Scalable High-Performance Analysis of Scientific Data

“Somewhere, something incredible is waiting to be known.”

–Carl Sagan

Feature tracking in fusion simulations (image courtesy of Hanqi Guo)
Overview
We study the use of supercomputers, besides their traditional role of simulation and modeling, for the analysis and visualization of scientific data.

We analyze data from scientific instruments and computer simulations generated by some of the largest science facilities in the world.

Scalable High-Performance Analysis of Scientific Data
Strategies

Software infrastructure, scalable algorithms, and application engagement

Software infrastructure

Develop libraries and middleware for the rapid development of scalable analysis algorithms, and for their in situ coupling to simulations and experiments.

Scalable algorithms

Use infrastructure to develop scalable, efficient, and parallel data analytics for high-volume and high-velocity data streams.

Application engagement

Domains such as high-energy physics, materials science, and environmental science provide test cases to validate and test solutions and drive research in new directions.
**Software**

*Programming models for parallel analysis and dataflow coupling, and analysis libraries built on top of them*

**DIY**  
[https://github.com/diatomic/diy](https://github.com/diatomic/diy)  
Dmitriy Morozov, Tom Peterka  
Scalable block-parallel data analysis

**Decaf**  
[https://github.com/tpeterka/decaf](https://github.com/tpeterka/decaf)  
Orcun Yildiz, Tom Peterka  
Dataflow communication for in situ HPC workflows

**Tess**  
[https://github.com/diatomic/tess](https://github.com/diatomic/tess)  
Dmitriy Morozov, Tom Peterka  
Parallel Delaunay and Voronoi tessellation

**FTK**  
[https://github.com/hguo/ftk](https://github.com/hguo/ftk)  
Hanqi Guo  
Parallel high-dimensional simplicial feature tracking

**MFA**  
[https://github.com/tpeterka/mfa](https://github.com/tpeterka/mfa)  
David Lenz, Tom Peterka  
Parallel multivariate functional approximation
Data Analytics Software Stack

Applications
Simulations, experiments, observations, ensembles, standalone tools, workflows

User Libraries
Analysis libraries, standard visualization/analysis packages

- FTK (Feature Tracking)
- Tess (Delaunay Tessellation)
- MFA (Functional Approximation)

Data Movement
Data movement building blocks within one component (DIY) and between components (Decaf)

- DIY (Block Parallelism)
- Decaf (Decoupled Dataflows)

System Libraries
Programming model and runtime

System Services
Storage systems, resource managers, schedulers
Projects, Applications

- MFA: Multivariate Functional Approximation
- Triple Convergence Workflows
- Digital Twins
- RAPIDS2 SciDAC Institute
- HEP SciDAC Partnership
- CANGA SciDAC Partnership
- CODAR ECP Co-design Center
Block-Parallel Programming
Encapsulate Data Movement in Middleware

Ad hoc

Application

Analysis Algorithm

Many individual MPI commands

OS / Runtime

Structured

Application

Analysis Algorithm

Data Movement

Compute | Exchange | Reduce | Write

OS / Runtime

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

void ParallelAlgorithm() {
    ...
    foreach(&LocalAlgorithm);
    exchange();
    reduce();
    write_blocks();
}

void LocalAlgorithm() {
    ...
}

Tess2
Gaia
VTK
FTK
Reeber
Many others

DIY
Blocks are units of work and communication. Blocks exchange information with each other using DIY's communication algorithms. DIY manages block placement in MPI processes and memory/storage.

=> flexible, high-performance programs that are easy to write and debug.
Decomposing Data Into Blocks

- The block is the basic unit of data decomposition.
- Blocks don’t have to be “blocky.”
- Any subdivision of data (e.g., a set of graph nodes, a group of particles, etc.) is a block.

Structured Grid

Unstructured Mesh

AMR Grid

Graph
DIY provides point to point and different varieties of collectives within a neighborhood via its enqueue/exchange/dequeue mechanism.

