Scalable Software Components for Ultrascale Visualization Applications

Wes Kendall, Tom Peterka, Jian Huang

SC Ultrascale Visualization Workshop 2010
11-15-2010

UltraVis
SciDAC Institute for Ultrascale Visualization

Primary Collaborators


Jian Huang


Tom Peterka


Rob Ross


Han-Wei Shen

SEELAB
University of Tennessee

OHIO STATE

THE UNIVERSITY of TENNESSEE

Argonne NATIONAL LABORATORY

Scalable Analysis Core Components

Our position:

-Scientists need to perform more analysis at run-time and post-simulation
  -No turn-key system exists
  -Data are often very large and complex
  -Always a need for custom applications
-Large data analysis has tough initial barriers
  -Lack of resources / venues
  -Steep learning curve (low-level MPI)
  -Bottlenecks are often application/platform-specific

Interactivity is pivotal – depends on scalability:

-Balance load (computation, communication)
-Minimize or optimize data movement (storage and network)
-Hide data movement (overlap with work)

Data bandwidth challenges:

-I/O
-Communication
  -Global reduction
  -Local nearest neighbor exchange
  -Partitioning and repartitioning
-Searching / sorting
  -Query-driven reduction for vis

Our answer – provide general and scalable core components:

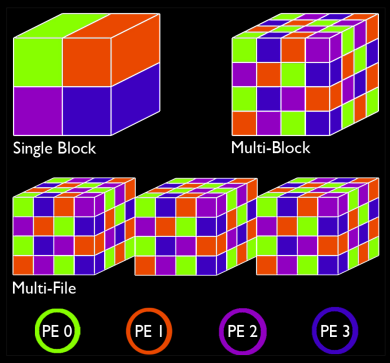
-Researchers can study new algorithms
-Vis groups (researchers and production) can build custom applications for current platforms
-Automatically obtain state of the art

BIL – The Block I/O Layer

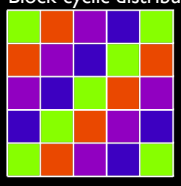
I/O patterns in analysis/visualization often revolve around block-oriented patterns. BIL abstracts these patterns across files and variables in raw, netCDF, and HDF formats.

BIL API:

-BIL_Add_{r,w}block{raw,nc,hdf}(
    block_bounds, file, variable, buffer);
-BIL_{Read,Write}();



Block-cyclic distribution



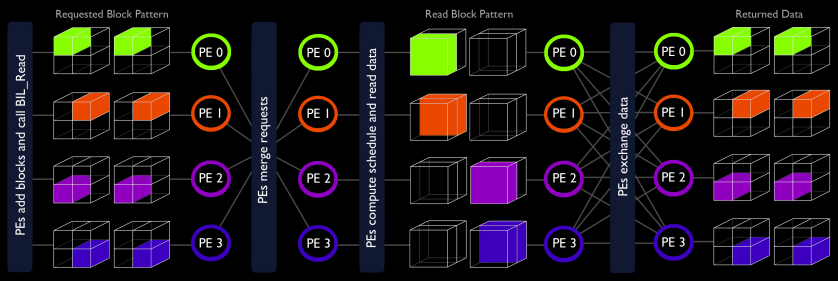
Disk access (first two rows)

Block distributions result in poor disk access

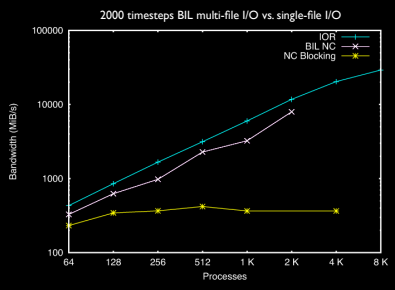
Applications (so far):

-Matrix decomposition (SVD)
-Volume rendering
-Parallel particle tracing (OSUFlow)
-Satellite data analysis

BIL – Implementation and Preliminary Results



BIL performs an application-level two-phase I/O over multiple variables and files

