SUPPORTING SQL QUERIES FOR SUBSETTING LARGE-SCALE DATASETS IN PARAVIEW

Yu Su*, Gagan Agrawal*, Jon Woodring†

*The Ohio State University

†Los Alamos National Laboratory

Los Alamos NATIONAL LABORATORY EST. 1943

SC’11 UltraVis Workshop, November 13, 2011

Subsetting Large-Scale Data

- Data subsetting is needed for efficient scientific large-scale visualization and analysis
  - Post-processing is still needed for some visualization and analysis scenarios (global time analysis, exploratory visualization, etc.)
  - Slow I/O and network bandwidth
  - Memory footprint decreasing per core at exascale
  - New data generated during analysis process increases the memory footprint

Data Subsetting in ParaView

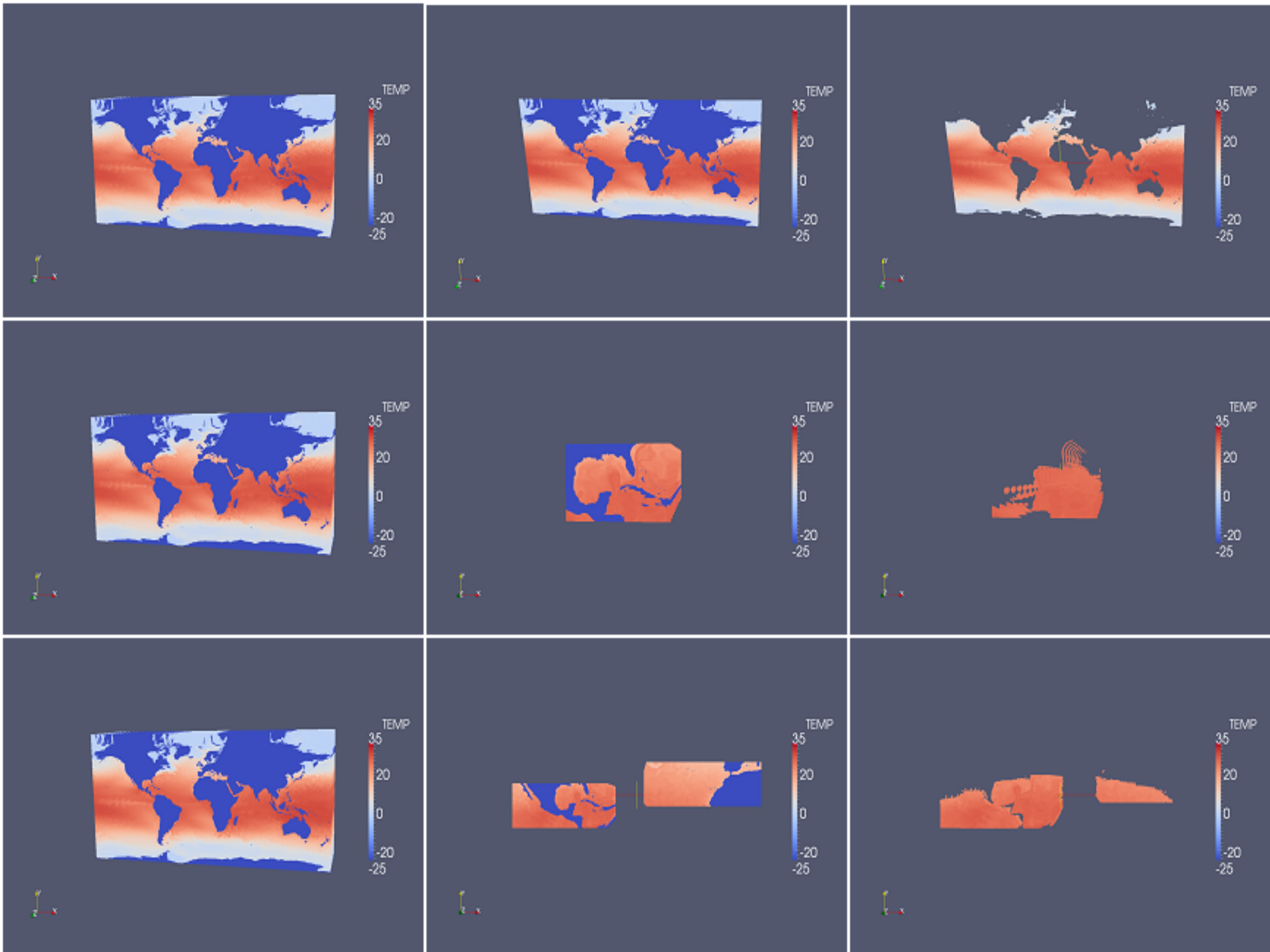
- Load an entire data set (or spatial subset)
  - Apply a series filters and/or the data selection interface
  - Multiple filters keep multiple data copies in memory
  - New grids may be created increasing memory and time
- Rectilinear Grid readers: Slow (can be fast for spatial)
  - Extract Subset/Slice/Clip Filter: Fast
  - Threshold Filter: Very Slow (new unstructured grid)
- Unstructured Grid readers: Slow
  - Extract Subset/Slice/Clip Filter: Medium
  - Threshold Filter: Slow (new unstructured grid)

