Title: Portable Parallel Halo and Center Finders for HACC

Author(s): Christopher Sewell
Katrin Heitmann
Li-ta Lo
Salman Habib
James Ahrens

Intended for: Oak Ridge Leadership Computing Facility Users' Meeting, July 2014
PORTABLE PARALLEL HALO
AND CENTER FINDERS FOR
HACC

Christopher Sewell, LANL
Katrin Heitmann, ANL
Ollie Lo, LANL
Salman Habib, ANL
Jim Ahrens, LANL
Outline

Background: SDAV, PISTON, VTK-m, data-parallel abstractions, and halo analysis
Algorithms: Halo and center finding
Results: Moonlight (LANL), Stampede (TACC), and Titan (ORNL)
Extensions: in-situ integration and Poisson center finder
Future needs: current bottlenecks, expected trends, possible hardware and software solutions
Background

SDAV, PISTON, VTK-m, Data-Parallel Abstractions, and Halo Analysis
SDAV VTK-m Frameworks

Objective: Enhance existing multi/many-core technologies in anticipation of in situ analysis use cases with LCF codes

Benefit to scientists: These frameworks will make it easier for domain scientists’ simulation codes to take advantage of the parallelism available on a wide range of current and next-generation hardware architectures, especially with regards to visualization and analysis tasks

Projects
EAVL, Oak Ridge National Laboratory
DAX, Sandia National Laboratory
DIY, Argonne National Laboratory
PISTON, Los Alamos National Laboratory

Work on integrating these projects with VTK is on-going, in collaboration with Kitware
Algorithms for Science Applications Using VTK-m

- The PISTON component of VTK-m focuses on developing data-parallel algorithms that are portable across multi-core and many-core architectures for use by LCF codes of interest.
- PISTON consists of a library of visualization and analysis algorithms implemented using Thrust, as well as a set of extensions to Thrust.
- PISTON algorithms are integrated into LCF codes in-situ either directly or through integration with ParaView Catalyst.

PISTON isosurface with curvilinear coordinates

Ocean temperature isosurface generated across four GPUs using distributed PISTON

PISTON integration with VTK and ParaView

LA-UR-14-21116
Brief Introduction to Data-Parallelism and Thrust

What algorithms does Thrust provide?

- Sorts
- Transforms
- Reductions
- Scans
- Binary searches
- Stream compactions
- Scatters / gathers

**Challenge:** Write operators in terms of these primitives only

**Reward:** Efficient, portable code

---

<table>
<thead>
<tr>
<th>input</th>
<th>4 5 2 1 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>transform (+1)</td>
<td>5 6 3 2 4</td>
</tr>
<tr>
<td>inclusive_scan (+)</td>
<td>4 9 11 12 15</td>
</tr>
<tr>
<td>exclusive_scan (+)</td>
<td>0 4 9 11 12</td>
</tr>
<tr>
<td>exclusive_scan (max)</td>
<td>0 4 5 5 5</td>
</tr>
<tr>
<td>transform_inscan (*2, +)</td>
<td>8 18 22 24 30</td>
</tr>
<tr>
<td>for_each (-1)</td>
<td>3 4 1 0 2</td>
</tr>
<tr>
<td>sort</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>copy_if (n % 2 == 1)</td>
<td>5 1 3</td>
</tr>
<tr>
<td>reduce (+)</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input1</th>
<th>0 0 2 4 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>input2</td>
<td>3 4 1 0 2</td>
</tr>
</tbody>
</table>

| upper_bound | 3 4 2 2 3 |
| permutation_iterator | 4 8 0 0 2 |
Halo Analysis

- An important and time-consuming analysis function within HACC is finding halos and the centers of those halos.
- HACC is designed as an MPI+X code, but the analysis operators are parallelized only among MPI ranks, because of the difficulty in porting different X implementations (OpenMP, CUDA, etc.) across all the architectures on which HACC is run.

