Improving the Speed and Scalability of the Data Assimilation Research Testbed

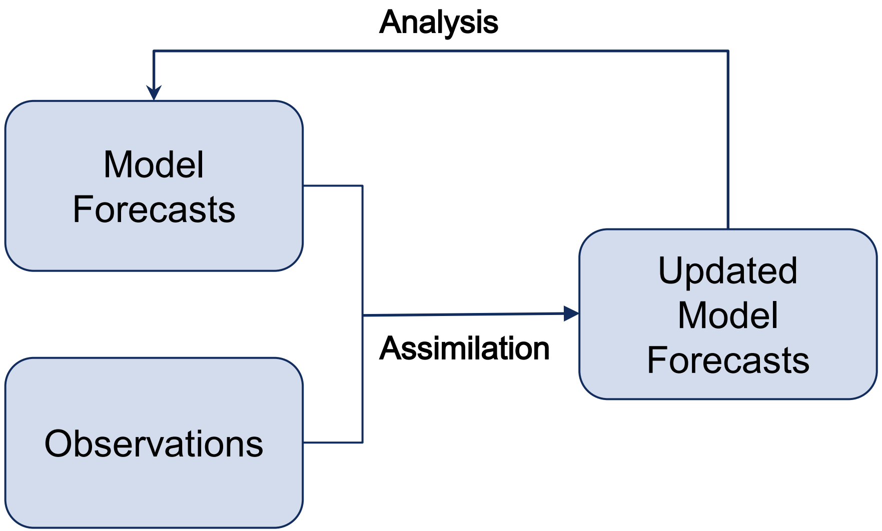
Jiachen (Ed) Liu
Drexel University
Mentors: Helen Kershaw, Jeffrey Anderson

NCAR
UCAR

Data assimilation is a process to combine model outputs and observations to improve model forecasts

Example: a temperature forecast

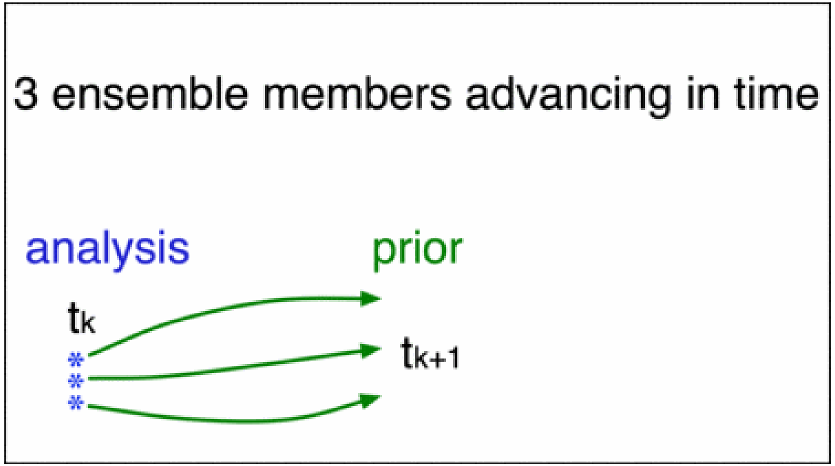
- Model forecast: a three dimensional atmospheric model which computes the temperature
- Observations: observed temperature with a thermometer at a given time
- Assimilation: Generate the statistically optimal value based on the forecast and the observation



The Data Assimilation Research Testbed (DART) helps researchers perform ensemble data assimilation

Ensemble data assimilation process can capture the uncertainties inherent to model forecasts and observations.

DART provides a platform for flexible and powerful ways to perform ensemble data assimilation with different models.



Ensemble DA with DART

Improving speed and scalability of DART is important for the future

- Although there are modules to utilize parallel computing resources to run DART, it is still computationally expensive.

