Performance Portable Halo and Center Finding in HACC

Li-Ta Lo

Jon Woodring

Chris Sewell

Adrian Pope

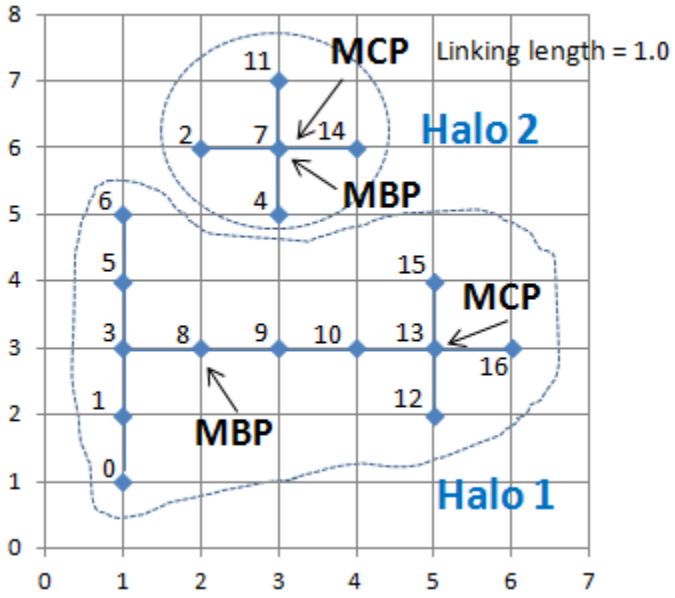
Katrin Heitmann

DOE COE Meeting

August 24, 2017

Halo and Halo Center

- Friends of Friends Halo
  - Connect each particle to other particles within a *linking length*
  - Two particle end up in the same Halo if there is a chain of friends between them
- Most Connected Particle (MCP)
  - The particle within in a Halo with the most friends
- Most Bound Particle (MBP)
  - The particle within the Halo with lowest potential
  - Generally not the same as MBP



Un-Accelerated Halo Finding in HACC

- Particles are distributed to compute nodes according to their domain decomposition.
- Overload zone are defined so the largest halo will be fully contained within one single node.
- Each node executes a KD-Tree based halo finding algorithm to find halos.
- Parallel halo finder merges the halos found such that a unique set of halo is reported.

Accelerated Halo Finding in HACC

- Particles are distributed to compute nodes according to their domain decomposition.
- Overload zone are defined so the largest halo will be fully contained within one single node.
- Each node executes *a performance portable* halo finding algorithm to find halos.
- Parallel halo finder merges the halos found such that a unique set of halo is reported.

Performance Portability through Parallel Primitives

- Parallel algorithms provided by the Thrust library
  - A large subset has been standardized as C++17 parallel algorithm library.

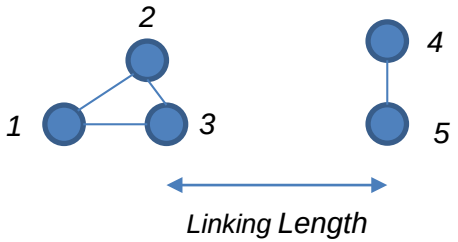
Algorithm	Example
Sort	[5 4 2 1 3] -> [1 2 3 4 5]
Transform(+1)	[5 4 2 1 3] -> [6 4 3 2 4]
Inclusive Scan	[5 4 2 1 3] -> [5 9 11 12 15]
Exclusive Scan	[5 4 2 1 3] -> [0 5 9 11 12]
Reduce(+)	[5 4 2 1 3] -> 15
Binary Search	Needles: [4 3 1 0 2] Haystack: [0 0 2 4 8] Upper bound: [3 4 2 2 3]

Performance Portability through Parallel Primitives

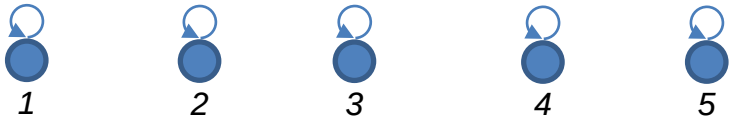
- Parallel algorithms provided by the Thrust library
  - The most important/useful/difficult part were left out.

