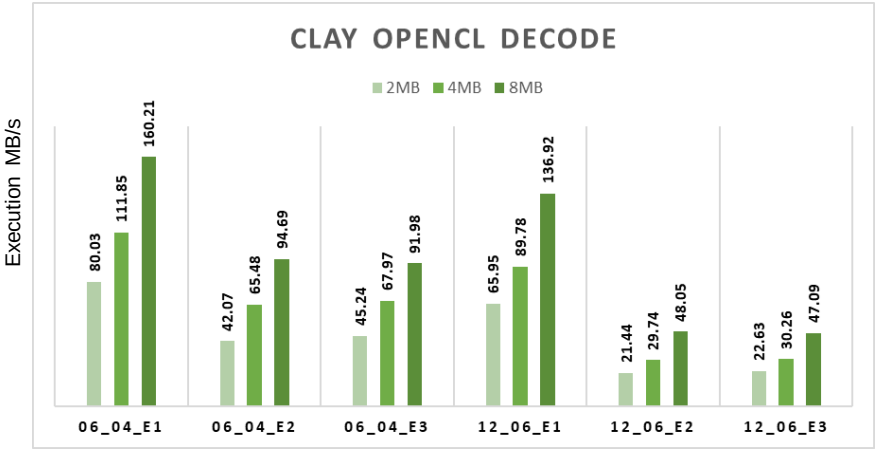
Encode and decode performances for various (k, m) values with different chunk sizes for CLAY algorithm.



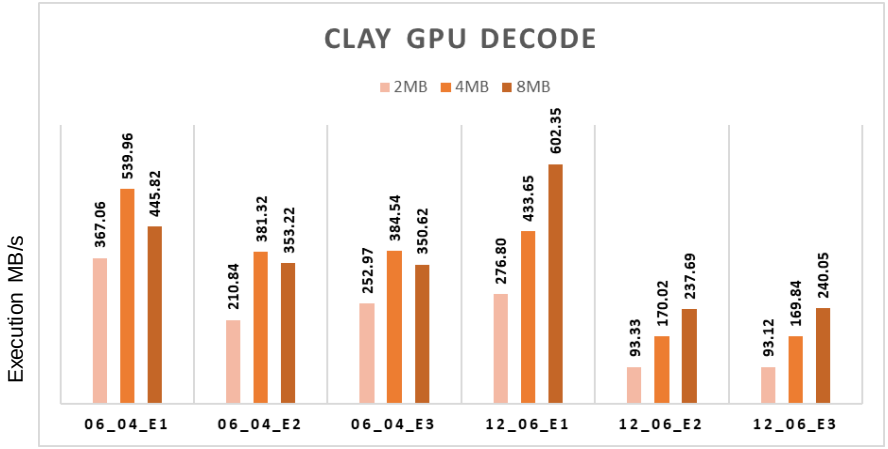


CLAY: Encode performance decrease with higher (k, m) values. In case of decode the performance is consistent with no. of erasures.

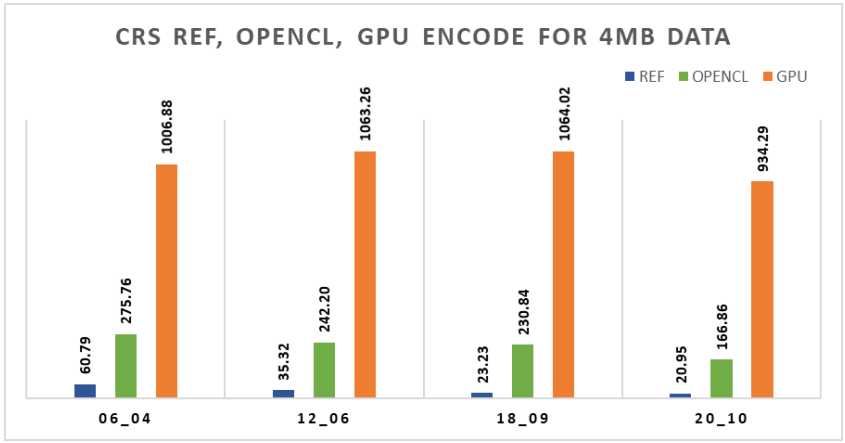
Results – CLAY Decode performance



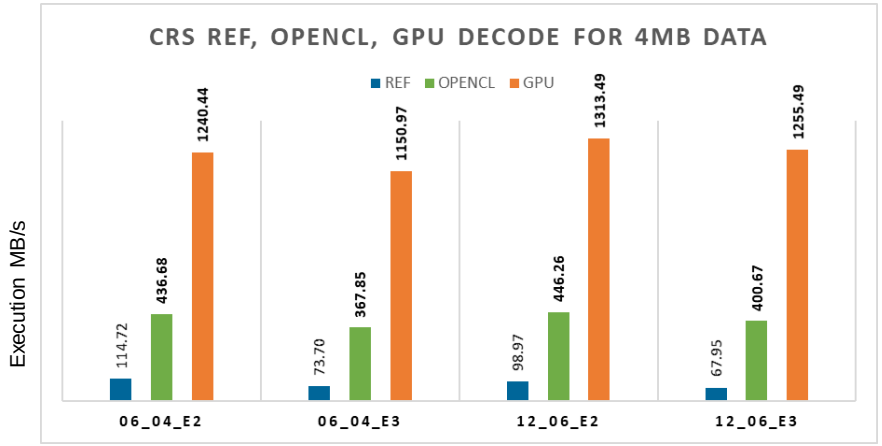
(12, 6) Decode with one erasure is ~3x and ~2.5x faster in OpenCL and GPU respectively.