*Image of the matter density field, produced by HACC simulation
Image credit: Joe Insley and the HACC team, Argonne*
Definitions

- Friend-of-friends (FOF) halo: connect each particle to all “friends”, i.e., all other particles within a specified “linking length” of it; two particles will end up in the same “halo” if there exists any chain of “friends” between them.
- Most connected particle (MCP) center: the particle within a halo with the most “friends”.
- Most bound particle (MBP) center: the particle within a halo with the lowest potential, where the potential for a given particle is computed as the sum over all other particles of the negative of mass divided by distance.

\[
\text{MBP center: } \arg\min_{i \in H} \left( - \sum_{j \in H, j \neq i} \frac{m_j}{d_{ij}} \right)
\]
Algorithms
Halo and Center Finding
Halo Finder Algorithm: Connected Components

\[
\text{for all vertices } i \quad D(i) := i \\
\text{while } (\text{true}) \{ \\
\quad \text{// Graft trees onto smaller} \\
\quad \text{// vertices of other trees} \\
\quad \text{for all } (i, j) \in E \quad \text{pardo} \\
\quad \quad \text{if } (D(i) = D(D(i)) \text{ and } D(j) < D(i)) \\
\quad \quad \quad \text{set } D(D(i)) := D(j) \\
\quad \text{// If all the vertices are in} \\
\quad \text{// rooted stars, then exit} \\
\quad \text{for all vertices } i \quad \text{pardo} \\
\quad \quad \text{set } \text{star}(i) := \text{true} \\
\quad \text{for all vertices } i \quad \text{pardo} \\
\quad \quad \text{if } (D(i) \neq D(D(i))) \\
\quad \quad \quad \text{set } \text{star}(i), \text{star}(D(i)), \text{star}(D(D(i))) := \text{false} \\
\quad \text{for all vertices } i \quad \text{pardo} \\
\quad \quad \text{set } \text{star}(i) := \text{star}(D(i)) \\
\quad \quad \text{if } (\text{star}(i) \text{ for all } i) \quad \text{break} \\
\quad \text{// Pointer jumping on each vertex} \\
\quad \text{for all } i \quad \text{pardo set } D(i) := D(D(i)) \\
\} 
\]

Reference: An Introduction to Parallel Algorithms, Joseph JáJá, 1992

LA-UR-14-21116
Halo Finder Algorithm: Computing Edges on the Fly

- If we define an edge to exist between two particles if and only if their distance is less than the linking length, the connected components solution is the FOF halos.
- However, it could take $O(n^2)$ time and $O(n^2)$ memory to directly compute and store all edges.
- Instead, partition the domain into bins with edge length equal to the linking length.
  - Friends can only exist in its own bin or one of 27 neighbor bins.
  - Compute edges on the fly in the algorithm, comparing each particle to each other particle in its bin and its neighbor bins to see which edges exist.
  - The number of bins (most empty) may be too large to store pointer to each, so instead store only pointers to neighbor bins for each particle.
  - Neighbor bins will be located in 1D vector in groups of three, so 9 not 27 pointers needed to neighbors for each particle.
Halo Finding: Binning Example

Two vectors are computed, $N_1$ and $N_2$, which contain the beginning and ending of three ranges (nine in 3D) in the sorted particle vectors for which each particle will need to search for its potential friends.
Center Finding Algorithms

- Most connected particle can be found easily by counting the number of friends for each particle during any iteration through the virtual edge list, and then using a max-reduce to find the maximum.
- Similarly, an approximation of DB scan can exclude any particle with too few friends from all edges.
- Most bound particle can be found by computing the potential for each particle in a highly parallel brute force approach.
- Centers for all halos can be computed simultaneously using segmented vectors.
MBP Center Finding Example

The inputs are the particle ids \((I)\), coordinates \((X, Y)\), and halo id \((D)\) found using the halo finding algorithm. The output is a vector \(C\) containing for each particle the id of the MBP center for its halo.
Results

