Parallel I/O Libraries and Techniques

Mark Howison

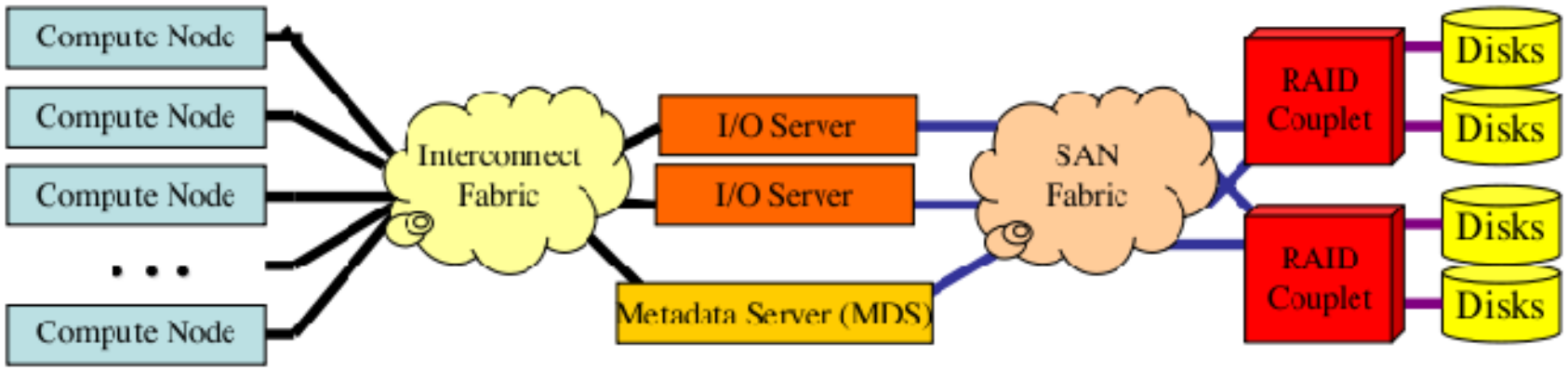
User Services & Support

I/O for scientific data

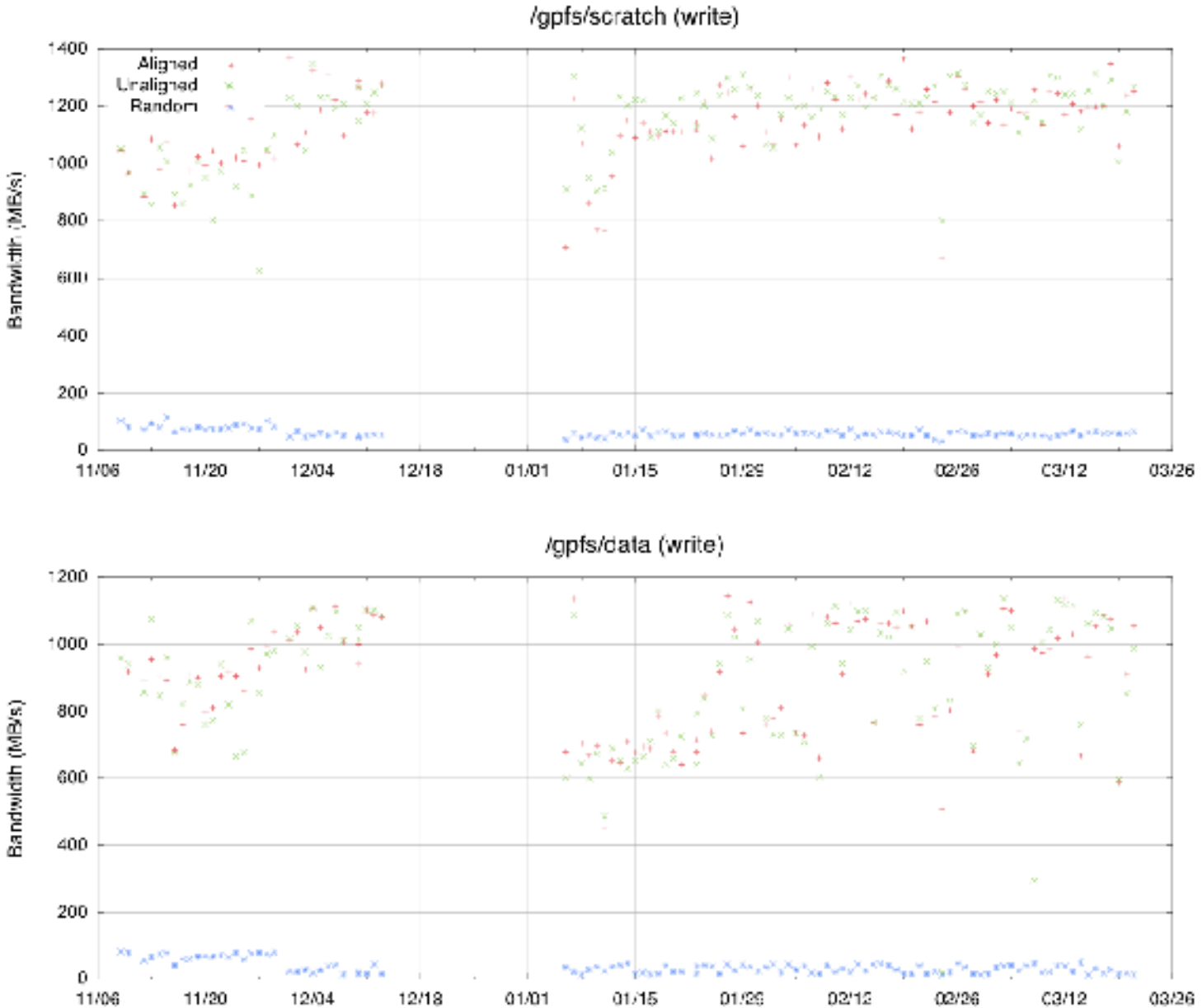
- I/O is commonly used by scientific applications to:
  - Store numerical output from simulations
  - Load initial conditions or datasets for processing
  - Implement 'out-of-core' techniques for algorithms that process more data than can fit in system memory
  - 'Checkpoint' to files that save the state of an application in case of system failure
- Usually access large amounts of data in a structured or sequential way
  - No random seeks throughout the file
- Writes are usually 'append-only'

Parallel file systems

- Most HPC systems are equipped with a parallel file system such as Lustre or GPFS that:
  - Abstract away spinning disks and RAID arrays by presenting 'I/O servers' to the user
  - Offer a single address space for accessing files that are really stored across many layers of devices
  - Can deliver aggregate bandwidth that is far above a single disk in a workstation or laptop
  - BUT often require careful tuning to achieve max performance



