Scalable Parallel Building Blocks for Custom Data Analysis

Tom Peterka, Rob Ross (ANL)
Attila Gyulassy, Valerio Pascucci (SCI)
Wes Kendall (UTK)
Han-Wei Shen, Teng-Yok Lee, Abon Chaudhuri (OSU)

Morse-Smale Complex of combustion in the presence of a cross flow

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division
Survey of Science and Analysis Applications

- No two analyses are alike
- Analysis at scale is data-movement bound
- Data movement operations are common among different analyses

Particle tracing of thermal hydraulics flow
Information entropy analysis of astrophysics
Morse-Smale complex of hydrodynamics and combustion
Executive Summary
DIY helps the user parallelize their analysis algorithm with data movement tools.

High-level motivations and assumptions
- Large-scale analysis (visual and numerical) in parallel on distributed-memory HPC machines
- Scientists, visualization researchers, tool builders
- In situ, coprocessing, postprocessing
- Parallelizing from scratch is arduous
- Scalable data movement is key
- The user is the expert and may already have serial code for the analysis.

A common set of operations can be identified and encoded in a library
- Decompose the domain
- Assign subdomains to processors
- Access data and store results
- Combine local and global operations
- Balance load, minimize communication
- Overlap communication with computation
- Scale efficiently

Benefits
- Researchers can focus on their own work, not on building parallel infrastructure
- Analysis applications can be custom
- Reuse core components and algorithms for performance and programmer productivity
DIY Structure

Features
Parallel I/O to/from storage
- MPI-IO, BIL
Domain decomposition
- Decompose domain
- Describe existing decomposition
Network communication
- Global reduction (2 flavors)
- Local nearest neighbor

Library structure
Written in C++
C bindings
Future Fortran bindings

DIY usage and library organization

DIY
- I/O
  - Read Data
  - Write Results
- Decomposition
  - Blocking
  - Assignment
- Communication
  - Neighbor
  - Global

Utilities
- Parallel Compression
- Datatype Creation
- Parallel Sort
Data Model

Features
- All input data and output analysis data is represented as MPI data types
- MPI data types can represent any C/C++/Fortran language structure
- User does not serialize / deserialize types prior to use
- Zero copy at application level saves time and space
- Custom MPI data types are an advanced topic
- DIY assists in MPI data type creation

C data structure

```c
struct Particle {
    float[4] pt;
    int steps;
};
```

DIY MPI data type

```c
MPI_Datatype type;
struct map_block_t map[] = {
    {MPI_FLOAT, OFST, 4, offsetof(struct Particle, pt), 1},
    {MPI_INT, OFST, 1, offsetof(struct Particle, steps), 1},
};
DIY_Create_datatype(0, 2, map, &type);
```
I/O: Parallel Reading Data and Writing Analysis Results

Data input
- Application-level two-phase I/O
- Reads raw, netCDF, HDF5 (future)
- Read requests sorted and aggregated into large contiguous accesses
- Data redistributed to processes after reading
- Single and multi block/file domains.

Analysis output
- Binary
- General header/data blocks
- Footer with indices
- Application assigns semantic value to DIY blocks
- Compression

Kendall et al., Towards a General I/O Layer for Parallel Visualization Applications, CG&A ‘11
3 Communication Patterns

Nearest neighbor

Round 0
k = 4

Round 1
k = 2

Results

Swap-based reduction

Merge-based reduction

Round 0
k = 4

Round 1
k = 2

Results
// initialize
int dim = 3; // number of dimensions in the problem
int tot_blocks = 8; // total number of blocks
int data_size[3] = {10, 10, 10}; // data size
MPI_Init(&argc, &argv); // init MPI before DIY
DIY_Init(dim, ROUND_ROBIN_ORDER, tot_blocks, &nbblocks, data_size, MPI_COMM_WORLD);

