DIY Parallel Data Analysis

I'm sure my wife will appreciate all the DIY I'm doing around the house for her!

Image courtesy pigtimes.com

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division
Scientific Data Analysis Today

- Big science = big data, and
  - Big data analysis => big science resources
- Data analysis is data intensive.
  - Data intensity = data movement.
- Parallel = data parallel (for us)
  - Big data => data decomposition
  - Task parallelism, thread parallelism, while important, are not part of this work
- Most analysis algorithms are not up to the challenge
  - Either serial, or
  - Communication and I/O are scalability killers
Definition of Data Analysis

- Any data transformation, or a network or transformations.
- Anything done to original data beyond its original generation.
- Can be visual, analytical, statistical, or data management.

Example of a data flow network
Data Analysis Comes in Many Flavors

Visual
Particle tracing of thermal hydraulics flow

Statistical
Information entropy analysis of astrophysics

Topological
Morse-Smale Complex of combustion

Geometric
Voronoi tessellation of cosmology
You Have Two Choices to Parallelize Data Analysis

**By hand**

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Algorithm</td>
</tr>
<tr>
<td>Stochastic</td>
</tr>
<tr>
<td>OS / Runtime</td>
</tr>
</tbody>
</table>

```c
void ParallelAlgorithm() {
    ...
    MPI_Send();
    ...
    MPI_Recv();
    ...
    MPI_Barrier();
    ...
    MPI_File_write();
}
```

**or**

**With tools**

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Algorithm</td>
</tr>
<tr>
<td>Stochastic</td>
</tr>
</tbody>
</table>

**Data Movement**

| OS / Runtime |

```c
void ParallelAlgorithm() {
    ...
    LocalAlgorithm();
    ...
    DIY_Merge_blocks();
    ...
    DIY_File_write();
}
```
DIY

helps the user write data-parallel analysis algorithms by decomposing a problem into blocks and communicating items between blocks.

Features
Parallel I/O to/from storage
Domain decomposition
Network communication
Utilities

Library
Written in C++ with C bindings
Autoconf build system (configure, make, make install)
Lightweight: libdiy.a 800KB
Maintainable: ~15K lines of code, including examples

DIY usage and library organization
Nine Things That DIY Does

1. Separate analysis ops from data ops
2. Group data items into blocks
3. Assign blocks to processes
4. Group blocks into neighborhoods
5. Support multiple instances of 2, 3, and 4
6. Handle time
7. Communicate between blocks in various ways
8. Read data and write results
9. Integrate with other libraries and tools
Many different analysis operations share a small set of communication patterns. These communication kernels together with supporting utilities for decomposition and I/O can be encapsulated, optimized, and reused.
3 Communication Patterns

Nearest neighbor

Swap-based reduction

Merge-based reduction
Different Neighborhood Communication Patterns

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).

How to enqueue items for neighbor exchange

- DIY offers several options
- Send to a particular neighbor or neighbors, send to all nearby neighbors, send to all neighbors
- Support for periodic boundary conditions involves tagging which neighbors are periodic and calling user-defined transform on objects being sent to them
Interaction with Other Libraries

DIY by design doesn’t include input or output data models. Rather than re-inventing them, it can import and export those models.

Import: Replicate model using DIY_Decomposed(), explicitly providing blocks and neighbors to DIY
Export: Just use the other model API. DIY does not prevent you from making other library calls.
Writing a DIY Program

Documentation
- README for installation
- User’s manual with description, examples of custom datatypes, complete API reference

Tutorial Examples
- Block I/O: Reading data, writing analysis results
- Static: Merge-based, Swap-based reduction, Neighborhood exchange
- Time-varying: Neighborhood exchange
- Spare thread: Simulation and analysis overlap
- MOAB: Unstructured mesh data model
- VTK: Integrating DIY communication with VTK filters
- R: Integrating DIY communication with R stats algorithms
- Multimodel: multiple domains and communicating between them

Initialize

Decompose domain (regular grid & postprocessing)

List decomposition (irregular data or in situ)

Read data from storage

Data exists in memory

Local analyze

Communicate

Write analysis to storage

Finalize

Legend
- User
- DIY or user
Published Performance and Scalability

DIY

• Peterka, T., Ross, R.: Versatile Communication Algorithms for Data Analysis. 2012 EuroMPI Special Session on Improving MPI User and Developer Interaction IMUDI'12, Vienna, AT.

DIY applications

Particle tracing of \( \frac{1}{4} \) million particles in a \( 2048^3 \) thermal hydraulics dataset results in strong scaling to 32K processes and an overall improvement of 2X over earlier algorithms.
Computation of information entropy in 126x126x512 solar plume dataset shows 59% strong scaling efficiency.
Computation of Morse-Smale complex in $1152^3$ Rayleigh-Taylor instability data set results in 35% end-to-end strong scaling efficiency, including I/O.
For $128^3$ particles, 41% strong scaling for total tessellation time, including I/O; comparable to simulation strong scaling.
Recap and Looking Ahead

Done: Benefits
• Productivity
  • Express complex algorithms flexibly
    • Multiple blocks per process
    • Complete / partial reductions
    • Neighbor inclusion and communication
  • Simplify existing tasks
    • Custom data type creation
    • Compression
• Performance
  • Published scalability
  • Configurable algorithms

To Do: Research Directions
• Advanced decomposition
  • Block groups
• Improved communication algorithms
  • Less synchronous, more overlap with computation
• High-level communication operations
  • Ghost cell exchange, kernel convolution (stencil)
• Load balancing
  • Block overloading, dynamic reassignment
• Programming models
  • MPI + X on Mira, Titan
• Usability
  • Improved API
Acknowledgments:

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

Funding
DOE SDMAV Exascale Initiative
DOE Exascale Codesign Center
DOE SciDAC SDAV Institute

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