- The focus of this work is to:
  1. Identify the computational barriers in DART with code profiling tools
  2. Improve the speed and scalability of DART through algorithmic changes

The speed and scalability of DART with various models are important for the future

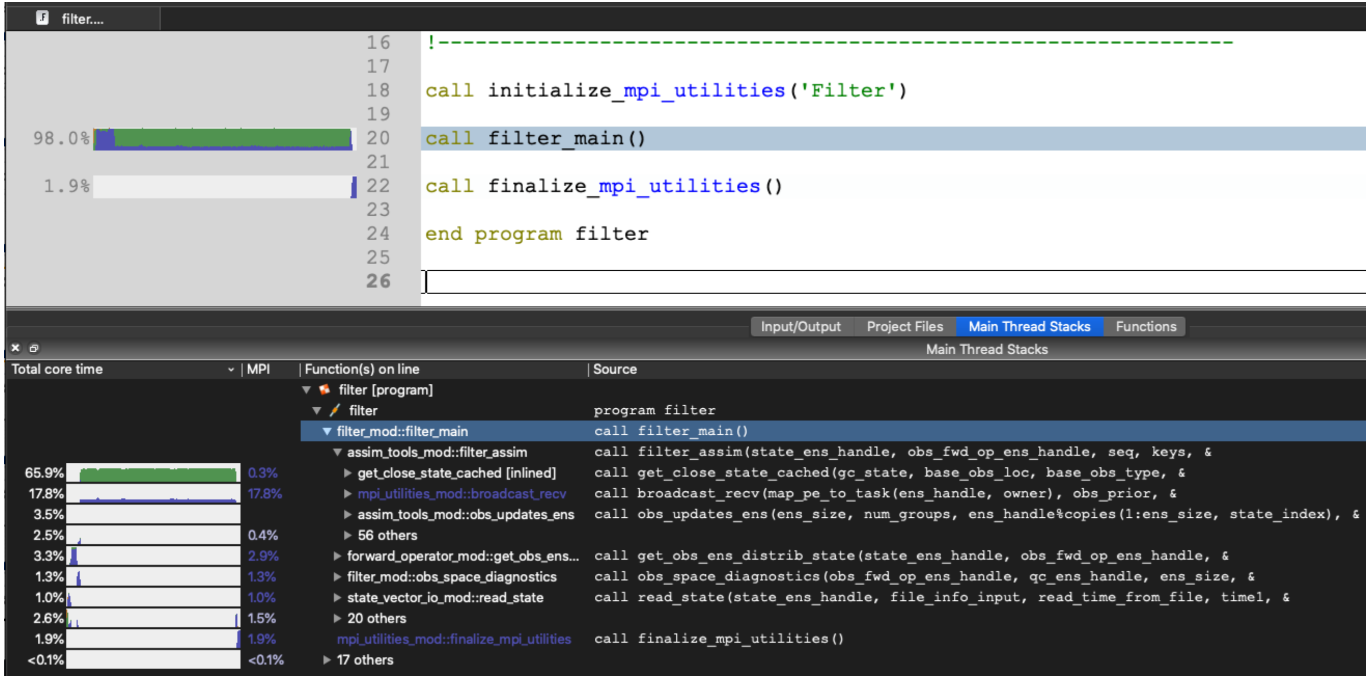
- Although there are modules to utilize parallel computing resources to run DART, it is still computationally expensive.

- The focus of this work is to:
  1. **Identify the computational barriers in DART with code profiling tools**
  2. Improve the speed and scalability of DART through algorithmic changes

The identification of computational barriers in DART is essential for the future

- Example code profiling result with arm -forge MAP tool of DART

arm FORGE



Initial profiling results of DART show that it generally scale well with increased computational resources

- Finite Volume Community Atmosphere Model (CAM-FV) test case

# of nodes	# of processors	Runtime from MAP [s]	filter_mod, compute (%)	filter_mod, mpi (%)	mpi_utilities (%)
2	36	2401.83	70	26.2	3.8
4	36	1071.377	44.5	47.1	8.4
8	36	658.034	24.5	61.8	13.6
10	36	339.992	39.2	34.1	26.6
20	36	285.397	25.8	41.5	32.7
10	4	1184.76	85.2	7.3	7.5
10	16	446.941	60.5	19.4	20

