Extreme I/O Scaling with HDF5

Quincey Koziol
Director of Core Software Development and HPC
The HDF Group
koziol@hdfgroup.org

Outline

- Brief overview of HDF5 and The HDF Group
- Extreme I/O Scaling with HDF5:
  - Application and Algorithm Efficiency
  - Application-based Fault Tolerant Methods and Algorithms
  - Application-based Topology Awareness
- What else is on the horizon for HDF5?

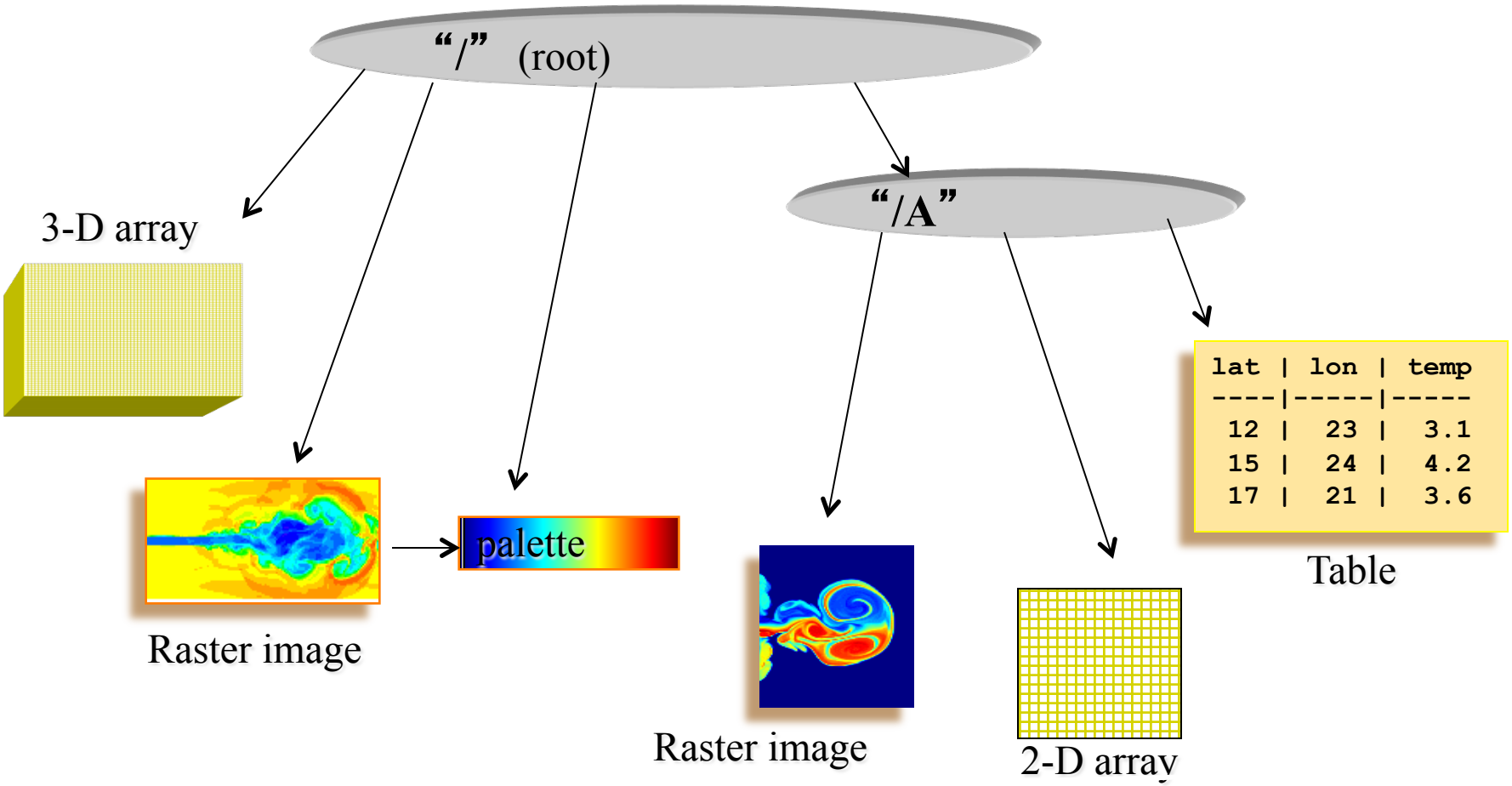
What is HDF5?