Algorithm	Example
Sort By Key	Keys: [0 2 0 1 1] -> [0 0 1 1 2] Values: [5 4 2 1 3] -> [5 2 1 3 4]
Inclusive Scan By Key	Keys: [0 0 1 1 1] Values: [5 4 2 1 3] -> [5 9 2 3 6]
Exclusive Scan By Key	Keys: [0 0 1 1 1] Values: [5 4 2 1 3] -> [0 5 0 2 3]
Reduce(+) By Key	Keys: [0 0 1 1 1] -> [0 1] Values: [5 4 2 1 3] -> [9 6]

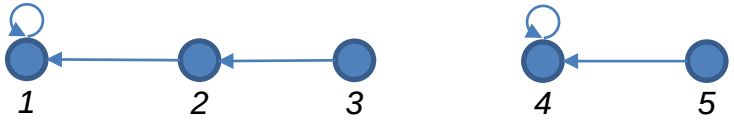
Halo Finder Algorithm: Parallel Connected Component



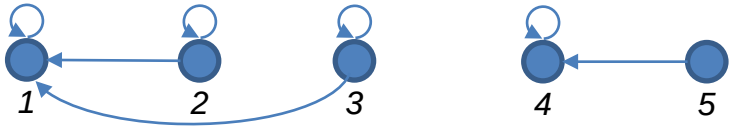
Initialize



Parallel Edge Connection



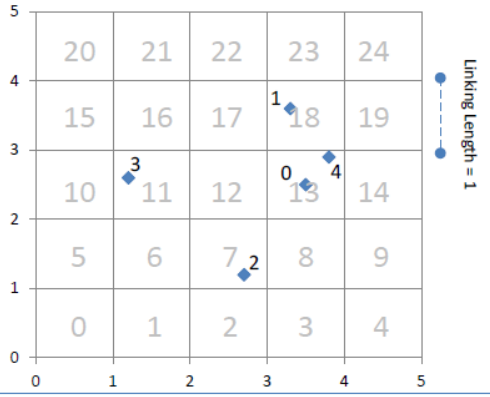
Pointer Jumping



Halo Finder Algorithm: Computing Edge

- If we define an edge exists between pairs of particles within linking length, connected component algorithms give the correct FoF Halo.
- However, it requires O(n^2) of space and time.
- We can improve the algorithm by binning particles into bins with edge length equal to linking length.
  - Each particle only need to search the 27 neighbor cells.
  - Use a sparse representation of bins and pointers to the particles in the bins.
  - Pointers are store as three groups of neighbor bins thus only 9 pointers rather than 27.

Binning Particles

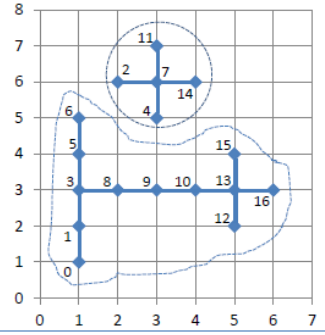



```
I          0        1        2        3        4
X        3.5      3.3      2.7      1.2      3.8
Y        2.5      3.6      1.2      2.6      2.9
for_each(cnt_itr(0), cnt_itr(0)+n, compute_bins(X, Y, B))
B         13       18        7       11       13
sort_by_key(B.begin(), B.end(), zip(X, Y, I))
B          7       11       13       13       18
I          2        3        0        4        1
for_each(cnt_itr(0), cnt_itr(0)+n, compute_neighbor_range_start(B, R))
R   1  6 11  5 10 15  7 12 17  7 12 17 12 17 22
lower_bound(B.begin(), B.end(), R.begin(), R.end(), N1.begin())
N1  0  0  1  0  1  4  0  2  4  0  2  4  2  4  5
for_each(cnt_itr(0), cnt_itr(0)+n, compute_neighbor_range_end(R))
R   3  8 13  7 12 17  9 14 19  9 14 19 14 19 24
upper_bound(B.begin(), B.end(), R.begin(), R.end(), N2.begin())
N2  0  1  4  1  2  4  1  4  5  1  4  5  4  5  5
```