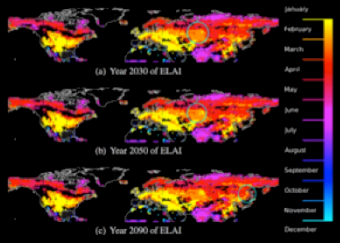



```
BIL_Add_rblock_raw(num_dims, block_starts,
block_sizes, file_name[0], MPI_FLOAT, &data[0]);

BIL_Add_rblock_raw(num_dims, block_starts,
block_sizes, file_name[1], MPI_FLOAT, &data[1]);

BIL_Read();
```

Code for this example

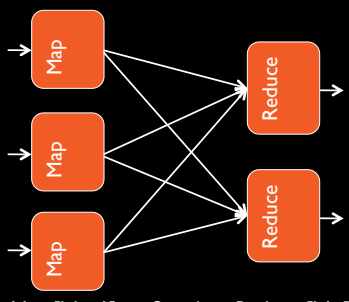
SQI – The Scalable Query Interface

Boolean range queries:
-Provide a powerful and intuitive interface for feature characterization
-Powerful enough by themselves to describe temporal trends



Beginning of Spring, [ELAI < .4]*T [ELAI > .4]?*

MapReduce:
-Parallel programming paradigm popularized by Google
-General interface + scalable implementation = countless efficient jobs



Map [k1, v1] Sort k Reduce [k1, [v1, v2, …]]

SQI is a combination of both querying and sorting
-Load balanced querying
-Heavily optimized parallel sorting (100 GB integer sort in < 0.2 s)

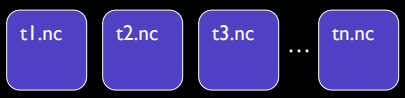
SQI – API and Usage Case: Drought Analysis

SQI API:
-SQI_Init(data_config);
-SQI_Query(ranges, return_attributes);
-SQI_Sort();

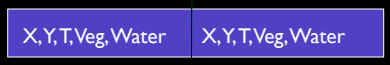
Data configuration:

```
data { files /dataset/dir .nc }
dim  { name X }
dim  { name Y }
dim  { name T, generate }
var  { name Veg }
var  { name Water }
```

Dataset layout:



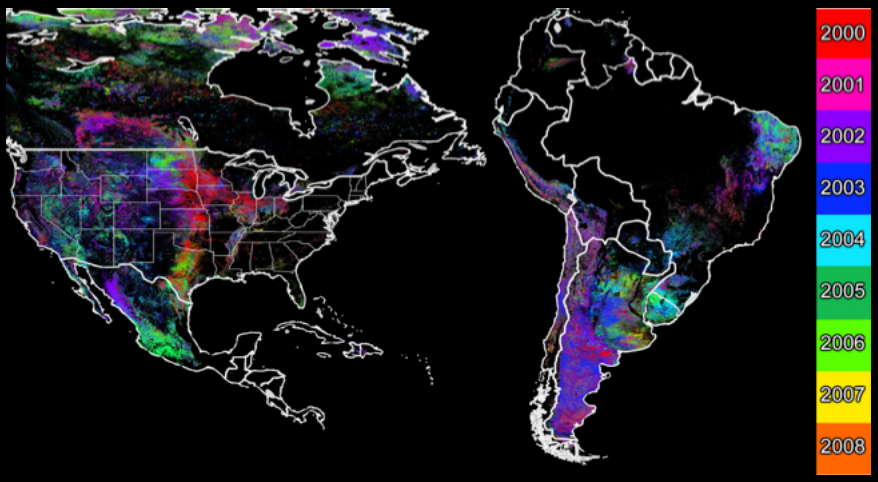
SQI_Init reads, "itemizes", and distributes points based on the configuration:



Example code (mostly pseudocode) for drought analysis:

