Foundations of Data-Parallel Particle Advection

Tom Peterka, Rob Ross Argonne National Laboratory
Boonth Nouanesengsay, Teng-Yok Lee, Abon Chaudhuri, Jimmy Chen, Kewei Lu, Han-Wei Shen
The Ohio State University
Wes Kendall, Jian Huang University of Tennessee, Knoxville
Zhanping Liu, Kentucky State University

Early stages of Rayleigh-Taylor Instability flow
Scientific Data Analysis and Specifically Particle Tracing

**General**
- Big science => HPC analysis
- Data analysis => data movement
- Parallel => distributed memory
data parallel
- Most analysis algorithms are not up to the challenge
  - Either serial or shared memory parallel
  - Communication and I/O are scalability killers

**Particle Tracing**
- Data sizes are large, and large numbers of particles are needed (hundreds of thousands) for accurate further analysis of field line features.
- High communication volume and data-dependent load balance make particle tracing challenging to parallelize and scale efficiently.
Moving from Postprocessing to Run-Time Scientific Data Analysis in HPC

Postprocessing particle tracing and visualization

Run-time particle tracing and postprocessing visualization
The Need for Parallel Particle Tracing

When data sizes are too large to move or process serially, parallel particle tracing needs to be executed on HPC machines. Results are available sooner, access to all data at full resolution is possible.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Grid size</th>
<th>Data size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>2048^3</td>
<td>98</td>
</tr>
<tr>
<td>RTI</td>
<td>2304 x 4096 x 4096</td>
<td>432</td>
</tr>
<tr>
<td>Flame</td>
<td>1408 x 1080 x 1100 x 32 time steps</td>
<td>608</td>
</tr>
</tbody>
</table>

Image courtesy Mark Petersen, Daniel Livescu, LANL. Code: CFDNS
Image courtesy Ray Grout, NREL, Hongfeng Yu, Jackie Chen, SNL Code: S3D
Simple Data Parallelization

1. Group data into blocks and assign blocks to processors.

2. Each voxel contains a velocity vector.

3. Advect particles along velocity vectors.

4. Exchange particles among processes when they reach the block boundary.

5. Repeat 3, 4

$8^3 = 512$ voxels
64 blocks
3 Processes

Voxel
Block
P0
P1
P2

$5 = 5$
OSUFlow and DIY

OSUFlow is a library of serial / parallel particle tracing functions that is parallelized using a library called DIY that helps the user write data-parallel analysis algorithms by decomposing a problem into blocks and communicating items between blocks.

**OSUFlow Features**
- Static / time-varying flows
- Regular / rectilinear / curvilinear / unstructured grids
- Fixed / adaptive step sizes
- Various integration methods

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

---

**Diagram**

- **Simulation**
  - Flash, Nek5000, HACC

- **Analysis Library**
  - ITL, Osuflow, Qhull, VTK

- **Visualization Tool**
  - ParaView, VisIt

- **DIY**
  - I/O
    - Read Data
    - Write Results
  - Decomposition
    - Blocking
  - Communication
    - Neighbor
    - Global
  - Utilities
    - Parallel Compression
    - Datatype Creation
    - Parallel Sort

**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 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
Howdy Neighbor

• Neighborhoods provide limited-range communication among arbitrary groupings of blocks with distributed, scalable data structures
• DIY provides different options within a neighborhood including sending an item to all neighbors near enough to receive it and periodic boundary conditions. Items are enqueued are subsequently exchanged (2 steps). Items are user-defined.

Two examples of 3 out of a total of 25 neighborhoods
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.
Configurable 3D / 4D Hybrid Algorithm

Internally, all blocks are 4D, but we allow separate grouping in space (blocks) and time (epochs) to control how much data are kept in-core in each epoch. This enables time-varying data to be traced natively in 4D, without requiring the entire 4D dataset to be resident in memory.

Algorithm
decompose domain into blocks
and assign blocks to processes
for (epochs) {
  read my process’ data blocks
  for (rounds) {
    for (my blocks) {
      advect particles
    }
    exchange particles
  }
}
Adjustable Synchronization Communication Algorithm

for (blocks in my neighborhood) {

    pack and send messages of block IDs and particle counts
    pack and send messages of particles

} 

wait for enough IDs and counts to arrive

for (IDs and counts that arrived) {

    receive particles

} 

Wait factor: the fraction of items for which to wait is adjustable. Typically we use 0.1 (wait for 10% of pending items to arrive in each round).
Wait Factor Communication Performance

Nonblocking point-to-point and waiting for all messages to arrive (wait factor = 1.0) offers little improvement over all-to-all communication, but dialing down the wait factor helps significantly.