A Faster Solution

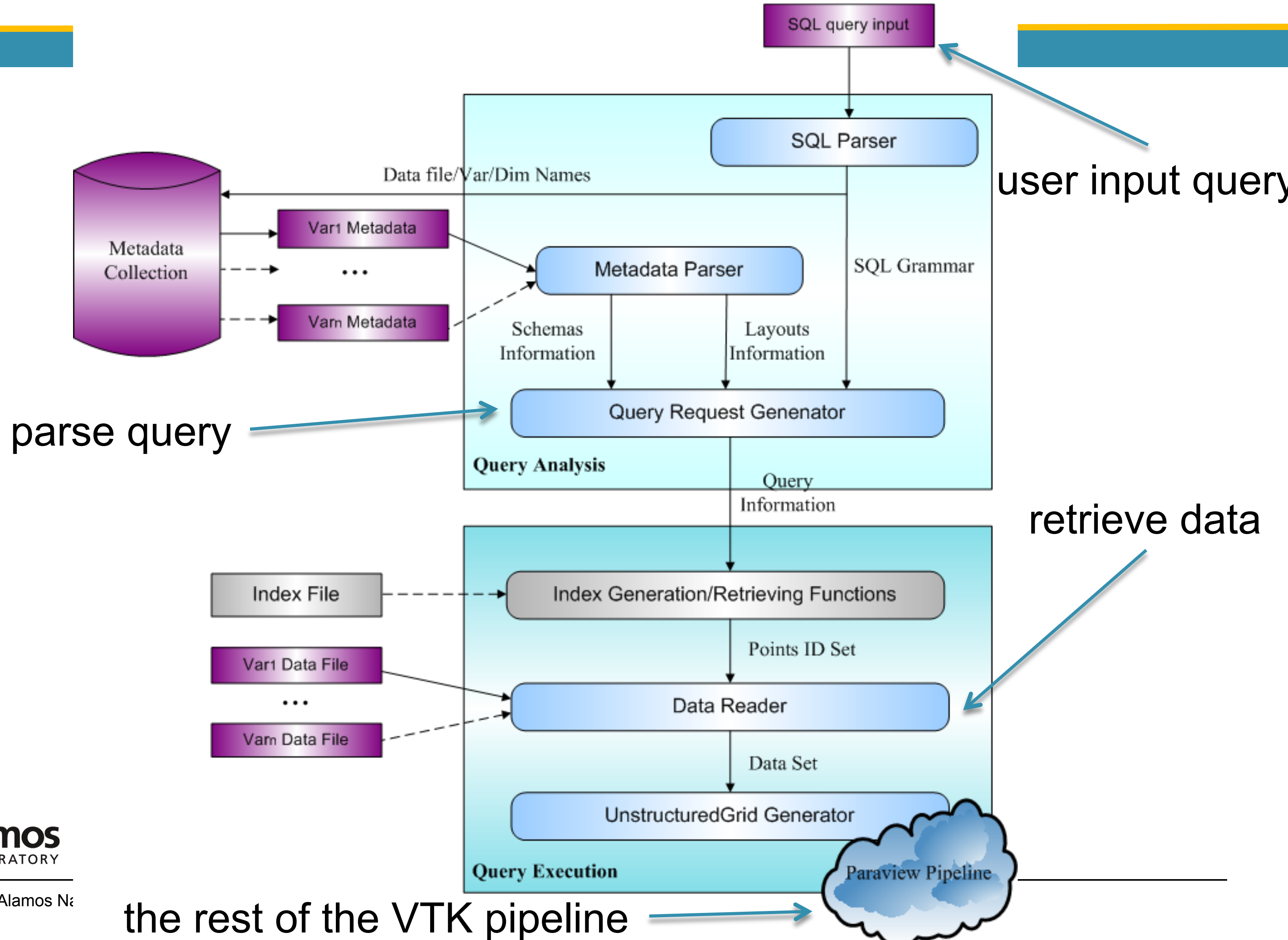
- Subset at the I/O level
  - Reduced I/O times and memory footprint
  - User specifies the subset in one query for both space and value ranges
- SQL queries in ParaView reader modules
  - Standard: A flexible language for specifying subsets
  - Efficiency: Smaller data load from disk to memory
  - Functionality: One query is equal to multiple filters