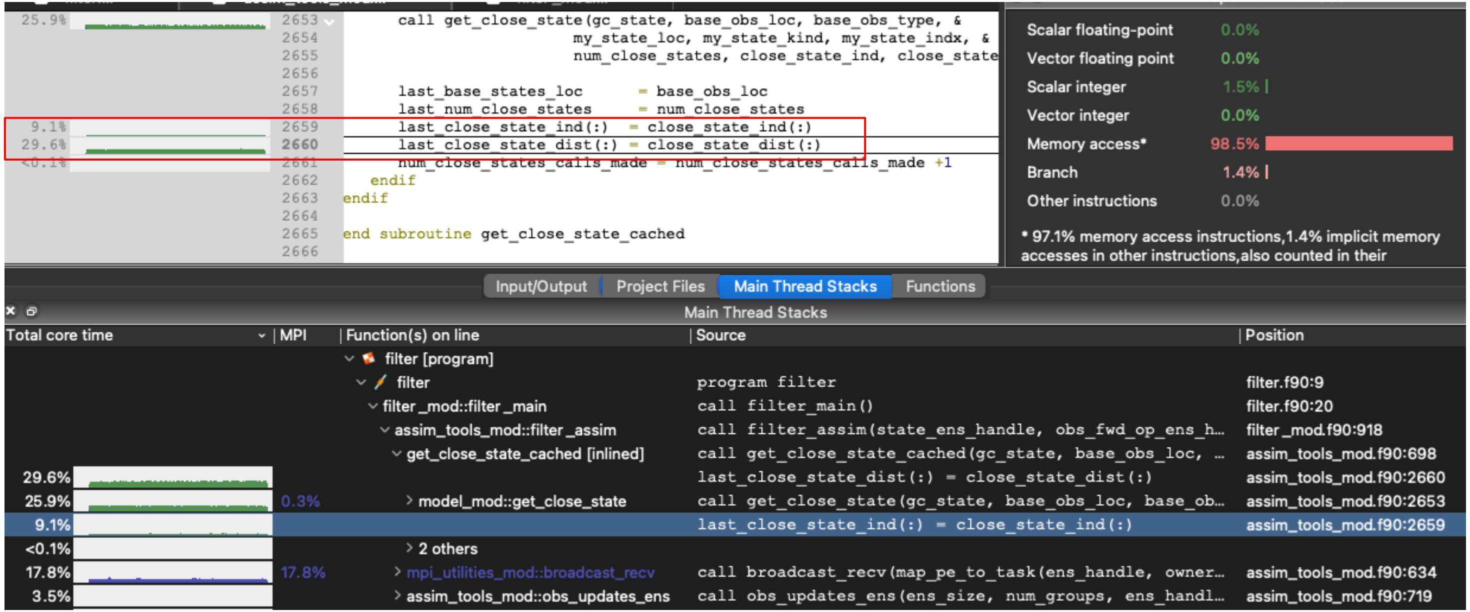
Initial profiling results of DART show that it generally scale well with increased computational resources

- In general, increased number of nodes and/or number of processors per node decrease the total runtime.
- However, as the number of nodes is relatively high, the effect of increasing number of nodes on total runtime reduction decreases.

# of nodes	# of processors	Runtime from MAP [s]	filter_mod, compute (%)	filter_mod, mpi (%)	mpi_utilities (%)
2	36	2401.83	70	26.2	3.8
4	36	1071.377	44.5	47.1	8.4
8	36	658.034	24.5	61.8	13.6
10	36	339.992	39.2	34.1	26.6
20	36	285.397	25.8	41.5	32.7
10	4	1184.76	85.2	7.3	7.5
10	16	446.941	60.5	19.4	20

Additional profiling results revealed redundant caching in DART consumes significant computational resources

- Atmospheric component of the Model for Prediction Across Scales (MPAS - ATM)



Additional profiling results revealed redundant caching in DART consumes significant computational resources

- The purpose of this subroutine is to cache the location and indices of the previous observation so that we can reduce computation time.

- However, these two lines of copying actually consume almost **40% of the total runtime of DART** for this case!

- It turns out that we don't need these two lines of code to perform the caching. These are redundant copying of very large arrays.

Additional profiling results revealed redundant caching in DART consumes significant computational resources

- Initial testing with this test case shows that without calling the subroutines, the computation time <u>reduced from 260 seconds to 64 seconds</u>.

- The problem is now resolved with a pull request at https://github.com/NCAR/DART/pull/368

The speed and scalability of DART with various models are important for the future

- Although there are modules to utilize parallel computing resources to run DART, it is still computationally expensive.