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
Global Communication Patterns

Round 0

k = 4

Round 1

k = 2

Results

Merge-reduce

Swap-reduce
Example Usage

```cpp
// initialization
Master master(world, num_threads, mem_blocks, ...);
ContiguousAssigner assigner(world.size(), tot_blocks);
decompose(dim, world.rank(), domain, assigner, master);

// compute on each block
master.foreach(&foo);

// neighbor exchange
master.exchange();

// reduction
RegularSwapPartners(dim, tot_blocks, k);
reduce(master, assigner, partners, &foo);

// callback function for each block
void foo(void* b, const Proxy& cp, void* aux)
{
    // receive
    for (size_t i = 0; i < in.size(); i++)
        cp.dequeue(cp.link()->target(i), incoming_data);
    // work on incoming data
    ...
    // send
    for (size_t i = 0; i < out.size(); i++)
        cp.enqueue(cp.link()->target(i), outgoing_data[i]);
}
```

```python
# initialization
import diy
w = diy.mpi.MPIComm()
m = diy.Master(w)

# callback function for each block
def foo(self, cp):
    # receive
    for i in range(len(cp.link())):
        gid = cp.link().target(i).gid
        o = cp.dequeue(gid)
    # work on incoming data
    ...
    # send
    for i in range(len(cp.link())):
        o = [cp.gid(), cp.link().target(i).gid]
        cp.enqueue(cp.link().target(i), o)

domain = diy.DiscreteBounds([0, 0, 0], [100, 100, 100])
d = diy.DiscreteDecomposer(3, domain, tot_blocks)
a = diy.ContiguousAssigner(w.size, tot_blocks)
d.decompose(w.rank, a, add_block)

# compute on each block
m.foreach(foo)

# neighbor exchange
m.exchange()

# reduction
partners = RegularSwapPartners(nbblocks)
diy.reduce(m, a, partners, foo1)
```
Parallel Voronoi and Delaunay Tessellation

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.

Peterka et al., High-Performance Computation of Distributed-Memory Parallel 3D Voronoi and Delaunay Tessellation. SC14.
Ptychographic Reconstruction at the APS

A test pattern, with 30 nm feature size, was raster scanned using a 5.2 keV X-ray beam. The total scanning time was about 20 minutes. Reconstruction time was ~2 minutes.

Nashed et al., Parallel Ptychographic Reconstruction. Optics Express 2014.
DIY is a programming model and runtime for HPC block-parallel data analytics.  
- Block parallelism  
- Flexible domain decomposition and assignment to resources  
- Efficient reusable communication patterns  
- Automatic dual in- and out-of-core execution  
- Automatic block threading
In Situ Workflows
Definition of In Situ Workflows

The coordination and communication among heterogeneous tasks, executing simultaneously in an HPC system, cooperating toward a common objective.
Why In Situ?

- Manage and reduce large data volumes from computations and experiments.

  Minimize data movement, save storage space, and boost resource efficiency—often while simultaneously increasing scientific precision.

- Enable scientific discovery from a broad range of data sources, over a wide scale of computing platforms.

  Improve real-time decision making, design optimization, and data-driven scientific discovery.
In Situ Workflows Today

BES workflow of dynamic ensemble of simulations and in situ detection of stochastic events

[Yildiz et al., 2019]
Neutrino event generation and parameter optimization for DUNE (2026).
Dataflows for In Situ Workflows

- **Task**: A serial or parallel program
- **Workflow**: Directed graph of tasks
- **Dataflow**: Information exchanged between workflow tasks
- **Decaf**: Dataflow middleware for in situ workflows
Time and Space Division

Workflow Graph

Producer
E.g., simulation
P
Consumer
E.g., analysis
C

Time Division Coupling

for each time step

Producer

Consumer

Space Division Coupling

for each time step

Producer

Data copy

Consumer
Data Redistribution

Producer task with 5 MPI ranks

Consumer task with 3 MPI ranks
LowFive Metadata Tree

HDF5 Data Model
- Hierarchical data model much like a UNIX file system
- Root is the file
- Internal nodes are groups
- Leaves are datasets or other objects (e.g., attributes)