// decompose domain
int share_face = 0; // whether adjoining blocks share the same face
int ghost = 0; // besides sharing a face, whether additional layers of ghost cells are needed
int ghost_dir = 0; // ghost cells apply to all or particular sides of a block
int given[3] = {0, 0, 0}; // constraints on blocking (none)
DIY_Decompose(share_face, ghost, ghost_dir, given);

// read data
for (int i = 0; i < nbblocks; i++) {
    DIY_Block_starts_sizes(i, min, size);
    DIY_Read_add_block_raw(min, size, infile, MPI_INT, (void**)&(data[i]));
}
DIY_Read_blocks_all();
Example API Continued

// your own local analysis

// merge results, in this example
// could be any combination / repetition of the three communication patterns
int rounds = 2; // two rounds of merging
int kvalues[2] = {4, 2}; // k-way merging, eg 4-way followed by 2-way merge
int nb_merged; // number of output merged blocks
DIY_Merge_blocks(in_blocks, hdrs, num_in_blocks, out_blocks, num_rounds, k_values, &MergeFunc, &CreateItemFunc, &DeleteItemFunc, &CreateTypeFunc, &num_out_blocks);

// write results
DIY_Write_open_all(outfile);
DIY_Write_blocks_all(out_blocks, num_out_blocks, datatype);
DIY_Write_close_all();

// terminate
DIY_Finalize(); // finalize DIY before MPI
MPI_Finalize();
Parallel Time-Varying Flow Analysis

Approach

- In core / out of core processing of time steps
- Simple load balancing (multiblock assignment, early particle termination)
- Adjustable synchronization communication

Algorithm

for (epochs) {
    read my process’ data blocks
    for (rounds) {
        for (my blocks) {
            advect particles
        }
        exchange particles
    }
}

Pathline tracing of 32 time-steps of combustion in the presence of a cross-flow

Parallelization within epochs and serialization across epochs adds greater flexibility.

Peterka et al., A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields, IPDPS ‘11
Parallel Information-Theoretic Analysis

Collaboration with the Ohio State University and New York University Polytechnic Institute

Objective

- Decide what data are the most essential for analysis
- Minimize the information losses and maximize the quality of analysis
- Steer the analysis of data based on information saliency

Information-theoretic approach

- Quantify Information content based on Shannon’s entropy
- Use this model to design new analysis data structures and algorithms

Shannon’s Entropy

The average amount of information expressed by the random variable is

\[ H(x) = - \sum_{i=1}^{n} p_i \log p_i \]
Parallel Topological Analysis
Collaboration with SCI Institute, University of Utah

- Transform discrete scalar field into Morse-Smale complex
- Nodes are minima, maxima, saddle points of scalar values
- Arcs represent constant-sign gradient flow
- Used to quickly see topological structure

Example of computing discrete gradient and Morse-Smale Complex
Two levels of simplification of the Morse-Smale complex for jet mixture fraction.

Gyulassy et al., The Parallel Computation of Morse-Smale Complexes, Submitted to IPDPS ‘12
Performance and Scalability

**Strong Scaling**

- **Information entropy**
- **Topological analysis**
- **Particle tracing**
Summary

Main ideas
- Scalable analysis is about moving, transforming, reducing, analyzing, storing data
- Scientists, researchers take ownership of their own analysis

Successes
- Supports numerous, diverse analysis techniques
- Enables serial algorithms to be parallelized
- Flexible combination of data movements
- Both postprocessing and in situ
- Efficient and scalable

Limitations
- Requires effort on the part of the user
- Needs a program and (expert?) programmer

Ongoing
Finish installing existing code for swap-based reduction
AMR, unstructured, particle decomposition
Hybrid parallelism?
Acknowledgments:

Facilities
Argonne Leadership Computing Facility (ALCF)
Oak Ridge National Center for Computational Sciences (NCCS)

Funding
DOE SciDAC UltraVis Institute
DOE SDMAV Exascale Initiative
ANL LDRD

https://svn.mcs.anl.gov/repos/diy/trunk

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division

LDAV’11 Symposium 10/24/11

“The purpose of computing is insight, not numbers.”
–Richard Hamming, 1962