- **A versatile data model** that can represent very complex data objects and a wide variety of metadata.
- **A completely portable file format** with no limit on the number or size of data objects stored.
- **An open source software library** that runs on a wide range of computational platforms, from cell phones to massively parallel systems, and implements a high-level API with C, C++, Fortran 90, and Java interfaces.
- **A rich set of integrated performance features** that allow for access time and storage space optimizations.
- **Tools and applications** for managing, manipulating, viewing, and analyzing the data in the collection.

The HDF5 Data Model

- Datasets
  - Multi-dimensional array of elements, together with supporting metadata
- Groups
  - Directory-like structure containing links to datasets, groups, other objects

Example HDF5 File



Why HDF5?

- Challenging data:
  - Application data that pushes the limits of what can be addressed by traditional database systems, XML documents, or in-house data formats.
- Software solutions:
  - For very large datasets, very fast access requirements, or very complex datasets.
  - To easily share data across a wide variety of computational platforms using applications written in different programming languages.
  - That take advantage of the many open-source and commercial tools that understand HDF5.

Who uses HDF5?

- Some examples of HDF5 users
  - Astrophysics
  - Astronomers
  - NASA Earth Science Enterprise
  - Dept. of Energy Labs
  - Supercomputing centers in US, Europe and Asia
  - Financial Institutions
  - NOAA
  - Manufacturing industries
  - Many others
- For a more detailed list, visit
  - http://www.hdfgroup.org/HDF5/users5.html

What is The HDF Group And why does it exist?

The HDF Group

- 18 Years at University of Illinois National Center for Supercomputing Applications (NCSA)
- Spun-out from NCSA in July, 2006
- Non-profit
- ~40 scientific, technology, professional staff
- Intellectual property:
  - The HDF Group owns HDF4 and HDF5
  - HDF formats and libraries to remain open
  - BSD-style license

The HDF Group Mission

To ensure long-term accessibility of HDF data through sustainable development and support of HDF technologies.

Goals

- Maintain, evolve HDF for sponsors and communities that depend on it
- Provide consulting, training, tuning, development, research
- Sustain the group for long term to assure data access over time

Extreme I/O with HDF5

- Application and Algorithm Efficiency
  - Scaling use of HDF5 from current petascale to future exascale systems
- Application-based Fault Tolerant Methods and Algorithms
  - File and system-based fault tolerance with HDF5
- Application-based Topology Awareness
  - I/O Autotuning research

Application & Algorithm Efficiency

- Success Story @ NERSC:
  - *Sifting Through a Trillion Electrons*
    - http://crd.lbl.gov/news-and-publications/news/2012/sifting-through-a-trillion-electrons/
  - 3-D magnetic reconnection dataset of a trillion particles, using VPIC application
  - Ran on NERSC's Cray XE6 *hopper* system, using 120,000 cores, writing each snapshot into a single 32TB HDF5 file, at a sustained I/O rate of 27GB/s, with peaks of 31GB/s
    - **~90% of maximum system I/O performance!**

Application & Algorithm Efficiency

- Aspects to HDF5 Efficiency
  - General Issues
  - Parallel Issues
  - Upcoming parallel Improvements:
    - File Truncation w/MPI
    - Collective HDF5 Metadata Modifications
    - Exascale FastForward work

General HDF5 Efficiency

- Faster HDF5 Performance: **Metadata**
  - Use the “latest” file format features
    - *H5Pset_libver_bounds()*
  - Increase size of metadata data structures
    - *H5Pset_istore_k(), H5Pset_sym_k(), etc.*
  - Aggregate metadata into larger blocks
    - *H5Pset_meta_block_size()*
  - Align objects in the file
    - *H5Pset_alignment()*
  - “Cork the cache”
    - Prevents “trickle” of metadata writes
    - [*Code example in slide notes*]

