“I have had my results for a long time, but I do not yet know how I am to arrive at them.”

–Carl Friedrich Gauss, 1777-1855

DIY Parallel Data Analysis

Image courtesy pigtimes.com

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

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division
Preliminaries
Moving from Postprocessing to Run-Time Scientific Data Analysis in HPC

Postprocessing analysis and visualization

Run-time analysis and visualization
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.
Examples

Streamlines and pathlines

FTLE

Information entropy

Stream surfaces

Ptychography

Morse-Smale complex

Voronoi and Delaunay tessellation
Common Denominators

• Big science => big data, big machines
• Most analysis algorithms are not up to speed
  • Either serial, or
  • Overheads kill scalability
• Solutions
  • Process data closer to the source
  • Write scalable analysis algorithms
  • Parallelize in various forms
  • Build software stacks of useful and reusable layers
• Usability and workflow
  • Develop libraries rather than tools
  • Users write small main programs and call into libraries
Abstractions Matter: Think Blocks, not Tasks

- Block = unit of decomposition
- Block size, shape can be configured
  - From coarse to fine
  - Regular, adaptive, KD-tree
- Block placement is flexible, dynamic
  - Blocks per task
  - Tasks per block
  - Memory / storage hierarchy
- Data is first-class citizen
  - Separate operations per block
  - Thread safety

Parallel data analysis consists of decomposing a problem into blocks, operating on them, and communicating between them.
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>

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>
<tr>
<td>Data Movement</td>
</tr>
<tr>
<td>OS / Runtime</td>
</tr>
</tbody>
</table>

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

Two examples of 3 out of a total of 25 neighborhoods
Usage
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
Execution: Same Core and Spare Core

**Same core**
- Initialize
- Compute
- Get Decomposition
- Access data
- Local analyze
- Communicate
- Write analysis
- Finalize

**Spare core**
- Fork
- Compute
- Copy
- Join
- Initialize
- Get Decomposition
- Access data
- Local analyze
- Communicate
- Write analysis
- Finalize
Time: Static and Time-Varying

Static

1. Initialize
2. Decompose domain
3. Read data
4. Local analyze
5. Communicate
6. Write analysis
7. Finalize

Time-varying

1. Initialize
2. Decompose domain
3. Read data
4. Local analyze
5. Communicate
6. Write analysis
7. Finalize
8. For each time block
9. For each round
Communication: Merge and Swap Reduction, Neighbor Exchange

Merge and swap reduction

Initialise

Decompose domain

for each time block

Read data

Local analyze

Merge or Swap Reduction

for each round

exchange

reduce

Write analysis

Finalise

for each time block

Decompose domain

for each round

Read data

Local analyze

for each item

Enqueue

Exchange

Flush

Write analysis

Finalise
One Example in Greater Detail
Parallel Tessellation

We developed a prototype library for computing in situ Voronoi and Delaunay tessellations from particle data and applied it to cosmology, molecular dynamics, and plasma fusion.

Key Ideas

• Mesh tessellations convert sparse point data into continuous dense field data.
• Meshing output of simulations is data-intensive and requires supercomputing resources
• No large-scale data-parallel tessellation tools exist.
• We developed such a library, tess.
• We achieved good parallel performance and scalability.
• Widespread GIS applicability in addition to the datasets we tested.
Strong and weak scaling for up to $2048^3$ synthetic particles and up to 128K processes (excluding I/O) shows up to 90% strong scaling and up to 98% weak scaling.
Temporal structure dynamics: As time progresses, the range of cell volume and density expands, kurtosis and skewness increases. These statistics are consistent with the governing physics of the formation of high- and low-density structures over time and can perhaps be used to summarize evolution at given time steps.

Density estimation: Tessellations as intermediate representations enable accurate regular grid density estimators.
Recap

Block abstraction for parallelizing data analysis

Encapsulate data movement in a separate library

Define design patterns for data movement in HPC data analysis in terms of:

• Execution
• Temporal behavior
• Communication pattern

The benefits are:

• Abstraction, implementation independence
• Reuse, programmer productivity
• Standardization
• Benchmarking
Further Reading

**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**

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

https://bitbucket.org/diatomic/diy

Tom Peterka
tpeterka@mcs.anl.gov

Mathematics and Computer Science Division