Moonlight, Stampede, and Titan
Results: Moonlight

**FOF Halo and MBP Center Finding**

- 1024^3 particles on 128 Moonlight nodes

**FOF Halo and MCP Center Finding**

- 1024^3 particles on 128 Moonlight nodes

**Graphs:**

- **FOF Halo and MBP Center Finding**
  - Time in seconds
  - Original 1 rpn
  - Original 16 rpn
  - PISTON

- **FOF Halo and MCP Center Finding**
  - Time in seconds
  - Original 1 rpn
  - Original 16 rpn
  - PISTON

**FOF + MBP:** PISTON ~4.9x faster than original with 16 rpn

**FOF + MCP:** PISTON ~2.5x faster than original with 16 rpn
Results: Visual comparison of halos

Original Algorithm    VTK-m Algorithm

LA-UR-14-21116
• To demonstrate the portability of our algorithms, the same code was compiled to the Thrust OpenMP backend (including our own OpenMP implementation of scan) and run on a 256³ particle data set on an Intel Xeon Phi SE10P (MIC) Coprocessor on a single node of Stampede at TACC
• PISTON version scales to more cores than running the existing serial algorithms with multiple MPI processes
This test problem has ~90 million particles per process. Due to memory constraints on the GPUs, we utilize a hybrid approach, in which the halos are computed on the CPU but the centers on the GPU. The PISTON MBP center finding algorithm requires much less memory than the halo finding algorithm but provides the large majority of the speed-up, since MBP center finding takes much longer than FOF halo finding with the original CPU code.
Results: Large Run on Titan

• Because the memory requirements increase with the number of MPI processes due to overload regions, the HACC simulation with this data can only use one MPI process per node, along with the associated GPU, given the memory available on Titan.
• Thus, there is an even greater potential for speed-up by utilizing the GPUs.
• The performance improvements using PISTON on GPUs allowed halo analysis to be performed on a very large $8192^3$ particle data set across 16,384 nodes on Titan for which analysis using the existing CPU algorithms was not feasible.
Extensions

*In-situ* Integration and Poisson Center Finder
In-situ Integration in HACC

- Successfully ran 500 time-step, $512^3$ particle simulation on Moonlight using our GPU halo and center finders integrated with HACC in-situ
- Completed prototype integration with CosmoTools
Extension: Potential Field Algorithm

- Optimization for MBP center finder: for each halo...
  - Superimpose a grid over the (extended) extents of the halo
  - Estimate particle density on the grid using binning
  - Solve Poisson equation for the potential on the grid using FFT (actually DST for zero boundary conditions approximation)
  - Find the grid point with minimum potential
  - Search around the neighborhood of the minimum potential grid for the particle with minimum potential
  - Return the position of such particle as the halo center
Future Needs

Current Bottlenecks, Expected Trends, and Possible Hardware and Software Solutions
Future Needs: Memory

Current bottlenecks
- Main memory: $8192^3$ simulation run with only 1 rpn because number of “ghost” particles increases with total number of ranks, and even 2 rpn exceeded total system memory
- GPU memory: Halo finding performed on CPU because of GPU memory limitations

Expected future trends
- Growth in computational rates will outpace growth in memory capacity and bandwidth

Possible solutions
- Hardware: More memory
- Software: Streaming and external memory algorithms
Future Needs: Resiliency

- Current bottlenecks (16k node simulation)
  - Start-up failure roughly every other time
  - After successful startup, failure within 12 hours about 1/3 of the time

- Expected future trends
  - Systems will have more nodes with more cores; more things that could fail

- Possible solutions
  - Hardware: Components with longer MTBF
  - Software: More robust run-time schedulers
Future Needs: Portability

Current bottlenecks
- Simulation code has to be rewritten to run, or at least to run efficiently, on new architectures
- Some potential optimizations have been foregotten because platform-specific optimizations would inhibit portability

Expected future trends
- More types of accelerators will come to market; individual systems will become more heterogeneous

Possible solutions
- Hardware: Standardize on an architecture (not likely)
- Software: Write programs using higher-level abstractions