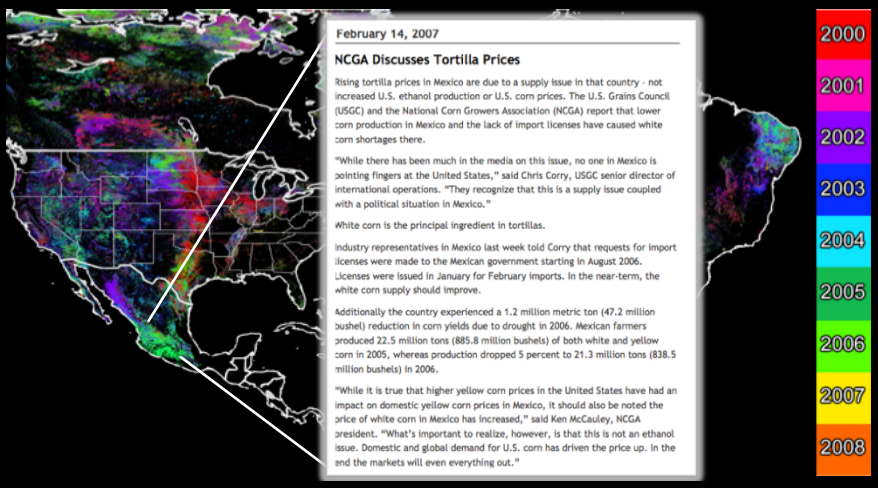
```
SQI_Init(data_config);
// Query low water and vegetation
for each hemisphere of world {
  SQI_Query(
    {hemisphere bounds, summer
timesteps, low veg, low water},
    { X, Y, T});
}
// Sort query results based on X, Y, T
SQI_Sort();
// For each {X,Y}, find the years that had
the most occurrences.
```


1.1 TB MODIS Drought Analysis



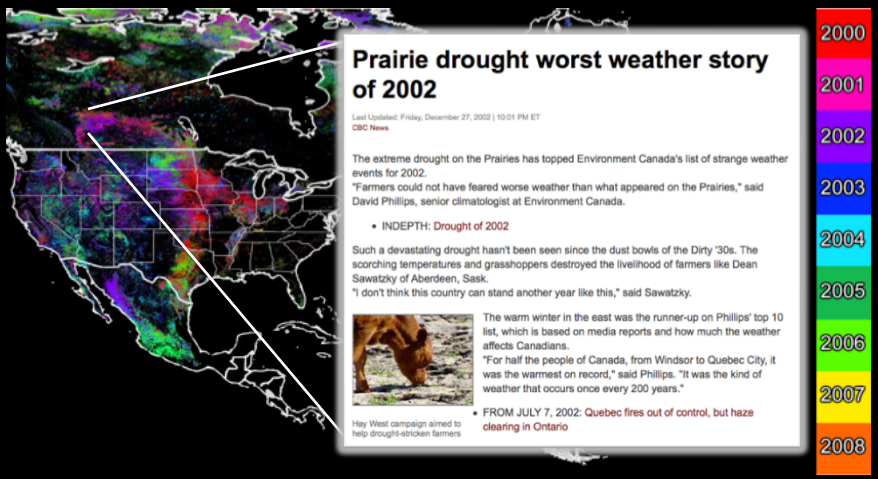
Cores	Read	Write	Filter	Distribution	Query	Sort	Analysis	Total Time
1K	85.23	0.68	47.11	22.40	1.01	14.17	0.22	170.82
2K	48.62	0.30	24.49	14.19	0.71	11.31	0.21	99.83
4K	41.27	0.31	11.80	8.78	0.34	9.18	0.10	71.78
8K	48.78	0.32	6.30	5.60	0.16	6.50	0.08	67.74
16K	46.56	0.30	3.45	3.02	0.08	4.20	0.07	57.68