General HDF5 Efficiency

- Faster HDF5 Performance: **Raw Data**
  - Avoid converting datatypes
    - If conversion necessary, increase datatype conversion buffer size (default 1MB) with *H5Pset_buffer()*
  - Possibly use chunked datasets
    - *H5Pset_chunk()*
  - Make your chunks the “right” size
    - Goldilocks Principle: Not too big, nor too small
  - Increase size of the chunk cache: default is 1MB
    - *H5Pset_chunk_cache()*

Upcoming General Improvements

- Sampling of New Features:
  - Single-Writer/Multiple Reader (SWMR)
    - Allows concurrent, lock-free access from a single writing process and multiple reading processes
  - Asynchronous I/O
    - Adding support for POSIX aio_*() routines
  - Scalable chunk indices
    - Constant time lookup and append operations for nearly all datasets
  - Persistent free space tracking
    - Options for tracking free space in file, when it's closed
- All available in 1.10.0 release

Parallel HDF5 Efficiency

- Create skeleton file in serial
  - Current parallel HDF5 requires collective modification of metadata
  - If structure of file is [mostly] known, create skeleton in serial, then close and re-open file in parallel
- Use collective I/O for raw data
  - *H5Pset_dxpl_mpio()*
  - Possibly use collective chunk optimization routines
    - *H5Pset_dxpl_mpio_chunk_opt(), H5Pset_dxpl_mpio_chunk_opt_num(), H5Pset_dxpl_mpio_chunk_opt_ratio(), H5Pset_dxpl_mpio_collective_opt()*
  - Query whether collective I/O occurs with:
    - *H5Pget_mpio_actual_io_mode()* and *H5Pget_mpio_actual_chunk_opt_mode()*

Upcoming Parallel Improvements

- File Truncation
  - Problem:
    - Current HDF5 file format requires file always truncated to allocated size when closed
    - File truncation with MPI is very slow
  - Solution:
    - Modify HDF5 library and file format to store allocated size as metadata within the HDF5 file
    - Gives large speed benefit for parallel applications
    - Available in 1.10.0 release

Upcoming Parallel Improvements

- Collective HDF5 Metadata Modification
  - Problem:
    - Current HDF5 library requires all metadata modifications to be performed collectively
  - Solution:
    - Set aside one (or more, eventually) process as metadata server and have other processes communicate metadata changes to server
    - Available in 1.10.0 release (possibly)

Upcoming Parallel Improvements

- Exascale FastForward work w/Whamcloud &EMC
  - Whamcloud, EMC & The HDF Group were recently awarded contract for exascale storage research and prototyping:
    - http://www.hpcwire.com/hpcwire/2012-07-12/doe_primes_pump_for_exascale_supercomputers.html
  - Using HDF5 data model and interface as top layer of next generation storage system for future exascale systems
  - Laundry list of new features in HDF5
    - Parallel compression, asynchronous I/O, query/index features, transactions, python wrappers, etc.

Extreme I/O with HDF5

- Application and Algorithm Efficiency
  - Scaling use of HDF5 from current petascale to future exascale systems
- Application-based Fault Tolerant Methods and Algorithms
  - File and system-based fault tolerance with HDF5
- Application-based Topology Awareness
  - I/O Autotuning research

Fault Tolerance and HDF5

- Metadata Journaling
  - Protect file by writing metadata changes to journal file first, then writing to HDF5 file.
  - Performance trade-off for good fault tolerance
  - Works with non-POSIX file systems
- Ordered Updates
  - Protect file by special data structure update algorithms - essentially copy-on-write and lock-free/ wait-free algorithms
  - Good performance and good fault tolerance
  - Only works on POSIX file systems

Fault Tolerance and HDF5

- MPI Fault Tolerance
  - Participating in MPI Forum since 2008, contributing and helping to advance features that will enable HDF5 to meet future needs
  - Forum's Fault Tolerance Working Group has proposed features that will allow MPI file handles to be "rebuilt" if participating processes fail
  - If this feature is voted into the MPI standard and available in implementations, we will adopt it, so that applications can be certain that their HDF5 files are secure

Extreme I/O with HDF5

- Application and Algorithm Efficiency
  - Scaling use of HDF5 from current petascale to future exascale systems
- Application-based Fault Tolerant Methods and Algorithms
  - File and system-based fault tolerance with HDF5
- Application-based Topology Awareness
  - I/O Autotuning research