Broader Research Context

- Automatic Data Virtualization Research at The Ohio State University
- Virtual Relational/XML view on low-level scientific data
  - A light-weight database management solution
  - Support for flat-files and HDF5 in past work
  - No need to load data in a specific database – the data are able to stay as-is


TEMP
35
20
0
-20
-25

ParaView Reader and SQL Queries



Retrieval/Indexing Functionality

- Spatial queries are relatively easy
  - NetCDF and HDF5 support spatial queries in their API
  - Unstructured data is harder (but feasible)
- Value queries harder: Bitmap Indexing!
  - One truth bit for each data element and value range (1 if datum is in value range, 0 if not) N x B bit matrix
    - Value bin cardinality B (range quantization) is a tradeoff for indexing time and space – and there are bitmap compression techniques (WAH, multi-hashing, etc.)
  - Fast bitwise operations on bitmap index determine the data point selection sets from queries

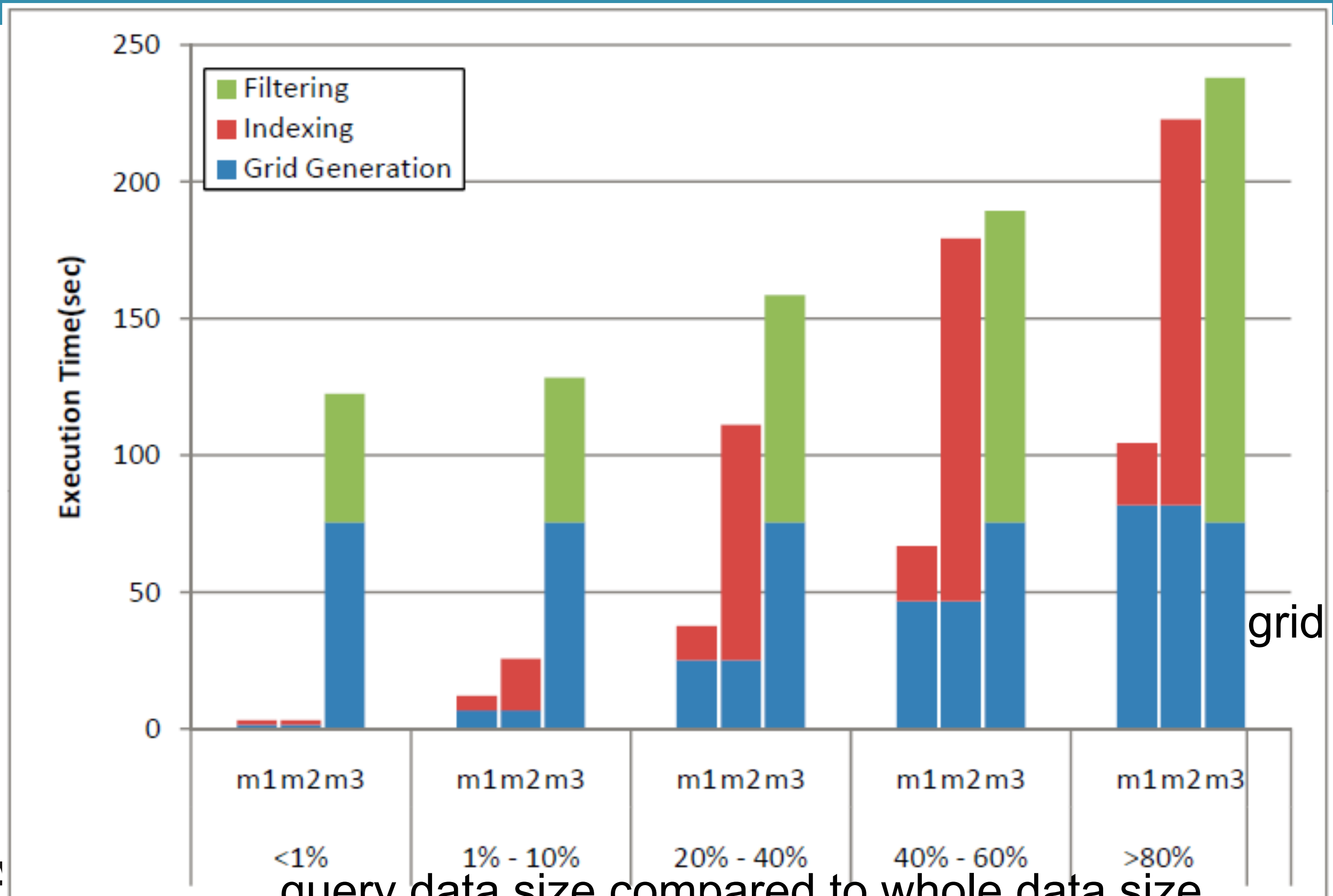
Experimental Test Setup

- POP (Parallel Ocean Program) NetCDF data files
  - 3600x2400x42 structured grid
  - 1.4 GB per variable – 4 variables (5.6 GB)
- SQL + NetCDF API + Bitmap indexing vs. reader modules + multiple VTK filters (single threaded)
  - Type 1: Spatial queries (skipped – it's as fast as the NetCDF API can service a spatial query)
  - Type 2: Value queries (100 random queries)
  - Type 3: Space + Value queries (100 random queries)

Experiment: Value Only Queries

- Spatial query ignored in this test
  - Two-level bitmap indexing
  - One-level bitmap indexing
  - Read whole data + multiple VTK threshold filters
- 2-level indexing
  - Coarse grain index for values for a first pass (fewer bins), followed by a finer grain indexing per bin in a second pass
  - Throws out a large number of candidates on first pass – more efficient than 1-level indexing in many cases

