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)
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.
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.
PEDAL (Parallel Extreme-Scale Data Analytics) Team

Me  Hanqi Guo  Orcun Yildiz  Mukund Raj  David Lenz
Software

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

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

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

Tess  https://github.com/diatomic/tess
Dmitriy Morozov, Tom Peterka
Parallel Delaunay and Voronoi tessellation

FTK  https://github.com/hguo/ftk
Hanqi Guo
Parallel high-dimensional simplicial feature tracking

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
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 (eg., a set of graph nodes, a group of particles, etc.) is a block.
Linking Blocks in Neighborhoods

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

Examples of 3 neighborhoods in a regular grid, unstructured mesh, and 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

![Diagram showing how to enqueue items for neighbor exchange](image)
Global Communication Patterns

Round 0
\( k = 4 \)

Round 1
\( k = 2 \)

Results

Merge-reduce

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

// compute, neighbor exchange
master.foreach(&foo);
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)
{
    for (size_t i = 0; i < in.size(); i++)
        cp.dequeue(cp.link()->target(i), incoming_data);
    // do work on incoming data
    for (size_t i = 0; i < out.size(); i++)
        cp.enqueue(cp.link()->target(i), outgoing_data[i]);
}

Example Usage

C++ and Python
Computation of Morse-Smale complex in 1152³ Rayleigh-Taylor instability data set results in 35% end-to-end strong scaling efficiency, including I/O.

Gyulassy et al., The Parallel Computation of Morse-Smale Complexes, IPDPS ’12.
Load-Balanced Synthetic Benchmark

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.
Cosmology simulations have severe load imbalance. Tessellating meshes using a k-d tree instead of regular grid is 50X faster.

Morozov and Peterka, Efficient Delaunay Tessellation Through K-D Tree Decomposition, SC16.
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 Yesterday

Simulation \rightarrow Visualization

[Zajac, 1964]
In Situ Yesterday

Simulation

Visualization

[Parker et al., 1997, SCIRun]
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
Data Redistribution

Different ways to split and merge a data model between $m$ producer ranks and $n$ consumer ranks while preserving semantic integrity.

- Round-robin redistribution
- Contiguous redistribution
- Bounding box redistribution

```c
int gid;
float pos[3*NB_POS];

// Declaring the fields
SimpleFieldi fieldId(gid);
// One semantic item consists of 3 floats (e.g., position
ArrayFieldf fieldPos(pos, 3*NB_POS, 3);

pConstructData container;
container->appendData("gid", fieldId,
    DECAF_NOFLAG, DECAF_SHARED,
    DECAF_SPLIT_DEFAULT,
    DECAF_MERGE_KEEP_FIRST);

container->appendData("pos", fieldPos,
    DECAF_POS, DECAF_PRIVATE,
    DECAF_SPLIT_DEFAULT,
    DECAF_MERGE_APPEND);
```
Time and Space Division

Workflow Graph

Producer
E.g., simulation

Consumer
E.g., analysis

Time Division Coupling

for each time step

Producer

Consumer

Space Division Coupling

for each time step

Producer
Data copy

Consumer
Python Workflow Definition

```python
w = nx.DiGraph()

# Task declaration
w.add_node('node0', start_proc=0, nprocs=4)
w.add_node('node1', start_proc=7, nprocs=2)
w.add_node('node2', start_proc=11, nprocs=1)

# Dataflow declaration
w.add_edge('node0', 'node1', start_proc=4, nprocs=3,
            func='dflow', path='mod_path,
            prod_dflow_redist='count',
            dfow_con_redist='count')
w.add_edge('node1', 'node2', start_proc=9, nprocs=2,
            func='dflow', path='mod_path,
            prod_dflow_redist='count',
            dfow_con_redist='count')

wf.workflowToJSON(w, mod_path, "linear3.json")

producer.addOutput('position', 'float', 1)
producer.addOutput('velocity', 'float', 1)
...consumerA.addInput('position', 'float', 1)
consumerB.addInput('velocity', 'float', 1)
```
Software: Decaf

Key features:
- Decoupled workflow links with configurable dataflow: efficient parallel communication
- Data redistribution patterns
- Flow control
- Python 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.
Higher Number of Dimensions

Form tensor product of basis functions multiplied by control points.

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

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]
Adaptive Refinement (The Outer Loop)

Start with minimum number of knots and control points
For all dimensions
  For all curves in current dimension
    Solve for control points
    For all knot spans
      Evaluate points at input point locations
      If (error > threshold)
        Insert knot and control point in middle of knot span

Accelerations:
- Low-d (curve) evaluation in keeping with iterative fitting
- Sparsely sample curves and/or knot spans

Overall algorithm
Fast, Intelligent Knot Placement: Shortcut the Outer Loop

Yeh et al. J. CAD 2020

Method

Results

(a) Input data
(b) Third derivative
(c) Cumulative feature
(d) Knot placement
(e) Resulting approximation

(a) Input data
(b) Max error
(c) Root mean squared error

Results (1-d slice of S3D data)
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>
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>
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

- Any number of dimensions
- Evaluate anywhere, not just at input points
- Evaluate to full high-order precision of the model
- Separable dimensions (no curse of dimensionality)
- $O((p+1)^d)$ computational complexity per point evaluated, where $p$ is degree
Multivariate High-Order Differentiation

- Compute derivative basis functions
  - $O(p^2)$, same as ordinary basis functions ($0^{th}$ derivative)
- Then use same multivariate high-order evaluation algorithm as before
- All the same advantages of evaluation: differentiate anywhere at full high-order precision
- Partial derivatives too
- Applications: gradients, Jacobians, Hessians, feature detection, lighting
f(x, y) = \frac{(x^4 + y^4 + x^2y^2 + xy)}{(x^3 + y^3 + 4)}
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 \(~ 20\%\) 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

Examples, Utilities
- Encoding
- Decoding
- VTK Conversion

MFA Block
- Encode
  - Fixed
  - Adaptive
- Decode
  - Values
  - Derivatives
- Tmesh
- Knot Placement
- Parameterization

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, Mukund Raj, David Lenz