Autotuning Background

- Software Autotuning:
  - Employ empirical techniques to evaluate a set of alternative mappings of computation kernels to an architecture and select the mapping that obtains the best performance.
- Autotuning Categories:
  - Self-tuning library generators such as ATLAS, PhiPAC and OSKI for linear algebra, etc.
  - Compiler-based autotuners that automatically generate and search a set of alternative implementations of a computation
  - Application-level autotuners that automate empirical search across a set of parameter values proposed by the application programmer

HDF5 Autotuning

- Why?
  - Because the dominant I/O support request at NERSC is poor I/O performance, many/most of which can be solved by enabling Lustre striping, or tuning another I/O parameter
  - *Scientists shouldn't have to figure this stuff out!*
- Two Areas of Focus:
  - Evaluate techniques for autotuning HPC application I/O
    - File system, MPI, HDF5
  - Record and Replay HDF5 I/O operations

Autotuning HPC I/O

- Goal: Avoid tuning each application to each machine and file system
  - Create I/O autotuner library that can inject "optimal" parameters for I/O operations on a given system
- Using Darshan* tool to create wrappers for HDF5 calls
  - Application can be dynamically linked with I/O autotuning library
  - No changes to application or HDF5 library
- Using several HPC applications currently:
  - VPIC, GCRM, Vorpal

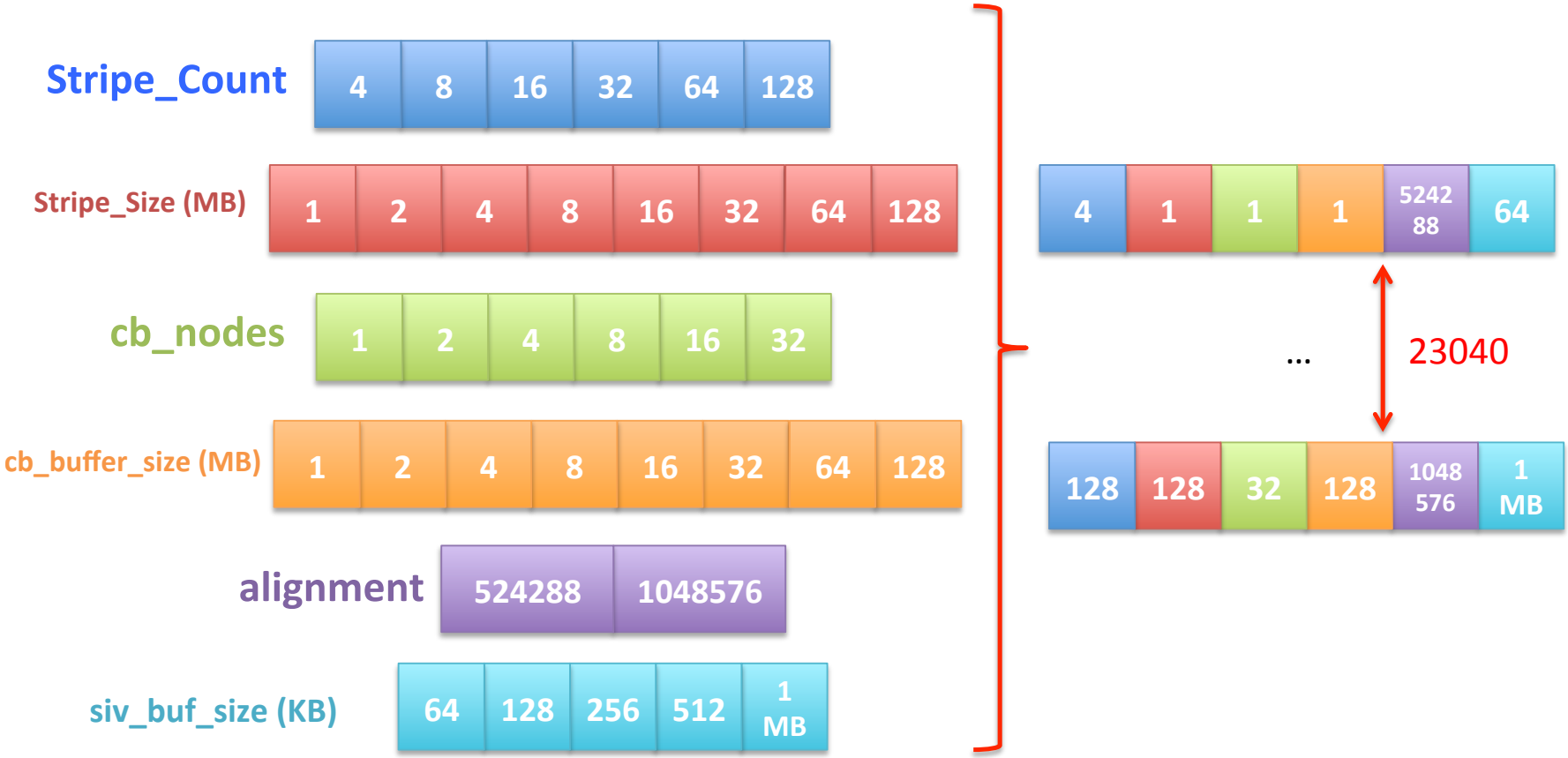
* - http://www.mcs.anl.gov/research/projects/darshan/

