Do-It-Yourself Data Analysis:
Selected Topics and Recent Adventures
(or, ten ways to win friends and parallelize data analysis)

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

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division
Argonne Data Science
Several levels of data movement for data-intensive science applications

Platform: Grids / clouds
Research: Collaborative / Remote Tasks
Groups: Futures Lab, CI
PIs: Foster

Platform: LCF
Research: Parallel analysis
Infrastructure
Groups: Futures Lab
PIs: Vishwanath

Platform: HPC machine
Research: Parallel analysis algorithms
Groups: Radix Lab
PIs: Peterka

WAN

LAN

Data-parallel
Executive Summary

DIY helps the user write data-parallel analysis algorithms.

Main ideas and Objectives
- Large-scale parallel analysis (visual and numerical) on HPC machines
- Scientists, visualization researchers, tool builders
- In situ, coprocessing, postprocessing
- Data-parallel problem decomposition
- Scalable data movement algorithms

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

Today’s talk
- Main design concepts, that include both
  - A DIY overview, as well as
  - Recent advances and new ideas
Observation #1: Data Analysis Comes in Many Flavors

- Particle tracing of thermal hydraulics flow
- Information entropy analysis of astrophysics
- Morse-Smale complex of combustion
- Voronoi tessellation of cosmology
## #1: Separate Analysis Ops from Data Ops

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Application Data Model</th>
<th>Analysis Data Model</th>
<th>Analysis Algorithm</th>
<th>Communication</th>
<th>Additional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Tracing</td>
<td>CFD</td>
<td>Unstructured Mesh</td>
<td>Particles</td>
<td>Numerical Integration</td>
<td>Nearest neighbor</td>
<td>File I/O, Domain decomposition, process assignment, utilities</td>
</tr>
<tr>
<td>Information Entropy</td>
<td>Astrophysics</td>
<td>AMR</td>
<td>Histograms</td>
<td>Convolution</td>
<td>Global reduction, nearest neighbor</td>
<td></td>
</tr>
<tr>
<td>Morse-Smale Complex</td>
<td>Combustion</td>
<td>Structured Grid</td>
<td>Complexes</td>
<td>Graph Simplification</td>
<td>Global reduction</td>
<td></td>
</tr>
<tr>
<td>Computational Geometry</td>
<td>Cosmology</td>
<td>Particles</td>
<td>Tessellations</td>
<td>Voronoi</td>
<td>Nearest neighbor</td>
<td></td>
</tr>
</tbody>
</table>

You do this yourself
Can use serial libraries such as OSUFlow, Qhull, VTK
(don’t have to start from scratch)
DIY Overview

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

Simulation
Flash, Nek5000, HACC

Visualization Tool
ParaView, VisIt

Analysis Library
ITL, Osuflow, Qhull, VTK

DIY

MPI

I/O
Read Data
Write Results

DIY
Decomposition
Blocking
Assignment

Communication
Neighbor
Global

Utilities
Parallel Compression
Datatype Creation
Parallel Sort
Observation #2: Application Data Model ≠ Analysis Data Model

HACC (cosmology) Data Model

```c
int num_particles;
float *xx, *yy, *zz;
float *vx, *vy, *vz;
float *phi;
int64_t pid;
uint16_t mask;
```

Tess (voronoi tessellation) Data Model

```c
float mins[3];
float maxs[3];
int num_verts;
int num_cells;
double *verts;
int *num_cell_verts;
int tot_num_cell_verts;
int *cells
double *sites;
int num_complete_cells;
int *complete_cells;
double *areas;
double *vols;
int tot_num_cell_faces;
int *num_cell_faces;
int *num_face_verts;
tot_num_face_verts;
int *face_verts;
```

Corollary: analysis X data model ≠ analysis Y data model
#2: Allow User to Define Data Model