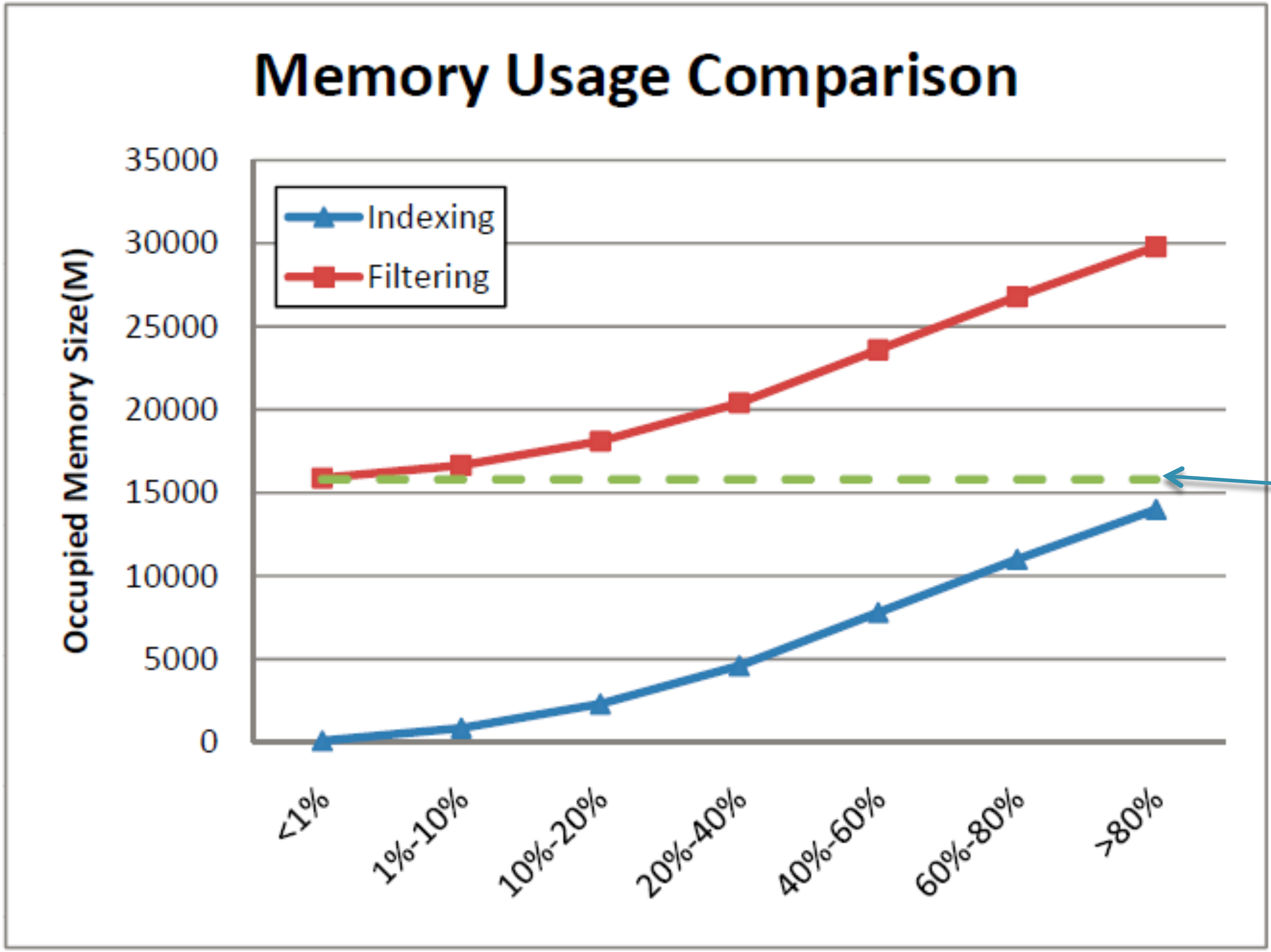
m1 – 2-level indexing and read only query data
m2 – 1-level indexing and read only query data
m3 – read all data and use multiple thresholds