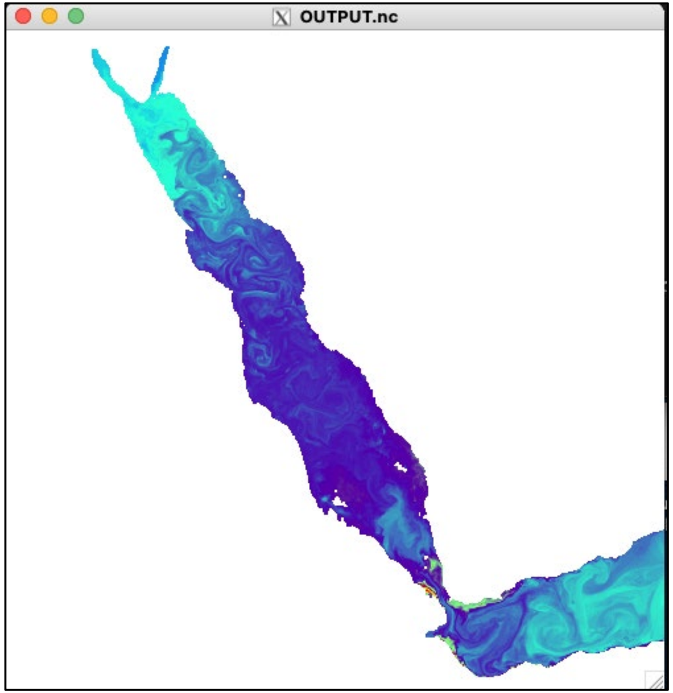
- The focus of this work is to:
  1. Identify the computational barriers in DART with code profiling tools
  2. **Improve the speed and scalability of DART through algorithmic changes**

A high-resolution assimilation run of MIT General Circulation Model for the red sea is a computational problem for DART

- The MIT General Circulation Model for the ocean (MITgcm-ocean) is a numerical model that can compute parameters related to the ocean.

- This specific run is on a 2000x2000x50 (latitude, longitude, depth) grid.

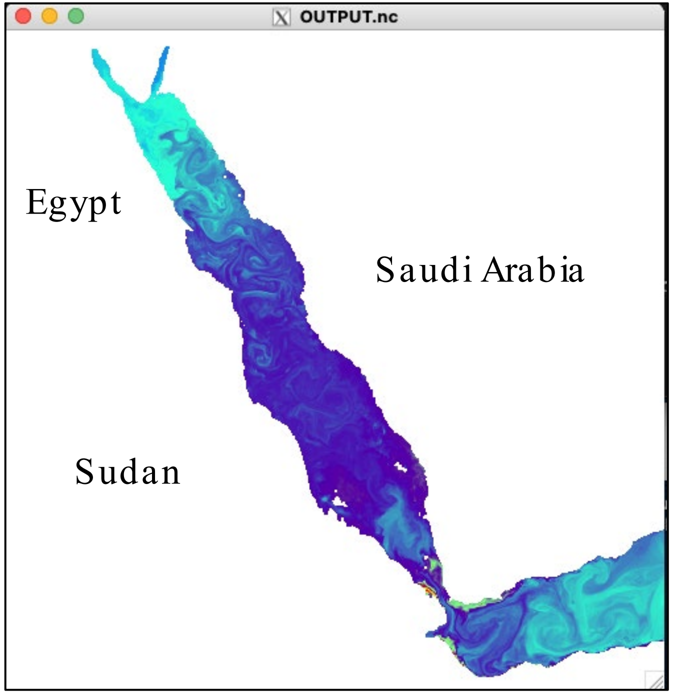
- DART cannot be run on Cheyenne or on the extreme memory nodes (4 TB) at Pittsburgh Supercomputing Center for this specific case.