1.1 TB MODIS Drought Analysis



Cores	Read	Write	Filter	Distribution	Query	Sort	Analysis	Total Time
1K	85.23	0.68	47.11	22.40	1.01	14.17	0.22	170.82
2K	48.62	0.30	24.49	14.19	0.71	11.31	0.21	99.83
4K	41.27	0.31	11.80	8.78	0.34	9.18	0.10	71.78
8K	48.78	0.32	6.30	5.60	0.16	6.50	0.08	67.74
16K	46.56	0.30	3.45	3.02	0.08	4.20	0.07	57.68

1.1 TB MODIS Drought Analysis



Cores	Read	Write	Filter	Distribution	Query	Sort	Analysis	Total Time
1K	85.23	0.68	47.11	22.40	1.01	14.17	0.22	170.82
2K	48.62	0.30	24.49	14.19	0.71	11.31	0.21	99.83
4K	41.27	0.31	11.80	8.78	0.34	9.18	0.10	71.78
8K	48.78	0.32	6.30	5.60	0.16	6.50	0.08	67.74
16K	46.56	0.30	3.45	3.02	0.08	4.20	0.07	57.68

Global and Local Communication

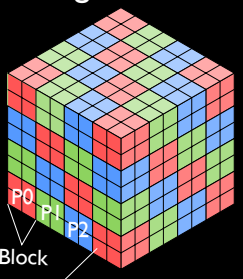
-Partitioning
- -Round robin
- -Graph
- -Repartitioning

-Global reduction
- -Merging
- -Compositing

-Local nearest-neighbor exchange
- -Particle tracing
- -Ghost cell exchange
- -Component labeling

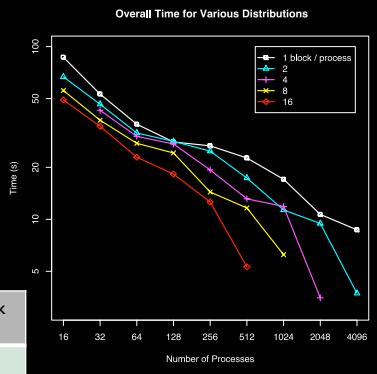
Partitioning and Repartitioning: Eg. Round Robin Assignment

1. Initial Partition
2. Compute, determine balance metric
3. Repartition to optimize metric



Partition data structure: Maintain local data only, not a global table of the partition. Do not want O(total data size) or O(total system size) memory use.

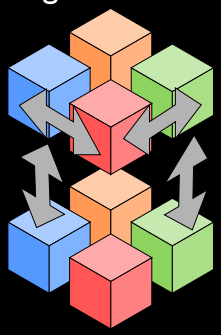
Global Block ID	Local Block ID	Block Data
0		
1	0	X,Y,Z
2		
3		
4	1	X,Y,Z
5		
6		
7	2	X,Y,Z



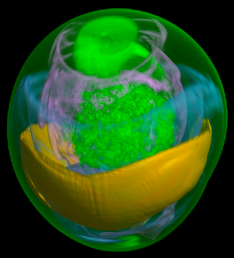
Particle tracing with 1, 2, 4, 8, and 16 blocks per process. A larger number of smaller blocks is better, to a limit.

Global Reduction Communication: Eg. Parallel Image Composition

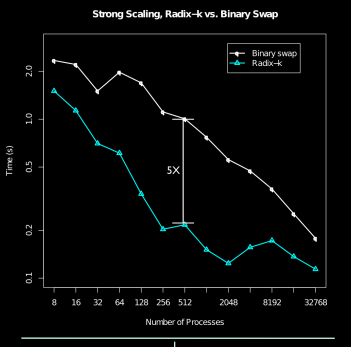
1. Round 1 exchange with k = 4, eg.
2. Round 2 exchange with k = 2, eg.
3. Repeat for as many rounds as desired (may be partial merge)



Parameters:
Number of rounds, k-values per round, swap or transfer, gather to root or parallel output, complete or partial merge



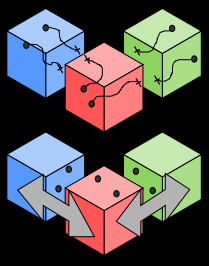
Example of using Radix-k image compositing to reduce multiple images into one