Autotuning HPC I/O

- Initial parameters of interest
  - File System (Lustre): stripe count, stripe unit
  - MPI-I/O: Collective buffer size, coll. buffer nodes
  - HDF5: Alignment, sieve buffer size

Autotuning HPC I/O

The whole space visualized



Autotuning HPC I/O

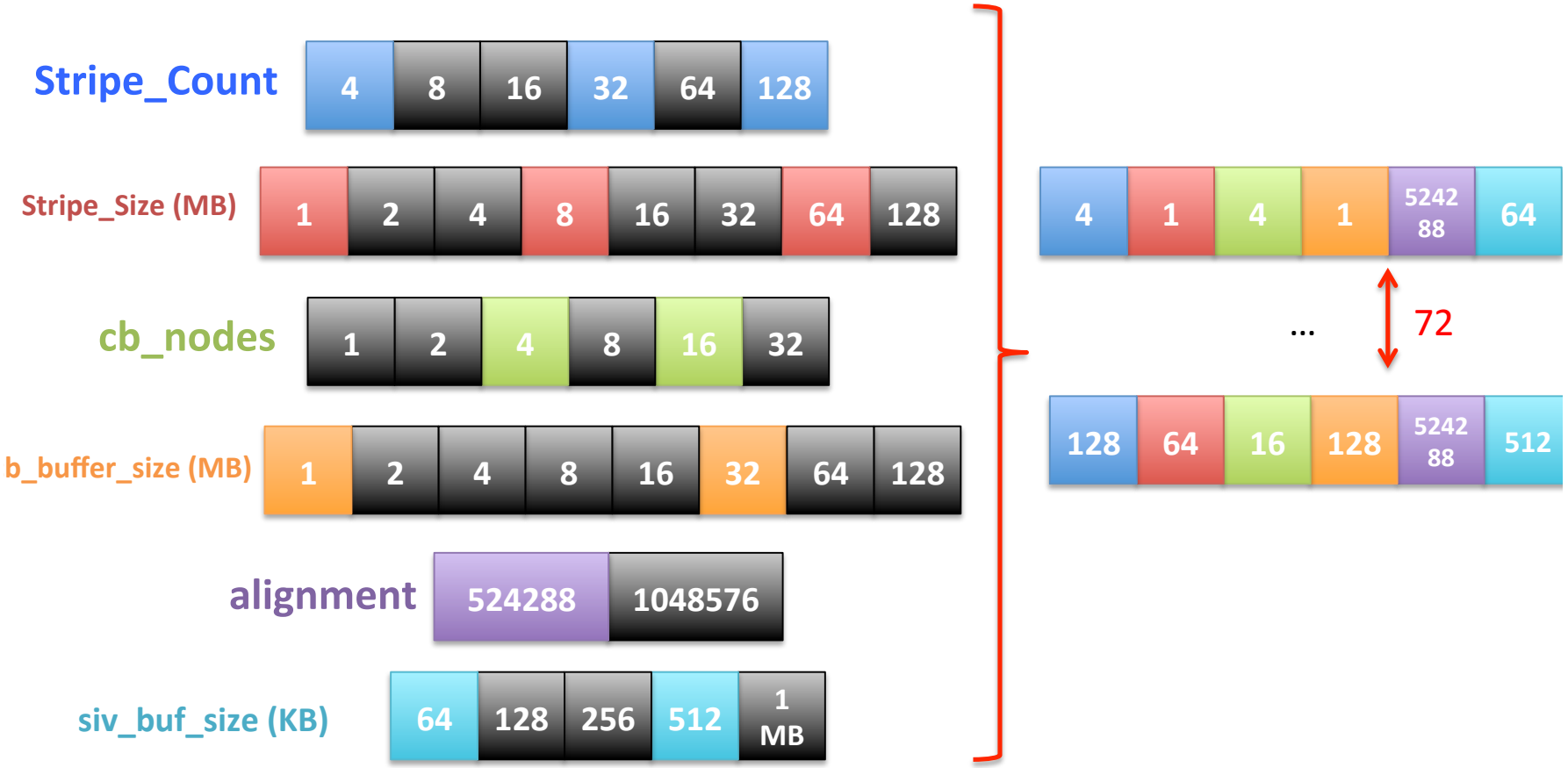
- Autotuning Exploration/Generation Process:
  - Iterate over running application many times:
    - Intercept application's I/O calls
    - Inject autotuning parameters
    - Measure resulting performance
- Analyze performance information from many application runs to create configuration file, with best parameters found for application/machine/file system