LowFive Data Model
- Our in-memory replica of HDF5 metadata
- One object for every HDF5 object
- Shallow or deep data pointer or copy
LowFive and Wilkins

**Workflow definition for 1 producer - 2 consumer example**

**func: prod**
- `start_proc: 0`
- `nprocs: 3`
- `passthru: 0`
- `metadata: 1`
- `outports:
  - filename: step1.h5`
  - `dsets:
    - name: /group1/grid`
    - `ownership: 1`
  - name: /group2/particles`  

**func: con1**
- `start_proc: 3`
- `nprocs: 1`
- `passthru: 0`
- `metadata: 1`
- `inports:
  - filename: step1.h5`
  - `dsets:
    - name: /group1/grid`

**func: con2**
- `start_proc: 4`
- `nprocs: 1`
- `inports:
  - filename: step1.h5`
  - `dsets:
    - name: /group2/particles`
Key features:
• Decoupled workflow links with configurable dataflow: efficient parallel communication
• Data redistribution patterns
• Flow control
• YAML API for workflow definition

Orcun Yildiz, ANL
github.com/tpeterka/decaf
Multivariate Functional Approximation
Piecewise Approximation of Discrete Data Points  
(approximation: fitting with error)

A curve can be approximated by the sum of basis functions $N_i$ multiplied by control points $P_i$. 

$$C(u) = \sum_{i=0}^{n-1} N_{i,p}(u) P_i.$$ 

Where:
- $u$ = parameter $[0.0 - 1.0]$ 
- $C(u)$ = curve evaluated at $u$ (vector valued) 
- $N_{i,p}$ = $i_{th}$ basis function of degree $p$ 
- $P_i$ = $i_{th}$ control point (vector valued) 

Example of seven basis functions. Each is nonzero for only $p+1$ control points.
Piecewise Polynomials, Knots, and Control Points

- The final curve is composed of pieces of polynomials of a given degree.
- Knots are the breakpoints between pieces of polynomials:
  - 1-d knot vector for each direction.
  - Controls the width of each basis function.
- Control points are coefficients that when multiplied by the appropriate basis function give the final polynomial piece:
  - Control points are real n-d points in the same space as the input points.
  - Convex hull around the resulting polynomial pieces.

Three basis functions of degree 3. Each is nonzero for only \( p+1 \) knot spans.

Control points that multiply each basis function.

Resulting polynomial pieces of degree 3, \( C^2 \) everywhere (degree – 1).
Higher Number of Dimensions

Form tensor product of basis functions multiplied by control points.

\[ V(u_1, \ldots, u_d) = \sum_{i_1} \cdots \sum_{i_d} N_{i_1,p}(u_1) \times \cdots \times N_{i_d,p}(u_d) P_{i_1, \ldots, i_d} \]

For scientific data, we want any number of domain dimensions (e.g., x, y, z, t) and one or more science variables (e.g., pressure, temperature)

[Trivariate B-splines, Martin, Cohen, Kirby, 2001, 2008]
Modeling Scientific Datasets

S3D Combustion Dataset

- 3-d input domain, 1 variable (velocity mag.)
- 704x540x550 (209 M) input points
- Degree = 3

<table>
<thead>
<tr>
<th>Input Points</th>
<th>Output Points</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2.1 \times 10^8$</td>
<td>$1.4 \times 10^7$</td>
<td>0.05%</td>
</tr>
</tbody>
</table>
Scientific Dataset Results

CESM Climate Dataset

- 2-d input domain, 1 variable (FLDSC)
- 1800x3600 (6.5 M) input points
- Degree = 3

