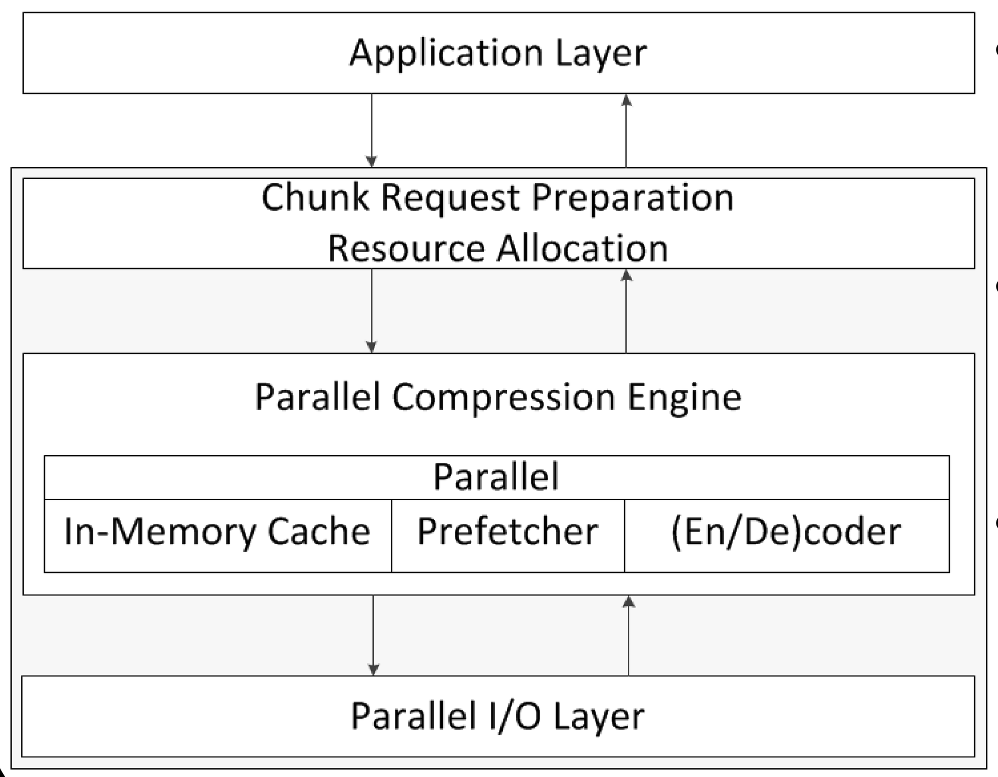
A Compression Framework for Multi-Dimensional Scientific Datasets

Tekin Bicer and Gagan Agrawal
Dept. of Computer Science & Eng.
Ohio State University