Sample output from MITgcm -ocean for the red sea

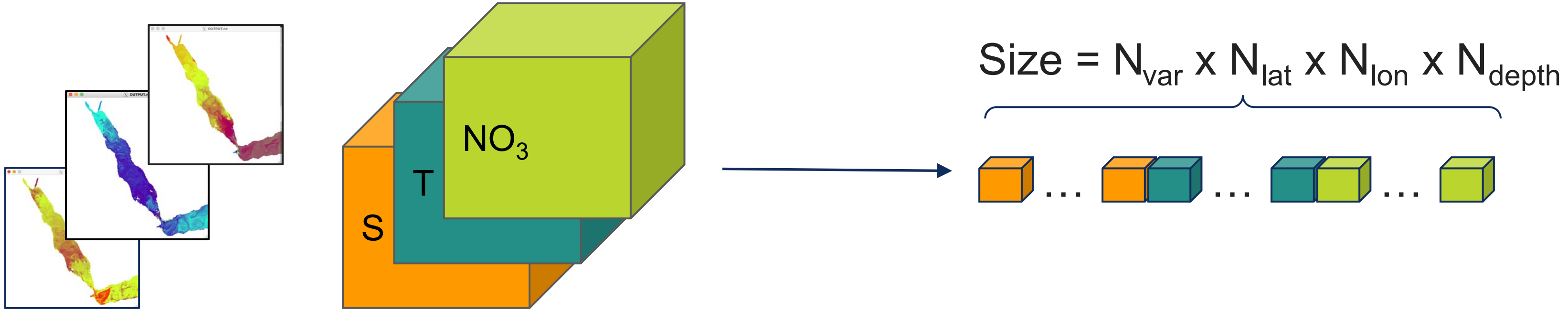
A high-resolution assimilation run of MIT General Circulation Model for the red sea is a computational problem for DART

- Memory overflow is likely the problem.
- The grid has land which is not used in the data assimilation process.
- In the state file, these values are usually *fill values*.
- Analysis shows **92%** of the grid are fill values.



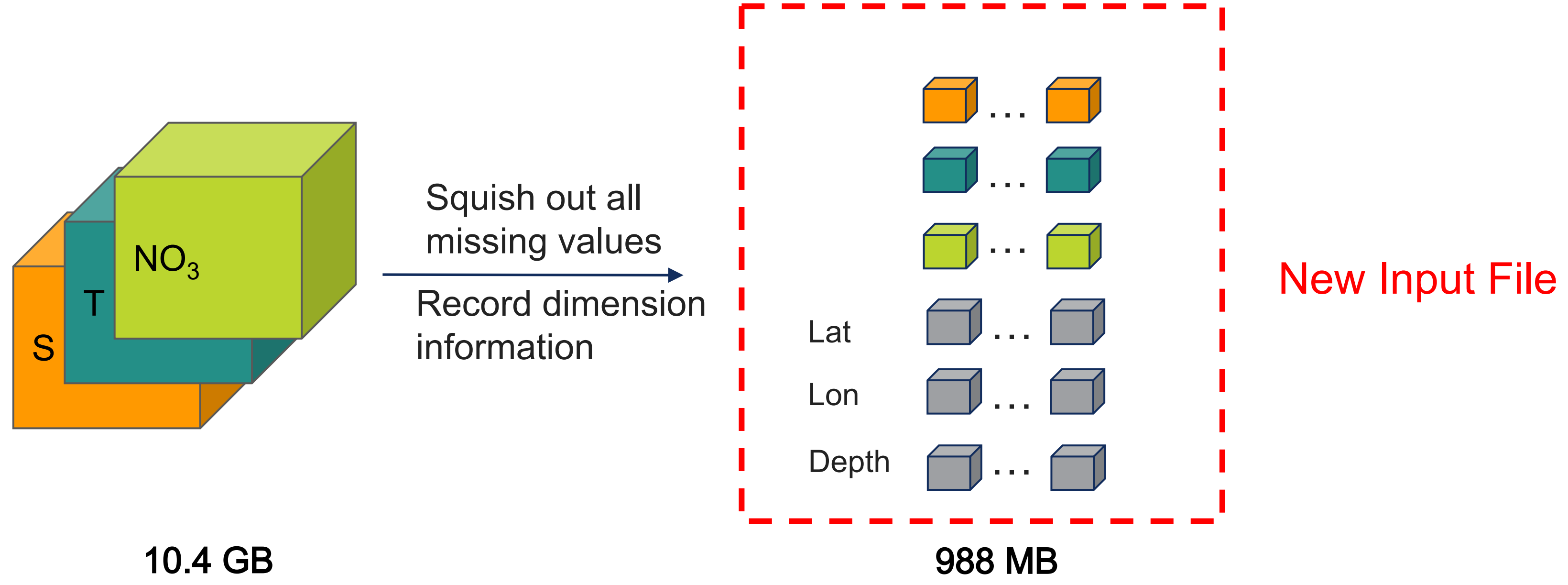
Sample output from MITgcm-ocean for the red sea