Results – CRS performance summary

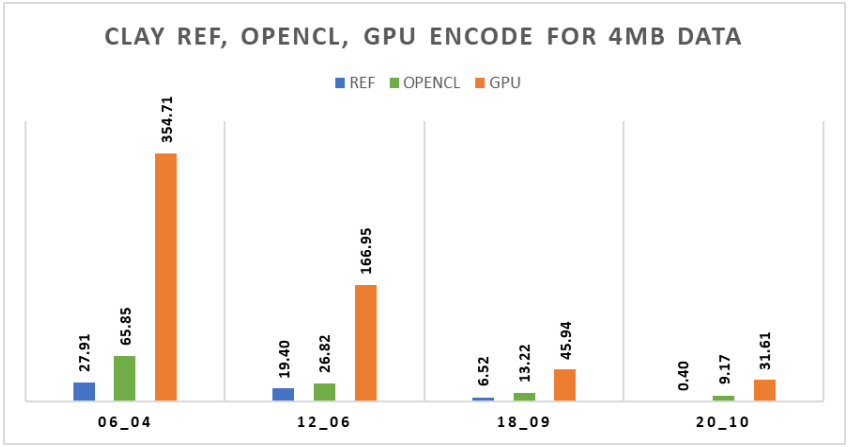


Approximate (3-5x) gain is observed in case of OpenCL and (10-18x) gain is observed in case of GPU. Gain increases with number of erasures.



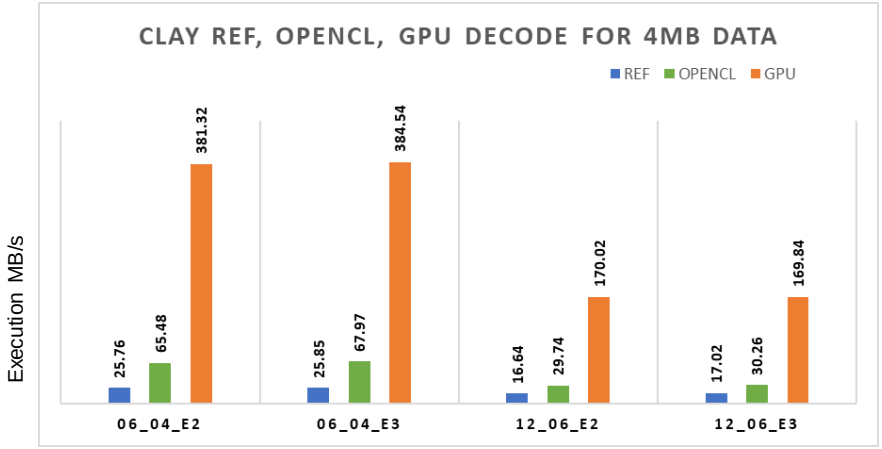
Encode bandwidth is approximately 4x and 16x for OpenCL and GPU respectively for (6, 4) and it gradually increases up to 45x with increase in (k, m) value. A slight decrease is seen with (k, m) value of (20, 10).

Results – CLAY performance summary



Encode bandwidth show 2-22x performance improvements for OpenCL and ~7-77x performance improvement for GPU for different (k, m) values.

The decode gain reduces with higher k, m values. It reduces from ~2.5x to 1.7x for OpenCL and from ~15x to ~10x for GPU.



Summary

- Accelerated Cauchy Reed Solomon (CRS) and Clay Code show good performance gain compared to corresponding reference versions on GPU and with OpenCL. The table below shows the maximum gain obtained in various cases.

Algo.	Encode		Decode	
	OpenCL	GPU	OpenCL	GPU
CRS	9.94	45.80	5.90	18.48
CLAY	22.84	78.78	2.63	14.88

- We continue the work of
  - Testing new and improved CRS and Clay code with a CEPH Cluster comprising four server machine with 16 core Intel Xeon CPU E5-2660 @ 2.20GHz, 64GB ram with NVIDIA GTX 1080 card and 60TB storage array

Erasure code : Future possibilities

- Erasure Coding Use Cases
  - Application Workload Dependent Resiliency
  - Storage Technology Dependent Resiliency
  - Integration of EC with File System
  - Data Migration for Resiliency Optimization

Reference

1. *Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. A Tale of Two Erasure Codes in HDFS. Usenix conference on File and storage technologies, 2015*
2. *Myna Vajha, Vinayak Ramkumar, Bhagyashree Puranik, Ganesh Kini, Elita Lobo, Birenjith Sasidharan, and P. Vijay Kumar, Indian Institute of Science, Bangalore; Alexandar Barg and Min Ye, University of Maryland; Srinivasan Narayanamurthy, Syed Hussain, and Siddhartha Nandi. Clay Codes: Moulding MDS Codes to Yield an MSR Code, Usenix conference on File and storage technologies, 2018.*
3. *Chengjian Liu, Qiang Wang, Xiaowen Chu, Yiu-Wing Leung. G-CRS: GPU Accelerated Cauchy Reed-Solomon Coding, IEEE Transactions on Parallel and Distributed Systems, 2018*
4. *Maheswaran Sathiamoorthy, Alexandros G. Dimakis, Megasthenis Asteris, Ramkumar Vadali, Dhruba Borthakur, Dimitris Papailiopoulos, Scott Chen. XORing Elephants: Novel Erasure Codes for Big Data, Proceedings of the VLDB Endowment, 2013.*

Thank you