<table>
<thead>
<tr>
<th>Input Points</th>
<th>Output Points</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$6.5 \times 10^6$</td>
<td>$1.7 \times 10^6$</td>
<td>0.05%</td>
</tr>
</tbody>
</table>
Scientific Dataset Results

Rayleigh-Taylor Instability Dataset

- 3-d input domain, 1 variable (velocity mag.)
- 288x512x512 (75 M) input points
- Degree = 3

<table>
<thead>
<tr>
<th>Input Points</th>
<th>Output Points</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5x10^7</td>
<td>5.7x10^6</td>
<td>1.1%</td>
</tr>
</tbody>
</table>
Time-Varying 4d Model

3d model of 1 time step

<table>
<thead>
<tr>
<th>Input Points</th>
<th>Reduction</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1x10^6</td>
<td>8X</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

4d model of 47 time steps

<table>
<thead>
<tr>
<th>Input Points</th>
<th>Reduction</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.8x10^7</td>
<td>14X</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

Tornado dataset

- 4-d space-time input domain, 1 variable (velocity mag.)
- 128 x 128 x 128 input points x 47 time steps
- Degree = 4
Unstructured Data

XGC fusion dataset

- 2-d point cloud, 1 variable (potential)
- 56,980 input points
- Degree = 2
- Automatic constraints in regions of little/no data

<table>
<thead>
<tr>
<th>Input Points</th>
<th>Reduction</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7x10^4</td>
<td>2.5X</td>
<td>1.3%</td>
</tr>
</tbody>
</table>
Applications

Keys

• High-order evaluation anywhere w/o interpolation
• High-order closed-form differentiation anywhere w/o finite differences

Uses

• High-order visualization: isosurfacing, slicing, volume rendering
• Smoothing, simplification, filtering, arbitrary down- and up-sampling
• Comparing or remapping datasets with different discretizations
• Using derivatives for gradient-descent optimization
• Performing functional PCA and other analyses

Goal

Do not convert back to original data model for analysis and visualization applications
Multivariate High-Order Evaluation, Differentiation, Integration

- Any number of dimensions
- Evaluate anywhere, not just at input points
- Evaluate to full high-order precision of the model in closed form
- Not a low-order interpolation, estimate, finite difference, or quadrature
- $O((p + 1)^d)$ computational complexity per point evaluated, where $p$ is degree
\[ f(x, y) = \frac{x^4 + y^4 + x^2y^2 + xy}{x^3 + y^3 + 4} \]
Multiple Levels of Parallelism

- Block parallelism
  - Decompose domain into blocks and execute in distributed-memory (DIY)
- Task parallelism
  - Decompose block into curves in current dimension and execute in tasks (TBB)
  - Evaluate multiple points concurrently (TBB, SYCL, Kokkos)
- Vector parallelism
  - Vectorize linear algebra operations (Eigen)

2-d sinc synthetic dataset partitioned into 64 DIY blocks, each block modeled independently (colored by block ID)
Scalability Results

Strong scaling:
- S3D data from turbulent fuel-jet combustion in a cross-flow. Velocity magnitude is modeled.
- \((704 \times 540 \times 550)\) input, variable velocity magnitude
- 2 DIY blocks per MPI process
- Number of control points per direction \(\sim 20\%\) of # input points, per direction
- Overlap: 2 input points per direction
Software: MFA

Key features:

• High-dimensional and multivariate
• High-order evaluation anywhere inside domain w/o interpolation
• High-order closed-form differentiation anywhere inside domain w/o finite differences
• Scalable parallelism
• Adaptive refinement

David Lenz, ANL
github.com/tpeterka/mfa
Acknowledgments

Facilities
Argonne Leadership Computing Facility (ALCF)
Argonne Laboratory Computing Resource Center (LCRC)
Oak Ridge Leadership Computing Facility (OLCF)
National Energy Research Scientific Computing Center (NERSC)

Funding
DOE ASCR Research Programs
DOE SciDAC Programs
DOE ECP Program

People
Hanqi Guo, Orcun Yildiz, David Lenz