grid creation

query data size compared to whole data size

Memory Usage on Value Queries

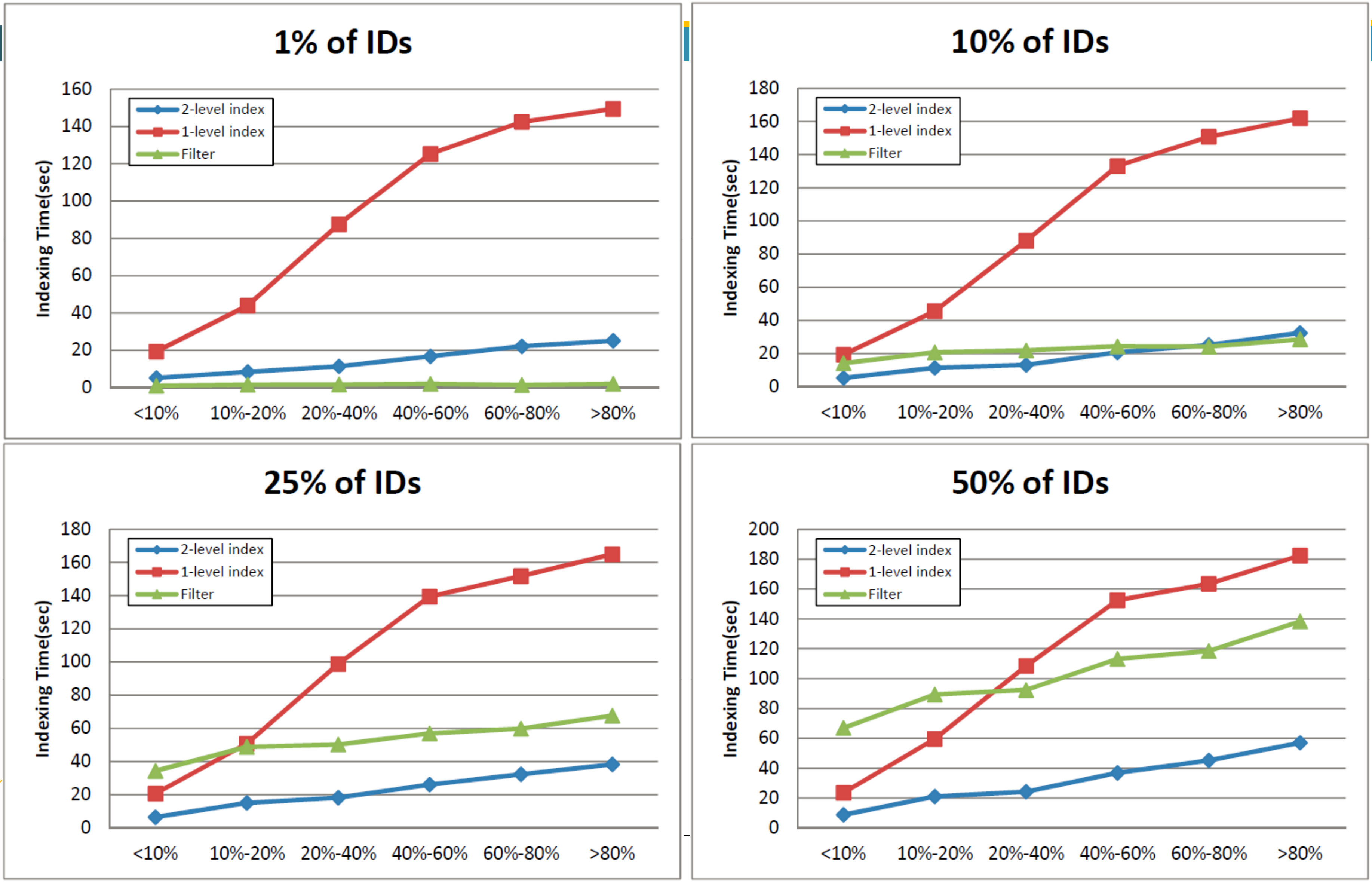


Memory occupied by reading the entire data set

Experiment: Space + Value Queries

- Subsetting on IDs (space) combined with values
  - 2-level indexing + NetCDF API
  - 1-level indexing + NetCDF API
  - VTK NetCDF reader + subset filter + threshold filter
    - Reader is smart that it only reads requested spatial subsets
- Dominant factor for indexing is still in values
  - SELECT temp FROM DATASET WHERE t_lat=0.5 AND t_lon = 0.5 AND temp<100;
  - SELECT temp FROM DATASET WHERE temp=0 AND t_lat>0;