Autotuning HPC I/O

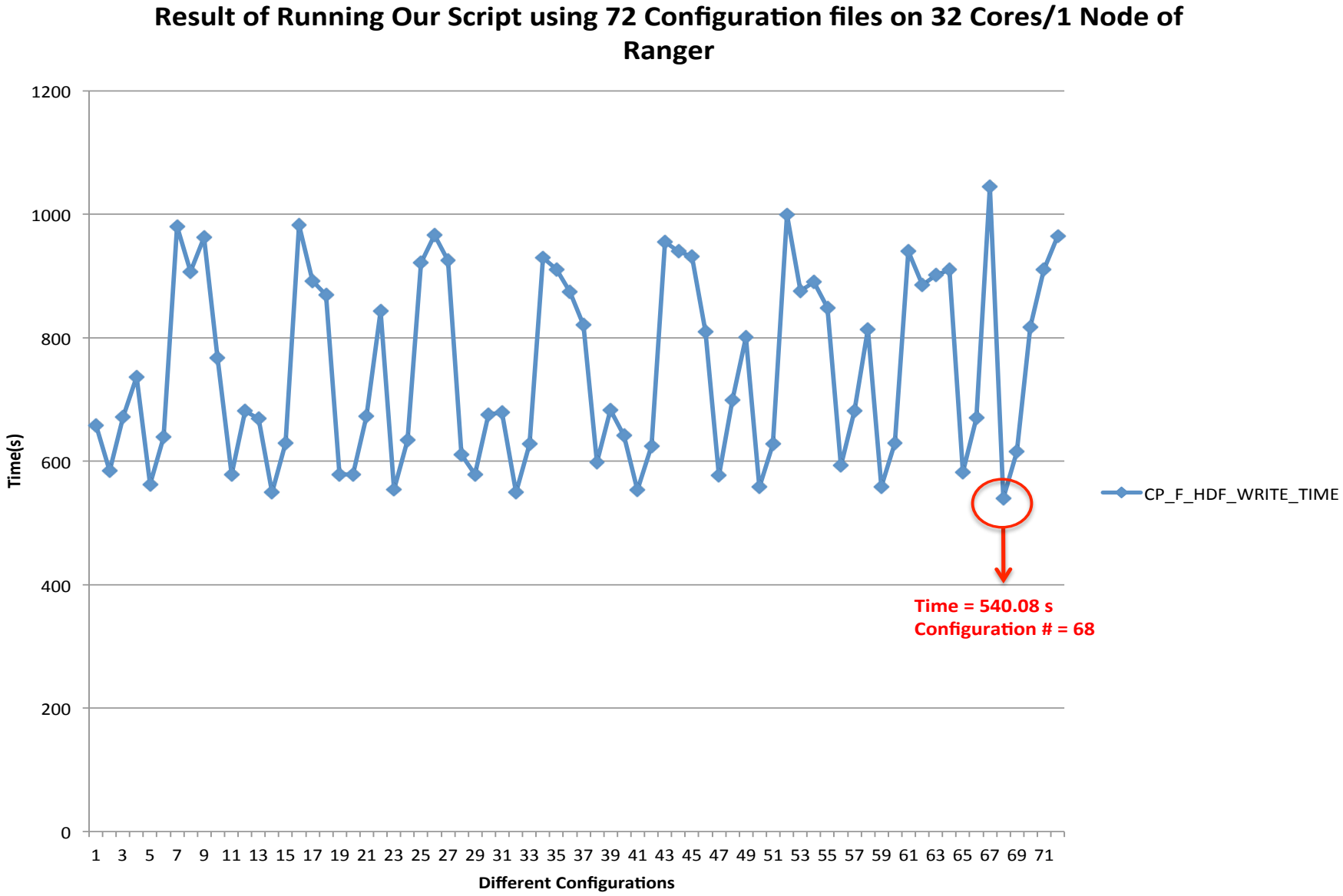
- Using the I/O Autotuning Library:
  - Dynamically link with I/O autotuner library
  - I/O autotuner library automatically reads parameters from config file created during exploration process
  - I/O autotuner automatically injects autotuning parameters as application operates