GPFS performance on Oscar

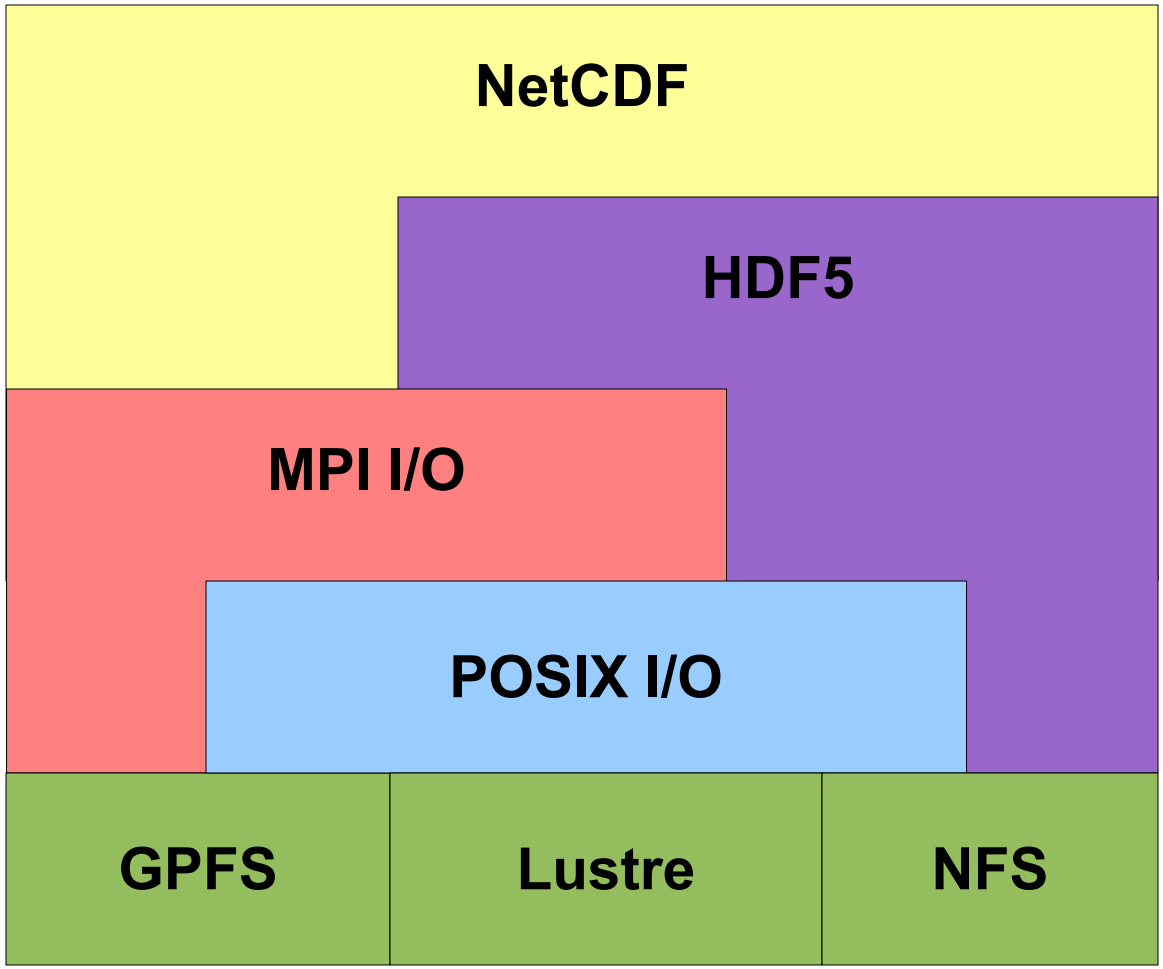


Parallel I/O software stack

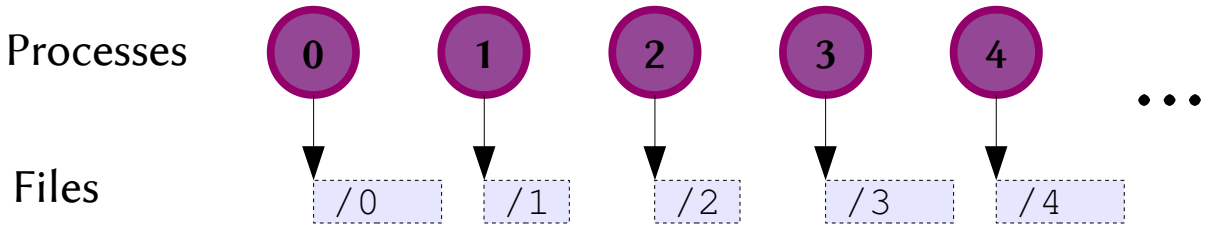
Application/
Library

Middle-ware

Kernel



File-per-process

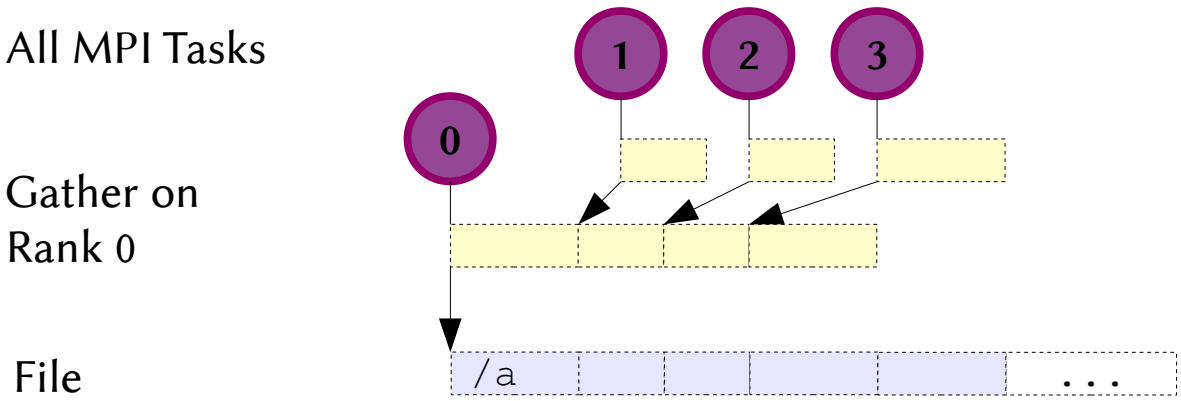


- Simplest to implement since each process access its own file independently without coordination among processes
- Parallel file systems often perform well with this type of access up to 1000s of files
  - Then synchronizing metadata becomes a bottleneck
- Large collections of files are unwieldy to manage
  - Even simple utilities like ls can break with 32K files
  - Creates data management headache for post-analysis, archiving, etc.

File-per-process (Cont'd)