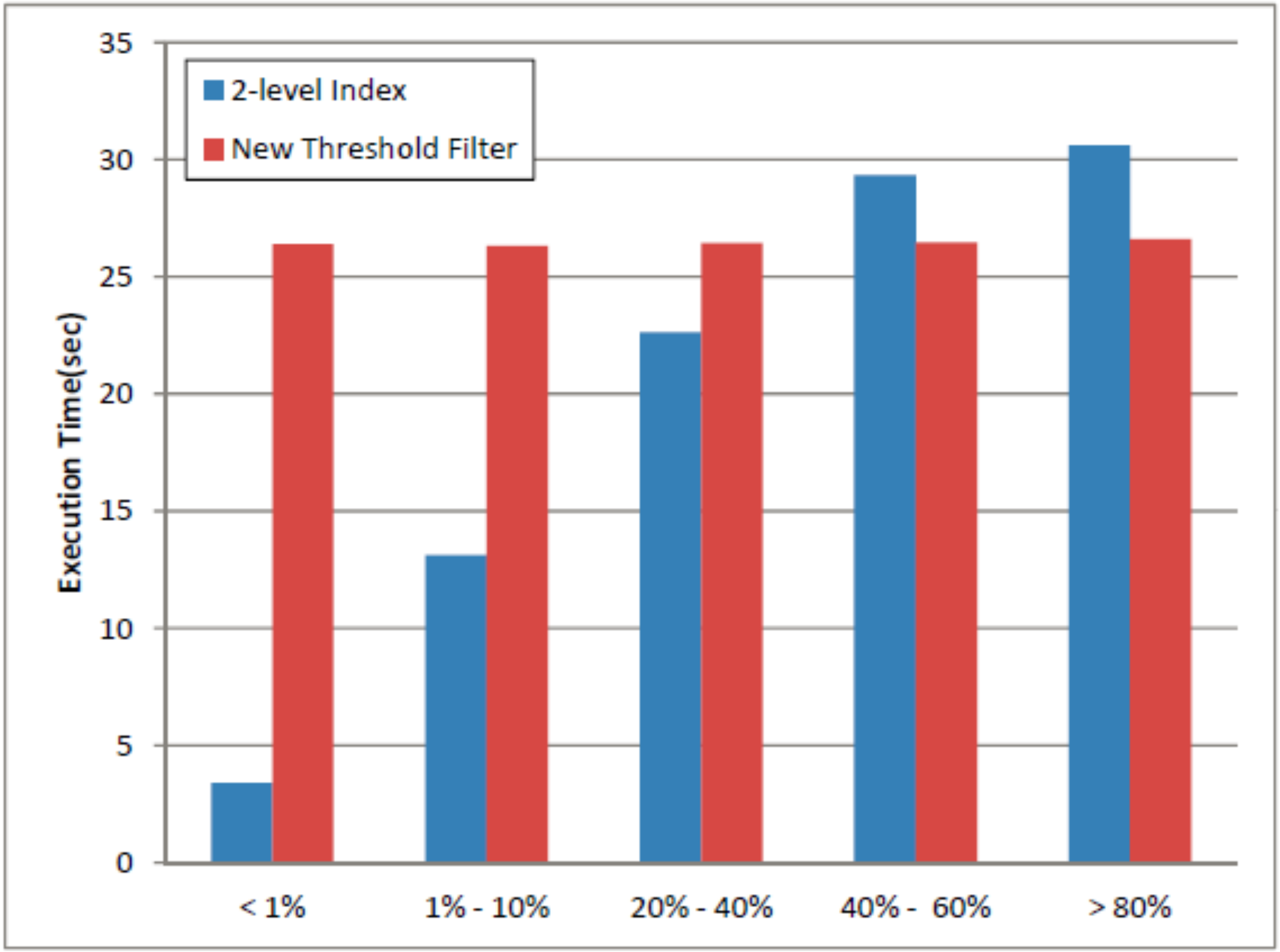
% of values vs. % of IDs (space) in query result



Remove Unstructured Grid Generation

- The first experiment showed that ParaView/VTK spends a lot of time generating new grids
  - This could be said for many filters – VTK tends to generate new grids too often, wasting time and space
- New filtering idea (seems obvious but it was "Aha!")
  - Instead of generating new grids after filtering, mark "unqualified" data values as NaN
  - Data stay on original grid
  - No extra grid generation (saves time and space)