MAX experiment data. Point to point with wait factor = 1.0 is virtually the same as all to all.

Flame stabilization data. Less synchronization (wait factor = 0.1) improves performance.
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. Most of this improvement comes from the wait factor. The left plot includes end-to-end time, including I/O, computation, and communication. The right image shows 8 thousand particles, much fewer than were actually tested.
Computational load is data dependent: data blocks containing vortices (sinks) attract particles and have high angular frequency requiring thousands more advection steps to compute than blocks with homogeneous flow. In the following slides, we evaluate three solutions: particle termination, multiblock assignment, and dynamic block re-assignment.

One process containing 4 blocks, with one block containing a vortex, can affect the load balance of the entire program execution.
Particle Termination

**Problem:** A busy process causes others to wait, which propagates throughout the system.

**Solution:** Particles that don’t exit the current block after one round are terminated. There is no loss of information because these particles have near-zero velocity.

<table>
<thead>
<tr>
<th>Without Particle Termination</th>
<th>With Particle Termination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. Computation Time</td>
<td>243 s</td>
</tr>
<tr>
<td>Total Execution Time</td>
<td>256 s</td>
</tr>
<tr>
<td></td>
<td>55 s</td>
</tr>
<tr>
<td></td>
<td>67 s</td>
</tr>
</tbody>
</table>

Jumpshots of 128 processes: process 105 is computation-bound and causes all others to wait. Terminating particles that do not leave the current block reduces maximum computation time and overall time.
Decomposing the domain into a larger number of smaller blocks helps, to a limit. Computational hot-spots are more likely to be amortized over a greater number of processes. Limiting factor: smaller blocks incur less computation and more communication because surface area / volume increases.

Example of 512 voxels decomposed into 64 blocks and assigned to 3 processes. Each process contains 21 or 22 blocks.

Decompositions of 1, 2, 4, 8, and 16 blocks per process in the MAX dataset, 512^3, 8K particles. Higher block numbers reduce the overall execution time. Early particle termination not applied in these tests.
MAX Experiment Results

Strong scaling, $512^3, 1024^3, 2048^3$ data, 128K particles, 1 time-step

Platform: IBM Blue Gene/P

Data courtesy Aleks Obabko and Paul Fischer, ANL
Rayleigh-Taylor Results

Weak scaling, $2304 \times 4096 \times 4096$ data, 16K to 128K particles, 1 time-step

Weak Scaling

Data courtesy Mark Petersen and Daniel Livescu, LANL

Platform: IBM Blue Gene/P
Flame Stabilization Results

Weak scaling, 1408 x 1080 x 1100 data, 512 to 16K particles, 1 to 32 time-steps

Platform: IBM Blue Gene/P

Data courtesy Ray Grout, NREL and Jackie Chen, SNL
Top: Streamlines of thermal hydraulics. Bottom: Pathlines of tornado

Top: Mesh for office airflow. Bottom: Streamlines for office airflow

Top: Mesh for blunt fin. Bottom: Streamlines for blunt fin

VTK Integration

VTK Parallel Reader → Adapter → VTK Parallel Streamline / Pathline Filter

OSUFlow

DIY

Adapter → VTK Renderer

Courtesy Zhanping Liu and Jimmy Chen
Summary

Keys to Successes
Configurable time-space data structure with variable size epochs and blocks
  Load as many time steps into memory as possible
Communication algorithm with adjustable synchronization
  Less synchronization is better, eg., wait for 10% of pending messages
Simple load balancing strategies
  Multiple blocks per process, particle termination

Ongoing / future work
Continuing to study dynamic load balancing and prediction using graph methods
AMR and unstructured grid parallelization
VTK integration
Hybrid messaging / threading parallel approaches
Recommended 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.

Particle Tracing Applications


Foundations of Data-Parallel Particle Advection

Thank You

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

Funding
US DOE SciDAC SDAV Institute

Subversion repositories
https://svn.mcs.anl.gov/repos/osuflow/trunk
https://svn.mcs.anl.gov/repos/diy/trunk

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division