<table>
<thead>
<tr>
<th>C data structure</th>
<th>DIY data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>float mins[3];</td>
<td>1. DIY_Datatype type;</td>
</tr>
<tr>
<td>float maxs[3];</td>
<td>struct map_block_t map[] = {</td>
</tr>
<tr>
<td>double *verts;</td>
<td>{ DIY_FLOAT, OFST, 3, offsetof(struct vblock_t, mins) },</td>
</tr>
<tr>
<td>double *sites;</td>
<td>{ DIY_DOUBLE, ADDR, v-&gt;num_verts * 3, DIY_Addr(v-&gt;verts) },</td>
</tr>
<tr>
<td>int *complete_cells;</td>
<td>{ DIY_DOUBLE, ADDR, v-&gt;num_cells * 3, DIY_Addr(v-&gt;sites) },</td>
</tr>
<tr>
<td>double *areas;</td>
<td>{ DIY_INT, ADDR, v-&gt;num_complete_cells, DIY_Addr(v-&gt;complete_cells) },</td>
</tr>
<tr>
<td>double *vols;</td>
<td>{ DIY_DOUBLE, ADDR, v-&gt;num_complete_cells, DIY_Addr(v-&gt;vols) },</td>
</tr>
<tr>
<td>int *num_cell_faces;</td>
<td>{ DIY_INT, ADDR, v-&gt;num_complete_cells, DIY_Addr(v-&gt;num_cell_faces) },</td>
</tr>
<tr>
<td>int *num_face_verts;</td>
<td>{ DIY_INT, ADDR, v-&gt;tot_num_cell_faces, DIY_Addr(v-&gt;num_face_verts) },</td>
</tr>
<tr>
<td>int *face_verts;</td>
<td>{ DIY_INT, ADDR, v-&gt;tot_num_face_verts, DIY_Addr(v-&gt;face_verts) },</td>
</tr>
<tr>
<td></td>
<td>{ DIY_FLOAT, OFST, 3, offsetof(struct vblock_t, maxs) },</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td></td>
<td>DIY_Create_struct_datatype(DIY_Addr(vblock), 10, map, dtype);</td>
</tr>
</tbody>
</table>

- Any C/C++/Fortran data structure can be represented as an DIY (MPI) data type
- DIY uses data type to fetch data directly from memory or storage
- User does not pack / unpack (serialize / deserialize) data
- Zero copy at application level saves time and space
- DIY helps make data type creation easier
The block is DIY’s basic unit of data. Original dataset is decomposed into generic subsets called blocks, and associated analysis items live in the same blocks. Blocks contain one or more instances of the data type described earlier.
All data movement operations are per block; blocks exchange information with each other using DIY’s communication algorithms. DIY manages and optimizes exchange between processes based on the process assignment. This allows for flexible process assignment as well as easy debugging.
#5: Time is Like Space, but Special

- Time often goes forward only
- Usually do not need all time steps at once

<table>
<thead>
<tr>
<th>4D</th>
<th>3D</th>
<th>×</th>
<th>1D Temporal Extent</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Xmax, Ymax, Zmax, Tmax)</td>
<td>(Xmax, Ymax, Zmax)</td>
<td>(Tmin)</td>
<td></td>
</tr>
<tr>
<td>4D Block</td>
<td>3D Spatial Extent</td>
<td>1D Temporal Extent</td>
<td></td>
</tr>
<tr>
<td>(Xmin, Ymin, Zmin, Tmin)</td>
<td>(Xmin, Ymin, Zmin)</td>
<td>(Tmax)</td>
<td></td>
</tr>
<tr>
<td>4D Neighborhood (not drawn)</td>
<td>3D Spatial Neighborhood</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hybrid 3D/4D time-space decomposition. Time-space is represented by 4D blocks that can also be decomposed such that time blocking is handled separately.
#6: Group Blocks into Neighborhoods

- Limited-range communication
- Allow arbitrary groupings
- Distributed, local data structure and knowledge of other blocks (not master-slave global knowledge)