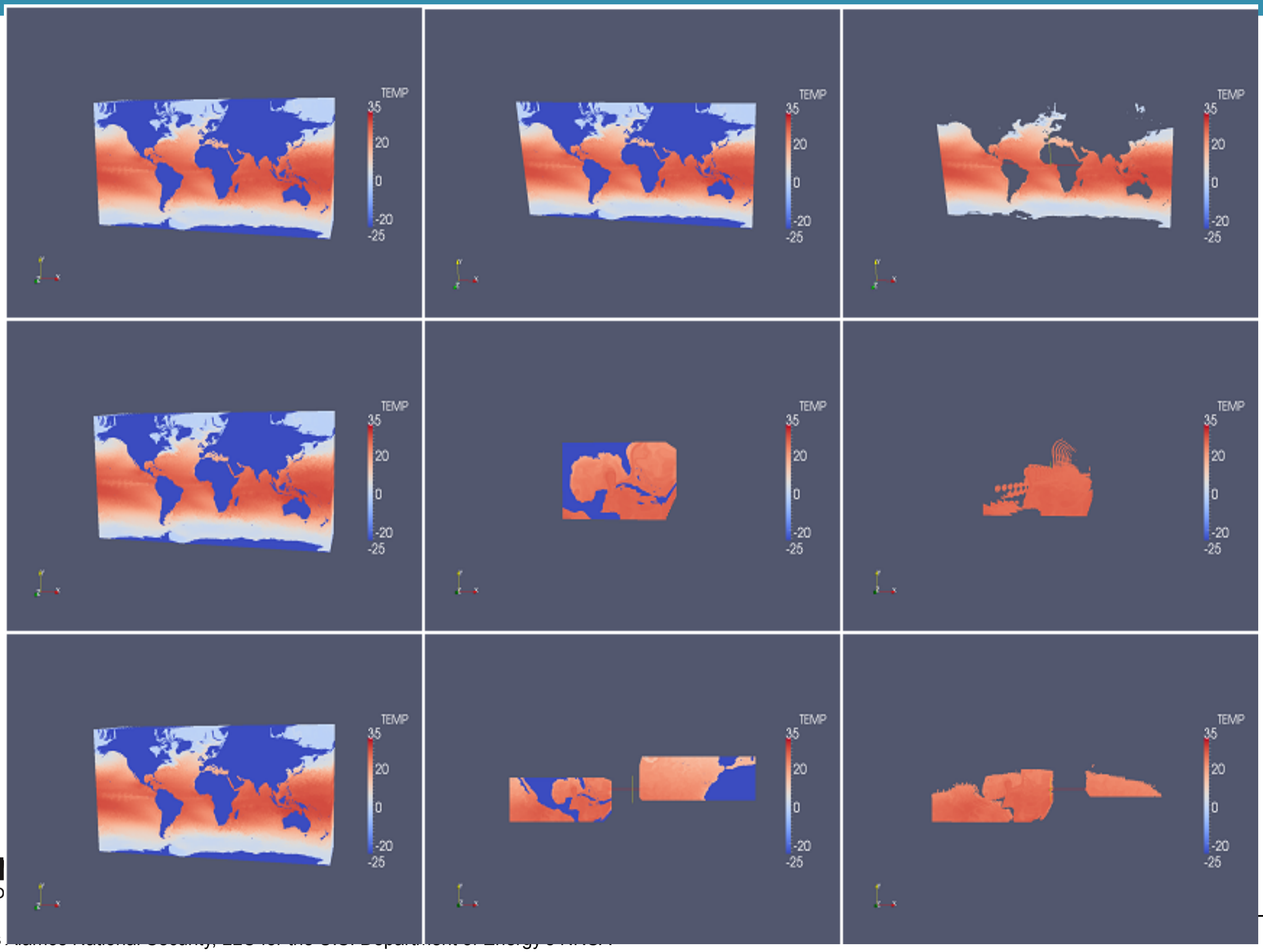
Indexing vs. new Threshold filter (set unqualified values to NaN)



Conclusion

- Use SQL queries to support flexible data subsetting
  - Translate query into operations
  - Use NetCDF/HDF5 API to support spatial subset
  - Use bitmap indexing to support value subsets
  - No need to move data into a database
- Reduced memory and time for query
  - Multi-level indexing can drastically improve time
  - Skipping extra grid generation steps in VTK can improve time and memory usage as well

Questions? Thanks for listening!



Los Alamos NATIONAL LABORATORY EST. 1943

Bitmap Compression (Bitmaps can be large)

- WAH compression
  - Compress the bit vectors based on continuous 0s or 1s
  - Can't subset the by the IDs (spatial dimensions) before the value indexing operation (assuming we don't add x, y, z to the bitmap index)
- Multi-Hash compression
  - Use multiple hash functions to set 1s for hash(id, value) for each 1 in a bit vector
  - Supports subsetting over both IDs (space) and values
  - Hash clashes for (id, value) to same array position (false positives)