Autotuning HPC I/O

Smaller set of space visualized



Autotuning HPC I/O



Configuration #68

```
<Parameters>
        <High_Level_IO_Library>
                <sieve_buf_size> 524280 </sieve_buf_size>
                <alignment> 262144,524288 </alignment>
                <!-- H5Pset_alignment function gets 2 args: (Threshold, Alignment) -
        </High_Level_IO_Library>

        <Middleware_Layer>
                <cb_buffer_size> 134217728 </cb_buffer_size>
                <cb_nodes> 16 </cb_nodes>
        </Middleware_Layer>

        <Parallel_File_System>
                <striping_factor> 32 </striping_factor>
                <striping_unit> 8388608 </striping_unit>
        </Parallel_File_System>
</Parameters>
```

Autotuning HPC I/O

- Remaining research:
  - Determine “speed of light” for I/O on system and use that to define “good enough” performance
  - Entire space is too large to fully explore, we are now evaluating genetic algorithm techniques to help find “good enough” parameters
  - How to factor out “unlucky” exploration runs
  - Methods for avoiding overriding application parameters with autotuned parameters

Recording and Replaying HDF5

- Goal:
  - Extract an “I/O Kernel” from application, without examining application code
- Method:
  - Using Darshan again, record all HDF5 calls and parameters in “replay file”
  - Create parallel replay tool that uses recording to replay HDF5 operations

Recording and Replaying HDF5

- Benefits:
  - Easy to create I/O kernel from any application, even one with no source code
  - Can use replay file as accurate application I/O benchmark
  - Can move replay file to another system for comparison
  - Can autotune from replay, instead of application
- Challenges:
  - Serializing complex parameters to HDF5
  - Replay files can be very large
  - Accurate parallel replay is difficult

HDF5 Road Map

HDF5 in the Future

"It's tough to make predictions, especially about the future"

– Yogi Berra

Plans, Guesses & Speculations

- Focus on asynchronous I/O, instead of improving multi-threaded concurrency of HDF5 library:
  - Library currently thread-safe, but not concurrent
  - Instead of improving concurrency, focus on heavily leveraging asynchronous I/O
  - Use internal multi-threading where possible
- Extend Single-Writer/Multiple-Reader (SWMR) to parallel applications
  - Allow an MPI application to be the writer

Plans, Guesses & Speculations

- Improve Parallel I/O Performance:
  - Continue to improve our use of MPI and file system features
  - Reduce # of I/O accesses for metadata access
- Improve Journaled HDF5 File Access:
  - Make raw data operations journaled
  - Allow "super-transactions" to be created by applications
  - Enable journaling for Parallel HDF5

Plans, Guesses & Speculations

- Improve raw data chunk cache implementation
- More efficient storage and I/O of variable-length data, including compression
- Work with HPC community to serve their needs:
  - Focus on high-profile applications or "I/O kernels" and remove HDF5 bottlenecks discovered

You tell us!

Questions?