Center Finding Algorithms

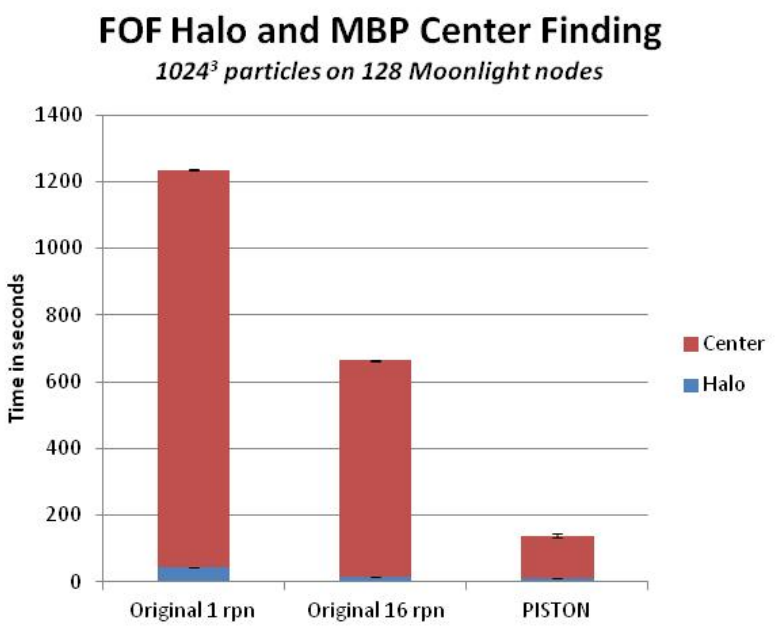
- Most Connected Particle can be found by counting the number of friends within the 27 bins. A **segmented** max scan can then determine the particular particle with most friends within each halo.
- Most Bound Particle can be found by first calculating its potential within each halo. Followed up by a similar approach as MCP to find the particle with minimum potential.

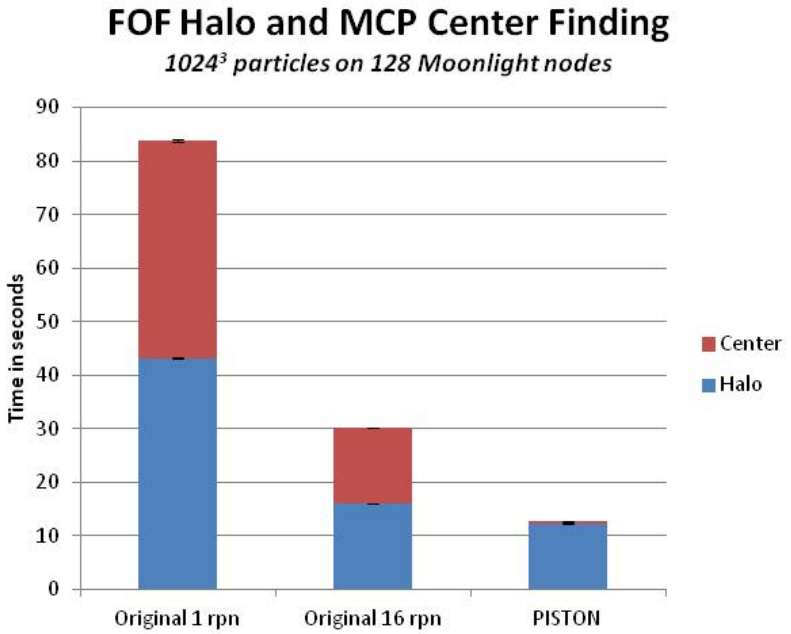
MBP Center Finding




```
I       0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
X       1   1   2   1   3   1   1   3   2   3   4   3   5   5   4   5   6
Y       1   2   6   3   5   4   5   6   3   3   3   7   2   3   6   4   3
D       0   0   2   0   2   0   0   2   0   0   0   2   0   0   2   0   0
sort_by_key(D.begin(), D.end(), zip(I, X, Y)
D       0   0   0   0   0   0   0   0   0   0   0   0   2   2   2   2   2
I       0   1   3   5   6   8   9  10  12  13  15  16   2   4   7  11  14
inclusive_scan_by_key(D.begin(), D.end(), cnt_itr(0), H1.begin(), min)
H1      0   0   0   0   0   0   0   0   0   0   0   0  12  12  12  12  12
sequence(H2.begin(), H2.end(), 0)
H2      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
inclusive_scan_by_key(D.rbegin(), D.rend(), H2.rbegin(), H2.rbegin(), max)
H2     11  11  11  11  11  11  11  11  11  11  11  11  16  16  16  16  16
for_each(cnt_itr(0), cnt_itr(0)+n, compute_potential(X, Y, H1, H2, P))
P    -4.01 -5.22 -5.77 -5.22 -4.01 -6.02 -5.83 -5.93 -4.84 -6.01 -4.83 -4.46 -2.91 -2.91 -4.00 -2.91 -2.91
inclusive_scan_by_key(D.begin(), D.end(), P.begin(), Pmin.begin(), min)
Pmin -4.01 -5.22 -5.77 -5.77 -5.77 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -2.91 -2.91 -4.00 -4.00 -4.00
inclusive_scan_by_key(D.rbegin(), D.rend(), Pmin.rbegin(), Pmin.rbegin(), min)
Pmin -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -6.02 -4.00 -4.00 -4.00 -4.00 -4.00
transform(cnt_itr(0), cnt_itr(0)+n, C.begin(), equals_min_pot(I, P, Pmin)
C       0   0   0   0   0   8   0   0   0   0   0   0   0   0   7   0   0
inclusive_scan_by_key(D.begin(), D.end(), C.begin(), C.begin(), max)
C       0   0   0   0   0   8   8   8   8   8   8   8   0   0   7   7   7
inclusive_scan_by_key(D.rbegin(), D.rend(), C.rbegin(), C.rbegin(), max)
C       8   8   8   8   8   8   8   8   8   8   8   8   7   7   7   7   7
sort_by_key(I.begin(), I.end(), zip(D, C))
I       0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
C       8   8   7   8   7   8   8   7   8   8   8   7   8   8   7   8   8
```

