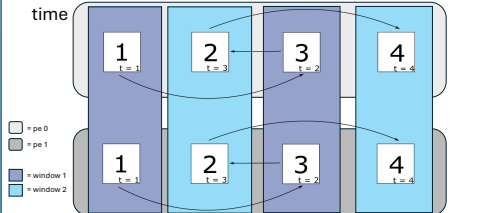


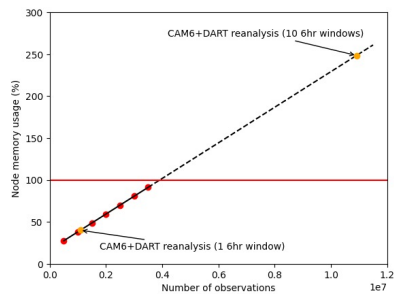


1. Background

- The Data Assimilation Research Testbed (DART) combines real-world observations and model forecasts to improve our understanding of the Earth system
- Current parallelization efforts have so far split the state variables and forward operator calculations across multiple processors; however, the observation sequence has not been parallelized yet, resulting in time-consuming reads of the sequence and excessive memory consumption resulting from observation duplication.
- This project's goal is to provide a distribution method which efficiently load balances the observation sequence.

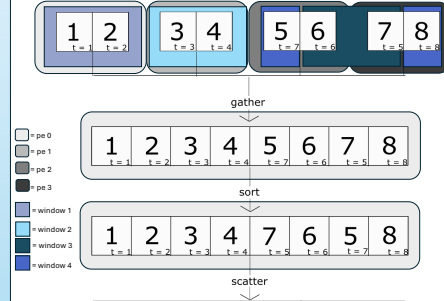
2. Current Method

- Each process reads in a copy of the observation sequence
 - Processes reference their own copies of the observation sequence when assimilating state and calculating forward operators
 - Observations are ordered in a linked list; order in which linked list is traversed is from earliest to latest observation time
- 
- Current method provides immediate access to every observation in observation sequence across all processes
 - Number of observations which can be stored in single set is limited to the available memory on a single process (see below)
- o Ex. Each node on Derecho has 256 GB, and each node has 128 processes. $256 / 128 = 2$ GB per process



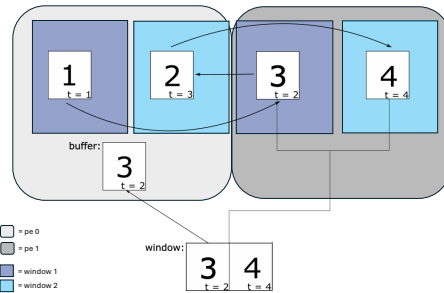
3. Alternative Methods

Gather-sort-scatter



- Each process reads a subset of the observation sequence (binary file traversed with fseek)
- All observations are retrieved by the first process
- Observations are sorted in time order
 - Each process receives observations in a specific time window; traversing time window does not require access to other processes' observations
- New observation sequence split into subsets and scattered to every process

One-sided communication

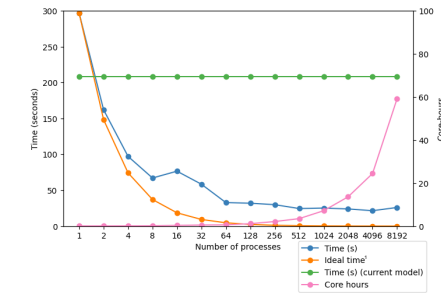


- Each process reads a subset of the observation sequence
- Each process opens a memory window for the observations they own
- Processes use one-sided operations to retrieve observations not available in their local memory from other processes' memory windows

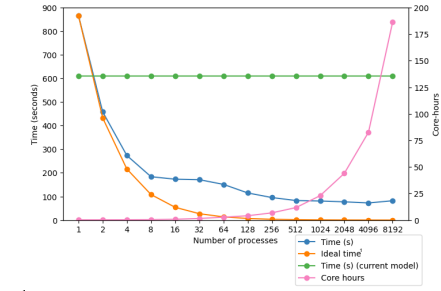
4. Performance Results

Gather-sort-scatter

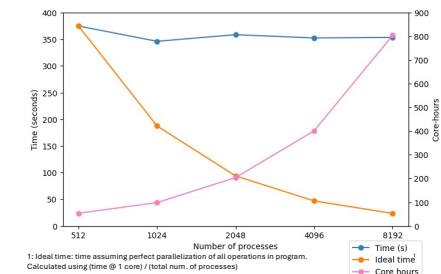
Scaling: 64,950,921 observations



Scaling: 208,901,231 observations



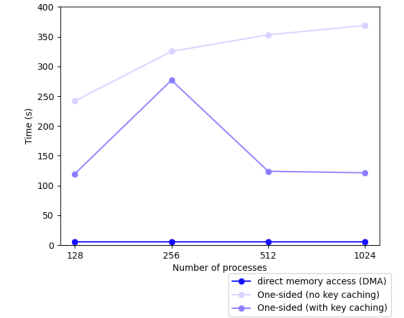
Scaling: 626,703,695 observations



- Time to read observation sequence from file reduces as more processes read from file
- Scaling is constrained by sequential (single process) sort of observations by time order
 - One process has every observation during "sort" step; sequence size constrained by per-node memory (each process can only directly access memory on its node)
 - Parallel sorting algorithms (ex. Sample-sort) may alleviate this constraint

One-sided communication

Retrieval speed: 64,950,921 observations



Peak per-process memory usage

# of obs:	64,950,921	208,901,231	626,703,695
Gather	7.46 GB	23.6 GB	70.5 GB
One-sided	205 MB	279 MB	462 MB

- Peak memory per-process is reduced from gather model, since observations are not collected on single process and sorted
 - Observations can be read in time-sorted order by traversing the linked list using one-sided communication
- Retrieval of observations worsens as number of processes increases
- One-sided retrieval time reduced when keys of observations to be retrieved are already known by process; multiple gets can be performed simultaneously

5. Conclusion

- We developed and evaluated two methods which can be used to distribute observations across processes when observations are split
- Time to read observations is reduced by increasing number of processes reading observation sequence
- Reducing observation sequence duplication allows more observations to be stored
- Both models have drawbacks; ideal distribution method would combine methods from both distribution types