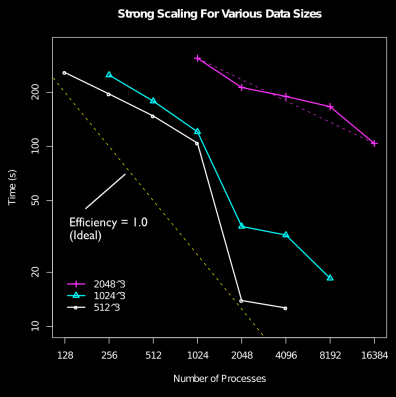


Scalability and performance speedup of Radix-k over binary swap reduction algorithms

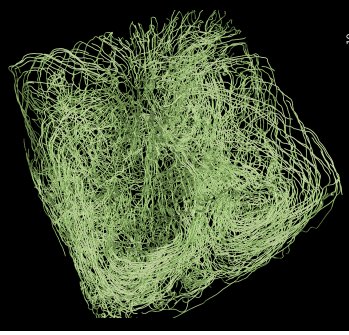
Local Nearest-Neighbor Communication: Eg. Parallel Particle Tracing

1. Perform local computations on blocks
2. Exchange objects among processes when they reach the block boundary.
3. Repeat



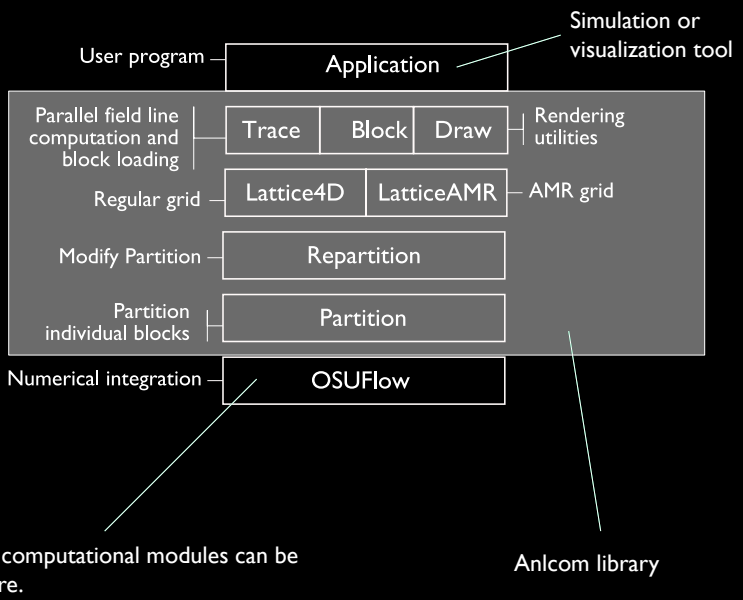


4096 streamlines generated from a Nek5000 flow field



Scalability for 3 data sizes and up to 16 K processes. End-to-end efficiency ~20%, including I/O

Library Organization



UltraVis
SciDAC Institute for Ultrascale Visualization

Conclusions / Where Do We Go From Here?

These tools have provided us with the ability to perform many types of analyses that were not previously possible due to overwhelming data demands.

We hope others can take advantage of our tools, study other large scale problems using them, and ultimately foster more community knowledge / involvement.

Future Goals:
BIL – Unstructured grid / AMR support
SQI – Out of core / further testing and development
ANLCom – Generalize for more applications and data types

BIL and SQI - http://seelab.eecs.utk.edu/

ANLCom - https://svn.mcs.anl.gov/repos/osuflow/anlcom

Kendall et al. SC `09 – Terascale Data Organization for Discovering Multivariate Climatic Trends
Petera et al. SC `09 – A Configurable Algorithm for Parallel Image-Compositing Applications
Kendall et al. EGPGV `10 – Accelerating and Benchmarking Radix-k Image Compositing at Large Scale