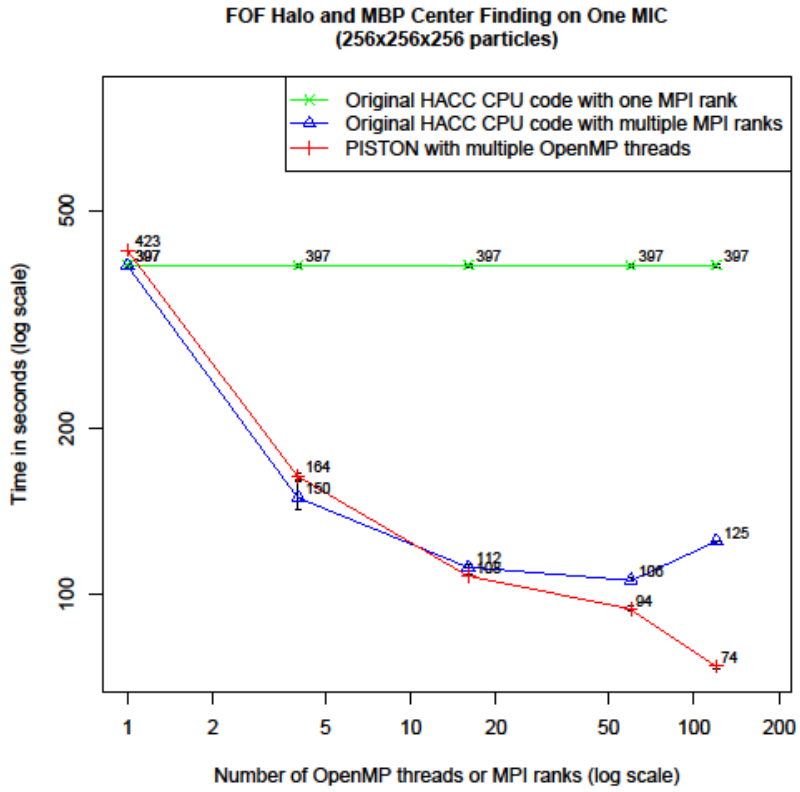
Results on Moonlight





FOF + MBP: ~4.9x faster than original with 16 rpn
FOF + MCP: ~2.5x faster than original with 16 rpn

Result on Stampede with Xeon Phi



Max runtime in for center finding on Mira, Cooley and Theta (in seconds)

Stage	Mira	Cooley (GPU)	Theta Thrust (Multi-Core KNL)	Theta (Single Core)
Center Finder	1794.900	131.45	73.815	461.390

- Performance study conducted by Thomas D. Uram, Chris Sewell and Adrian Pope
- Cooley is more 10x faster on Cooley's GPU (K80) than Mira' CPU
- More than 1.7x faster on Theta's KNL than Cooley's GPU