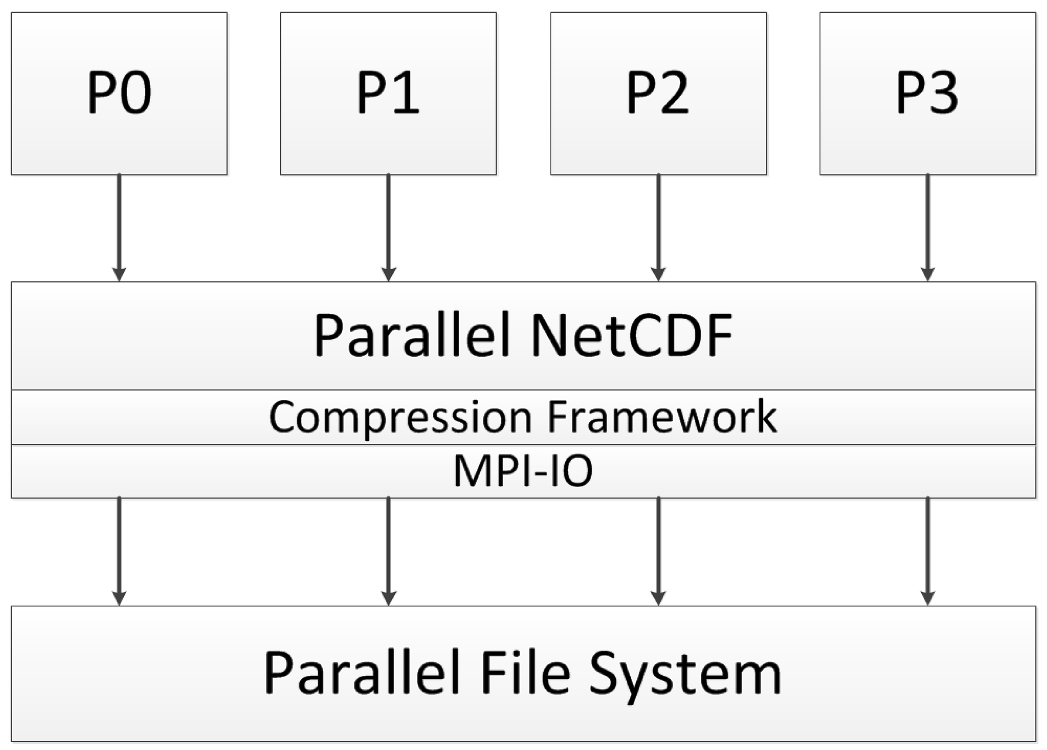
Introduction and Motivation

- Data collected from instruments and simulation are extremely valuable
- Data dissemination and analysis are complicated by the rapid growth of scientific data sizes
  - e.g., Global Cloud-Resolving Model (GCRM) produces 1PB of data for 4 km grid size over 10 day simulation
- Popular libraries for managing scientific datasets: NetCDF, PNetCDF, HDF5 etc.
- Compression can help storage and transfer

Proposed Compression Framework and Integration with PNetCDF



- Transparent access to compressed data from application layer
- Decoupled architecture between comp. engine and I/O layer
- Support for informed prefetching and in-memory cache



- We ported our comp. framework into PNetCDF library
- This library provides space efficient, array oriented data access with high performance I/O using ROMIO
- Widely used in scientific community

Challenges on Supporting Compression

- Compression can introduce additional computational complexity
  - Domain specific properties of scientific datasets can be exploited
  - Optimizations such as pipelining, parallel I/O and informed prefetching are desirable
- A framework which supports PnP of compression and decompression algorithms is needed
- Providing easy integration with data management and analysis software is challenging
  - Features of scientific dataset management libraries can be exploited

Compression Method for Scientific Data

- Most of the scientific datasets consist of single or double precision floating point numbers
- These datasets are array-oriented and adjacent cells are closely related with each other
- This relationship can be exploited with *prediction-based differential compression*
- *Example*: Consider a climate dataset, x, that consists of temperature of different locations

➢ First below equation is applied to x, and x' is generated

$$x'[i,j] = \begin{cases} x[i,j], & j = 0 \\ x[i,j] \oplus x[i,j-1], & j > 0 \end{cases}$$

➢ Second, the leading zeros are counted and represented in bits

➢ Third, the remaining part is appended and x'' is generated

x

299.55	299.55	299.54	299.54
299.75	299.79	299.78	299.76
300.01	300.05	300.09	300.08
302.00	302.05	302.05	302.05

x'

100010.....10	00.....0
100....10101010	00...0100101
100...101010	00...01101
100....111001010	0...01010101

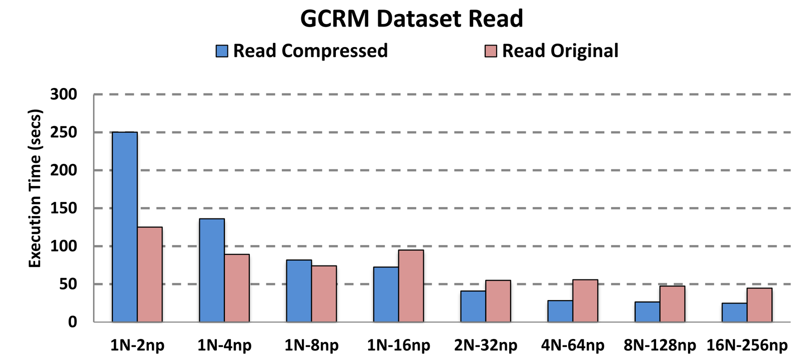
x''