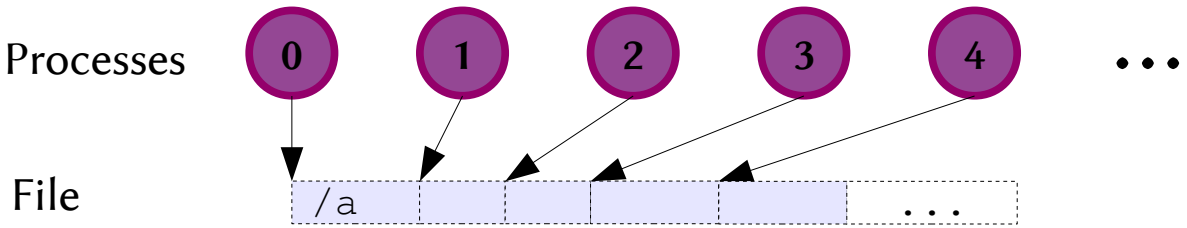
- Metadata bottleneck can be mitigated by using a 'square-root' file layout
  - For example, divide up 10,000 files into 100 subdirectories of 100 files.
- Unwieldy collections of files may not be an issue for checkpointing
  - Files most likely will not be read or accessed again
  - But, any restart will need to run with the same number of processes

Shared file, serialized access



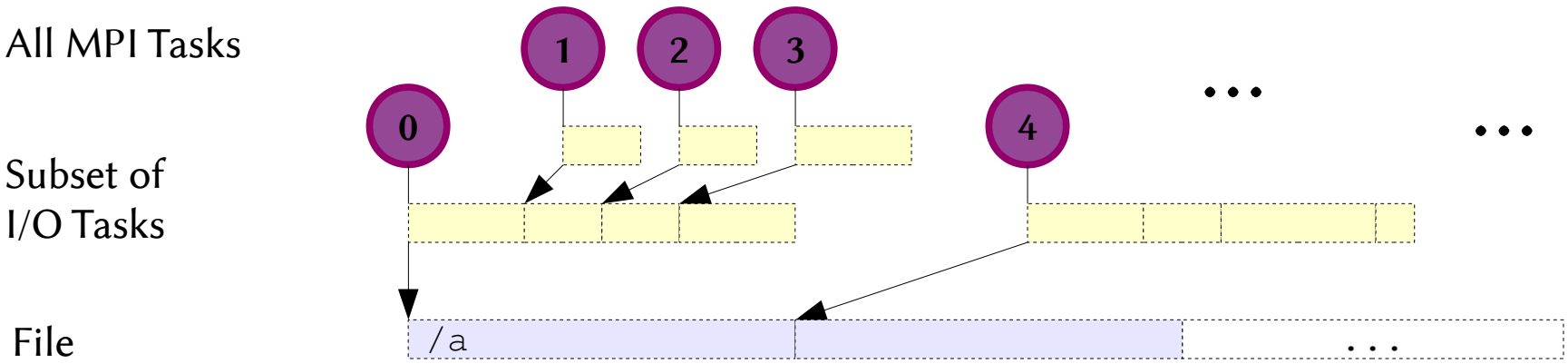
- All data is buffered through rank 0
  - Only rank 0 reads/writes: no coordination of file access
  - Rank 0 either gathers writes or scatters reads
- Not scalable!
  - May be ok for very small data, like for initialization parameters

