Parallel data analysis consists of decomposing a problem into blocks, operating on them, and communicating between them.
Preliminaries
Moving from Postprocessing to Run-Time Scientific Data Analysis in HPC
Parallel Data Analysis

- 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

**Question:** What is the best abstraction to express parallelism?
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.
The What and Why of a Block-Based Approach
Partition Data Into Blocks

The block is the basic unit of data decomposition. Original dataset is decomposed into generic subsets called blocks, and associated analysis items live in the same blocks. 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.
Create Multiple Decompositions

**Uses:**

1. Organize input (upper right)

2. Second decomposition suited for particular analysis (lower right)

3. Comparing multiple unrelated data domains (not shown)

- Original data
  - Arbitrary decomposition
  - 4 blocks indicated by color. No spatial locality assumed.

- Kd-tree decomposition
  - 4 new blocks spatially contiguous and load balanced by number of objects in each.

- Original block decomposition
  - Slab or pencil decomposition for FFT
  - 16 blocks, 3 procs indicated by color

- Need not be same number of blocks
All data movement operations are per block; blocks exchange information with each other using regular communication patterns. Runtime manages and optimizes exchange between processes based on the process assignment. This allows for flexible process assignment as well as easy debugging.
Handle Time

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

<table>
<thead>
<tr>
<th>4D</th>
<th>3D</th>
<th>1D</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_{max}, Y_{max}, Z_{max}, T_{max})</td>
<td>(X_{max}, Y_{max}, Z_{max})</td>
<td>(T_{min})</td>
</tr>
<tr>
<td>(X_{min}, Y_{min}, Z_{min}, T_{min})</td>
<td>(X_{min}, Y_{min}, Z_{min})</td>
<td>(T_{max})</td>
</tr>
<tr>
<td>4D Block</td>
<td>3D Spatial Extent</td>
<td>1D Temporal Extent</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.
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

gid = global block identification
lid = local block identification
pid = process identification
Communicate Locally and Globally Between Blocks

Round 0
k = 4

Round 1
k = 2

Results

Nearest neighbor

Swap-based reduction

Merge-based reduction
Different Neighborhood Communication Patterns

Provide point to point and different varieties of collectives within a neighborhood by enqueing and subsequently exchanging items (2 steps).

**How to enqueue items for neighbor exchange**

- 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
Left: 64 surfaces each seeded with 512 particles are advected in a 504x504x2048 simulation of a solar flare. Right: 64 surfaces each with 2K seeds in a 2K x 2K x 2K Nek5000 thermal hydraulics simulation. Time excludes I/O.

Lu et al., Scalable Computation of Stream Surfaces on Large Scale Vector Fields, SC14.
Easily Write OOC and Multithreaded Algorithms

With Dmitriy Morozov, LBNL


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.
Applications in Cosmology

Temporal structure dynamics: As time progresses, the range of cell volume and density expands, kurtosis and skewness increases, consistent with the governing physics.

Strong scaling (excluding I/O time) using CGAL for three time steps of HACC data of $1024^3$ particles. At later time steps, particles cluster into extremely dense and sparse regions, affecting load balance and reducing efficiency from 77% at $t=68$ to 14% at $t=499$. 
Recap

Block abstraction for parallelizing data analysis allows one to:

• Decompose data into blocks
• Assign blocks to processing elements
• Have several decompositions at once
• Overload blocks, migrate blocks between processing elements
• Communicate between blocks
• Migrate blocks in and out of core
• Thread blocks with finer-grained processing elements

All made possible by choosing blocks as the parallel abstraction

Think Blocks!
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

“The purpose of computing is insight, not numbers.”

–Richard Hamming, 1962

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