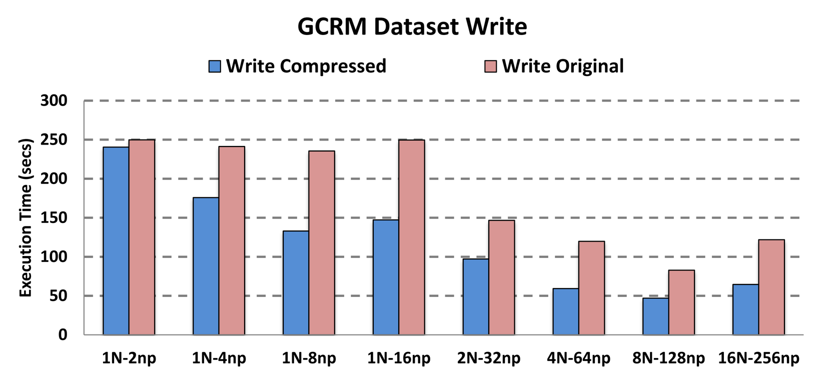
1000...11010	11111	111101	11111
1000...01010	11001100101	11100101	11100111
1000...01010	110111101	10110110101110	111101
1000...01010	110001010101	11111	11111

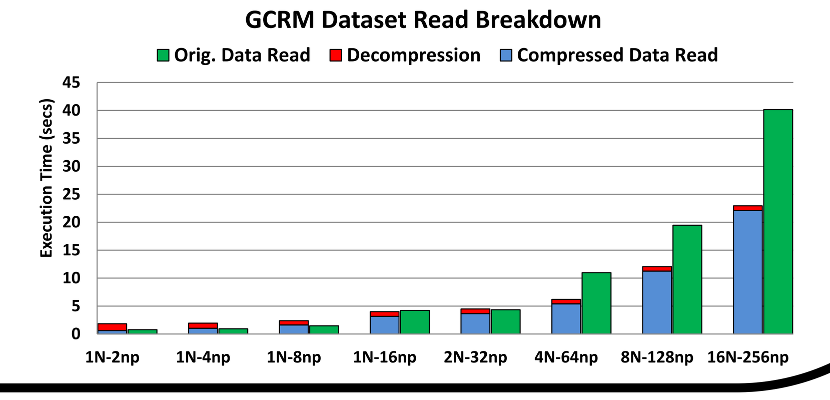
- x' is virtual and does not require storage
- Dropping least significant bits can further improve compression ratio (*lossy*)

Experimental Results

- Olympus cluster at PNNL
  - Lustre file system (8 OSTs, 1MB)
  - Each node has 16 Cores with 32 GB mem. (AMD Opteron 6272, 2.1GHz)
- GCRM Data: 68GB (35.4GB Comp.)
  - Simulation of 256 time steps, covers 28 km and 27 layers
  - Upto 16 nodes (256 cores)
- Speedups for read ops. are between 1.31 and 1.98 for =>8np config.
- Speedups for write ops. are between 1.52 and 2.07 for =>8np config.
- Decompression overhead decreases while the # of processes increases
  - 32.1–3.5 % for =>8np config.







Current Research Focus

- Determining the best compress algorithm
  - Sample the dataset, calculate the benefit values (comp. ratio vs. time)
  - Apply the best comp. alg., store this info. to record's metadata
- Find the optimum chunk size
  - Affects the comp. ratio as well as I/O throughput
  - Needs to exploit the parallel file system properties (e.g. stripe size, count)
- Detecting the application direction of the comp. alg.
  - Higher success ratios for predicted values result in better comp. ratio