Shared file, independent access



- All processes independently access exclusive regions of a single file
  - Coordination can take place at the parallel file system
  - Significant overhead for some access patterns where the file system has to serialize access
- Advantage of shared file lies in data management and portability

Shared file, collective access



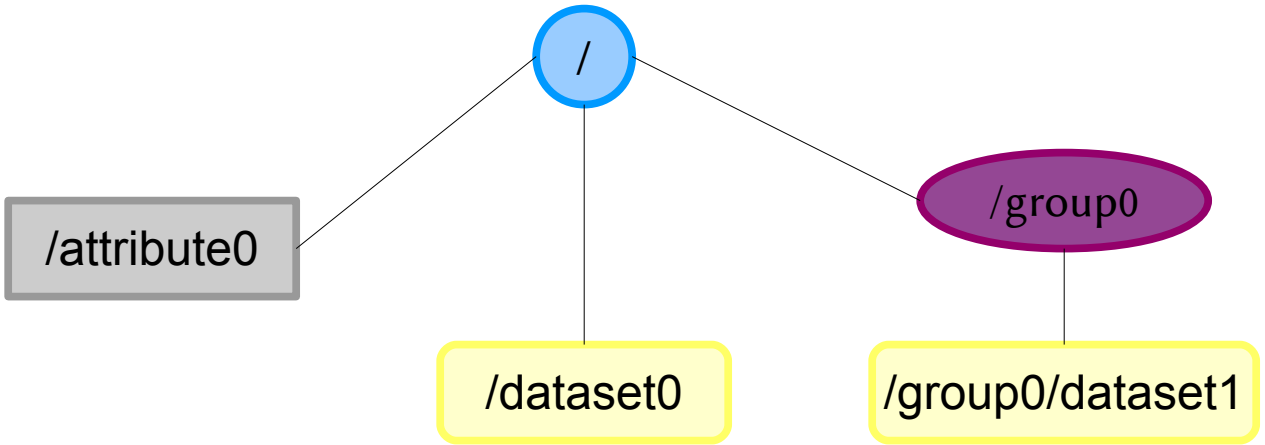
- Improves performance of shared-file access by offloading some of the coordination work from the file system to the application
  - A subset of processes is assigned to do I/O and buffers access to the file
  - Available in all modern MPI libraries (called “collective buffering”)

I/O libraries

- Hierarchical Data Format v5 (HDF5) is a portable I/O library used for storing scientific data
  - 'Object database' model
  - Don't have to worry about the specific layout of every byte in the file
  - Self-describing file format, handles platform differences (like endianess, native size of data types, etc.)
- NetCDF (Network Common Data Format)
  - Similar to HDF5
  - Used widely in the climate modeling community

Example HDF5 file

Logical view:



File view:

metadata	dataset0	dataset1

- Datasets are stored as flattened arrays within the file
- Attribute and group information is part of the HDF5 metadata
  - Different from the file system's metadata for the file