DART handles the state information by generating a 1-D DART vector



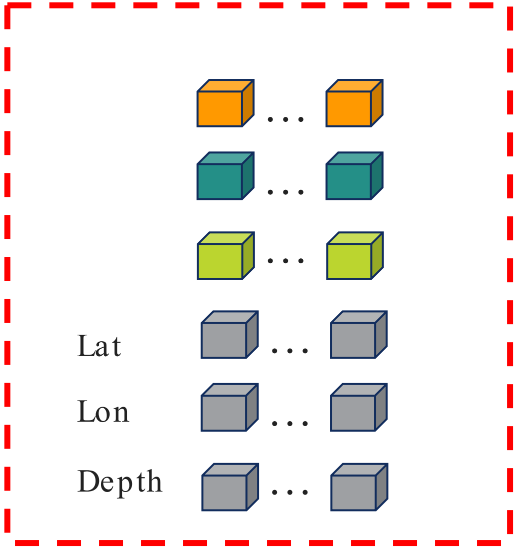
- The state might have several variables (salinity, temperature, nitrate concentration, etc.)
- DART reduces everything into a 1-D DART vector and performs data assimilation.
- **The missing values (land cells in an ocean model) stay in the vector.**

The squished state approach can significantly reduce the size of the state vector

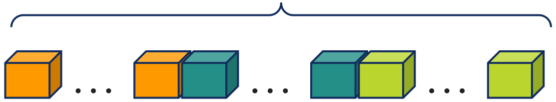


- The new input file does not have missing values at all.
- Additional dimension information is required because we squished the grid.

The squished state approach can significantly reduce the size of the state vector



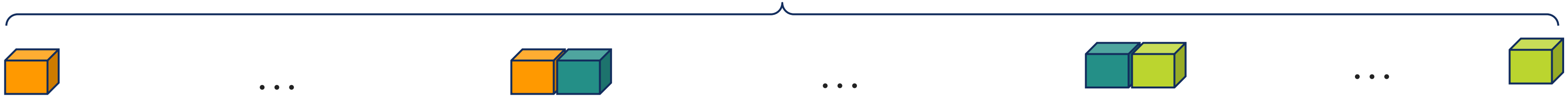
$$\text{Size} = \%\text{useful} \times N_{var} \times N_{lat} \times N_{lon} \times N_{depth}$$



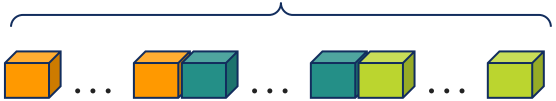
- If %useful is small, the size of the state vector can be reduced significantly, so the assimilation might be able to run

The squished state approach can significantly reduce the size of the state vector

$$\text{Size} = N_{var} \times N_{lat} \times N_{lon} \times N_{depth}$$



$$\text{Size} = \textcolor{red}{\%\text{useful}} \times N_{var} \times N_{lat} \times N_{lon} \times N_{depth}$$



- The DART model size (number of variables) <u>**decreased from ~2.63e9 to ~2.0e8.**</u>

The squished state approach improves the speed and scalability of DART*

	Medium Case (500x500x50)	Large Case (2000x2000x50)
Original	361s	N/A
Squished State	150s	1500s

- The computation time for the medium case decreased from 361 seconds to 150 seconds.
- The large case now runs properly.
- The squishing process can be done fairly easily without significant additional computational resources.

Future Work

- Make the squishing process "online" with DART
  - The users only need to specify if they want to use the squished state method.
- Write subroutines which reformulate the squished DART array back into its original form.

Acknowledgements

I would like to thank my mentors Helen and Jeff for their advice throughout the summer. I also want to thank the DAReS team, the CODE team, and CISL help desk for their support.