Two examples of 3 out of a total of 25 neighborhoods
DIY provides point to point and different varieties of collectives within a neighborhood via its enqueue_item mechanism. Items are enqueued are subsequently exchanged (2 steps).
#8: Make Global and Neighborhood Communication Fast and Easy

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Tracing</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Global Information Entropy</td>
<td>Merge-based reduction</td>
</tr>
<tr>
<td>Point-wise Information Entropy</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Morse-Smale Complex</td>
<td>Merge-based reduction</td>
</tr>
<tr>
<td>Computational Geometry</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Region growing</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Sort-last rendering</td>
<td>Swap-based reduction</td>
</tr>
</tbody>
</table>

Factors to consider when selecting communication algorithm:
- associativity
- number of iterations
- data size vs. memory size
- homogeneity of data

DIY provides 3 efficient scalable communication algorithms on top of MPI. May be used in any combination.
3 Communication Patterns

**Nearest neighbor**

- **Round 0**
  - $k = 4$
  - Results: 8 9 10 11 12 13 14 15

- **Round 1**
  - $k = 2$
  - Results: 8 9 10 11 12 13 14 15

**Swap-based reduction**

- **Round 0**
  - $k = 4$
  - Results: 8 1
  - 10 2
  - 11 3
  - 12 4
  - 13 5
  - 14 6
  - 15 7

- **Round 1**
  - $k = 2$
  - Results: 8 10
  - 12 14

**Merge-based reduction**

- **Round 0**
  - $k = 4$
  - Results: 8
  - 12
  - 14
  - 15

- **Round 1**
  - $k = 2$
  - Results: 8
  - 12
#9: Support Applications

In Situ Unstructured Spectral Meshes With Help from MOAB

- Decomposition assigned by the application, not DIY
- DIY needs to get the decomposition from the app
- Call on MOAB for help with connectivity

Given the above mesh, assume the green block wants ghost cells in a given ghost radius of size $t$.

Result: the green block will have these cells (original green cells plus transparent cells).
void foo(imesh *mesh) {
    // MOAB mesh
    DIY_Init(num_blocks);
    for (num_blocks) {
        // query MOAB for verts in block
        get_adjacencies(hex, adj_verts);
        BlockBounds(bounds); // find min/max of verts
        // query MOAB for local neighbors of vertices
        get_adjacencies(adj_verts, adj_hexes);
        store adj_hexes in neighbors, num_neighbors
        // query MOAB for remote neighbors
        get_sharing_data(adj_verts, remote_handles, remote_procs);
        remote_data = remote_handles, remote_procs;
        // query MOAB for local vertex ids
        loc_vids[block] =
            id_from handle(shared_adj_verts);
    }
    DIY_Decomposed(blocks, bounds, remote_data,
        num_remote_data, loc_vids, neighbors,
        num_neighbors);
}

while (!done) {
    for (cells) {
        for (neighbors) {
            if (cell intersects neighbor extents + t &&
                cell was not sent already &&
                cell did not come from neighbor)
                post cell to neighbor;
        }
    }
    num_recvd = DIY_Exchange_neighbors();
    done = DIY_Check_done_all(!num_recvd);
}
### #10: Work With Other Libraries

We are helped by:

- **Zoltan** - partitioning for dynamic load balancing
- **MOAB** - unstructured mesh management
- **HDF5 and parallel netCDF** - high level storage
- **MPI** (of course)

We can help with:

- **ITL** - Information theoretic analysis
- **MSC** - Morse-Smale analysis
- **OSUFlow** - Particle tracing
- **Qhull** - Computational geometry
- **VTK** - Visualization and analysis filters
Acknowledgments:

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

Funding
DOE SDMAV Exascale Initiative
DOE Exascale Codesign Center

http://www.mcs.anl.gov/~tpeterka/software.html
https://svn.mcs.anl.gov/repos/diy/trunk

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