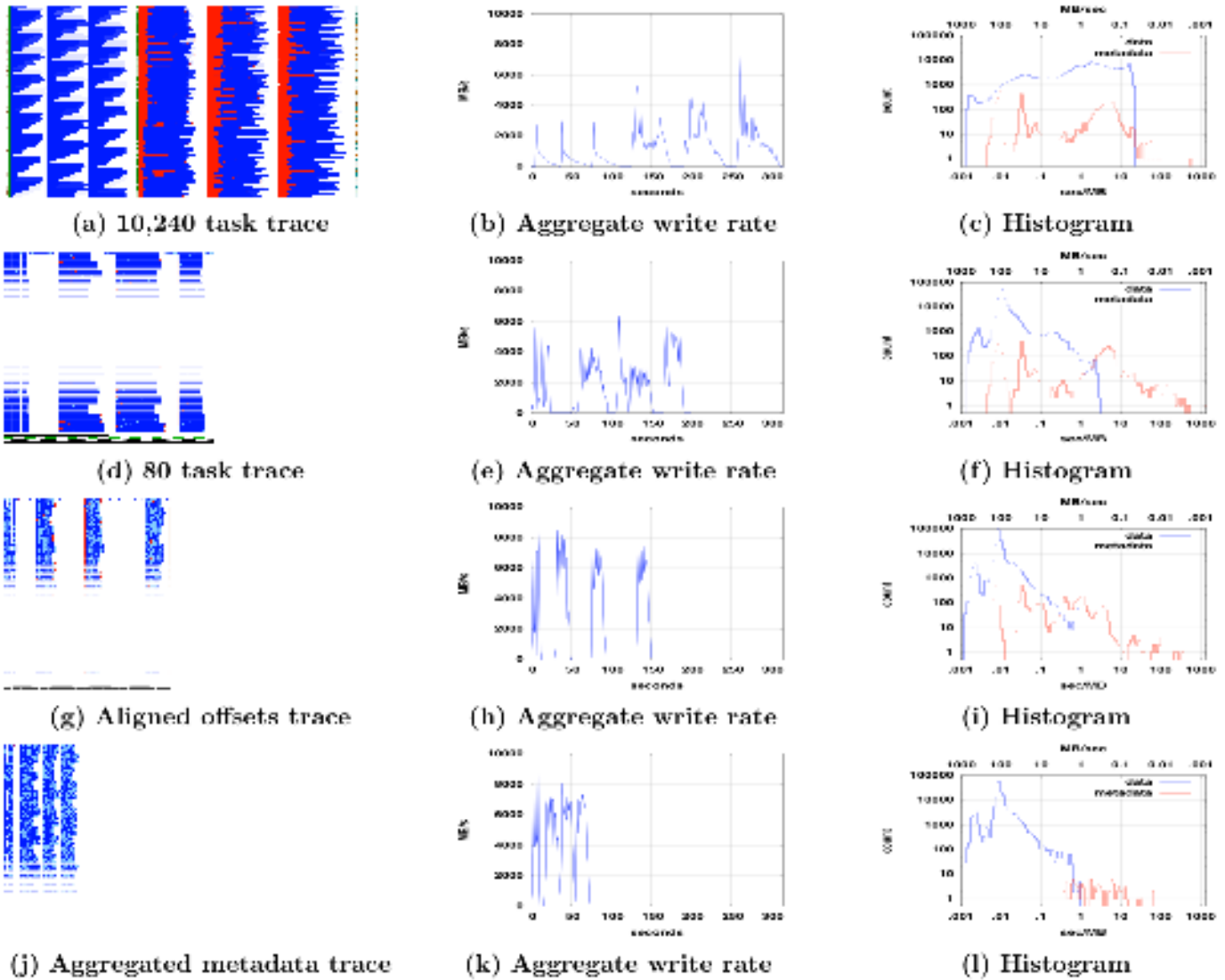
Common scientific I/O patterns

- Rectilinear 3D grids with balanced, cubic partitioning
- Rectilinear 3D grids with unbalanced, rectangular partitioning
- Contiguous but size-varying arrays, such as those found in adaptive mesh refinement
- For sample HDF5 code, see: http://www.nersc.gov/nusers/help/tutorials/io/5.php

Profiling and tracing I/O with IPM

- Integrated Performance Monitor (IPM) v.2 by default provides a summary of I/O activity
- Can also compile with an additional flag to enable full tracing
  - Generates detailed logs for each MPI task showing every POSIX I/O call (write, read, open, close, truncate, seek, etc.)
  - Can use this raw data to generate plots and histograms
  - To use on Oscar:
    `module load ipmio`
    Then relink program with $IPMIO in the link line

Example of I/O trace



(a) 10,240 task trace (b) Aggregate write rate (c) Histogram

(d) 80 task trace (e) Aggregate write rate (f) Histogram

(g) Aligned offsets trace (h) Aggregate write rate (i) Histogram

(j) Aggregated metadata trace (k) Aggregate write rate (l) Histogram

Figure 1: Trace graphs, aggregate write rates, and histograms for the GCRM I/O kernel with a baseline configuration and three progressive optimizations.

A. Uselton, M. Howison, N.J. Wright, D. Skinner, N. Keen, J. Shalf, K.L. Karavanic, & L. Oliker. Parallel I/O Performance: From Events to Ensembles. *IEEE International Parallel & Distributed Processing Symposium*, April, 2010.