Parallelizing the Climate Data Management System, version 3 (CDMS)

Denis Nadeau[1], Dean Williams[1], Jeff Painter[1], Charles Doutriaux[1]
[1] Lawrence Livermore National Laboratory

What is CDMS?

Implemented as a component of the Climate Data Analysis Tools (CDAT), the Climate Data Management System (CDMS) is used to automatically locate and extract metadata (i.e., variables, dimensions, grids, etc.) from the multi-model collection of model runs and analysis files. CDMS is defined as an object-oriented data management system, specialized for organizing multidimensional, gridded data used in climate analysis and simulation. As a fully Climate and Forecast (CF) compliant data access tool, CDAT (via CDMS) allows users to seamlessly read data from multiple sources for intercomparison studies. In addition to reading in netCDF CF compliant data, CDAT can also write data in this format. In addition to netCDF, CDAT can also read in HDF, GRIB, ASCII, binary, and other popular climate data file formats.

CDMS Overview

1. CDMS Python Application Programming Interface (API)
   - CDMS itself is implemented in a mixture of C and Python

Type	Description
Array	Numeric masked multidimensional data array. All elements of the array are of the same type.
Comptime	Absolute time value, a time with representation (year, month, day, hour, minute, second).
Dictionary	A collection of objects, indexed by key. All dictionaries in CDMS are indexed by strings, e.g.: axes['time'].
Float	Floating-point value.
Integer	Integer value.
List	An ordered sequence of objects, which need not be of the same type. New members can be inserted or appended. Lists are denoted with square brackets, e.g., [1, 2.0, 'x', 'y'].
None	No value returned.
Reltime	Relative time value, a time with representation (value, units since basetime). Defined in the cdtime module. cf. comptime.
Tuple	An ordered sequence of objects. Unlike lists, tuples elements cannot be inserted or appended. Tuples are denoted with parentheses, e.g., (1, 2.0, 'x', 'y').

2. "cdtime" module
   - Converting a units string, of the form 'units since basetime', a floating-point value based on the common calendars used in climate simulation. Basic arithmetic and comparison operators are also available.

3. Regridding Data
   - provides several methods for interpolating gridded data
   - from one rectangular, lat-lon grid to another (CDMS regridder)
   - between any two lat-lon grids (ESMF and SCRIP regridders)
   - from one set of pressure levels to another
   - from one vertical (lat/level) cross-section to another vertical cross-section.

4. Plotting CDMS data in Python
   - Data read via the CDMS Python interface can be plotted via a number of plotting packages which are included in the Ultrascale Visualization Climate Data Analysis Tools (UV-CDAT)

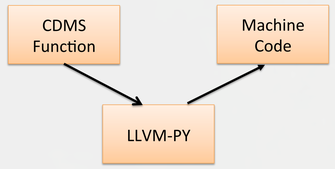
5. Climate Data Markup Language (CDML)
   - The Climate Data Markup Language (CDML) is the markup language used to represent metadata in CDMS. CDML is based on the W3C XML standard (http://www.w3.org).

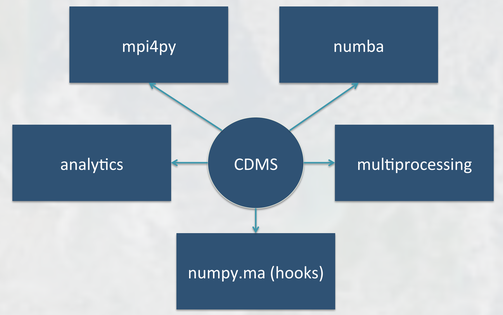
ABSTRACT

The Climate Data Management System is an object-oriented data management system, specialized for organizing multidimensional, gridded data used in climate analyses for data observation and simulation. The basic unit of computation in CDMS is the variable, which consist of a multidimensional array that represents climate information in four dimensions corresponding to: time, pressure levels, latitudes, and longitudes. As models become more precise in their computation, the volume of data generated becomes bigger and difficult to handle due to the limit of computational resources. Models today produce data at a time frequency as much as hourly and spatial resolution as fine as in satellite observations. As the amount of data grows, so does the time needed for scientists to analyze the data and retrieve useful information. The process threatens to become unmanageable. We can ease the burden of working with big data sets by parallelizing CDMS. Multiple approaches are possible. The most obvious one is embarrassingly parallel or pleasingly parallel programming where each computer node processes one file at a time. A more challenging approach is to send a piece of the data to each node for computation and each node will save the results at its right place in a file as a slab of data. This is possible with Hierarchical Data Format 5 (HDF5) using the Message Passing Interface (MPI). A final approach would be the use of Open Multi-Processing API (OpenMP) where a master thread is split in multiple threads for different sections of the main code. Each method has its advantages and disadvantages. This poster brings to light each benefit of these methods and seeks to find an optimal solution to compute climate data analyses in an efficient fashion using one or a mixtures of these parallelized methods.

CDMS Expansion

Numba gives the possibility to speed up part of CDMS with high performance functions written directly in Python. Numba works by generating optimized machine code using the LLVM (formerly Low Level Virtual Machine) compiler infrastructure at import time, runtime, or statically. Numba supports compilation of Python to run on either CPU or GPU hardware, and is designed to integrate with the Python scientific software stack.





Pool object offers a convenient mean of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism).

Use of multprocessing (pool of workers)

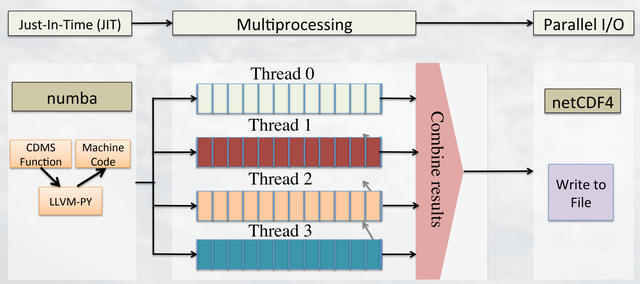
```
for i in arange(N):
        serial[i] = test_prime(i)

Replace with:
    test_prime.parallel = parallel_function(test_prime)
    parallel_result = test_prime.parallel(arange(N))
```

Parallelism is being introduced to CDMS as data warehouse becomes larger and requires new techniques for managing good query performance along large data sets. Parallel execution does provide the greatest performance executions, but new modules version such as Numpy have evolved to better performance and CDMS code can take advantage of them. The MV2 module can be replaced by Numpy using the __array_finalize__ hook available.

In addition, CDMS can take advantage of all the grid and axis capability and expand to create a package for data analyses.

Workflow

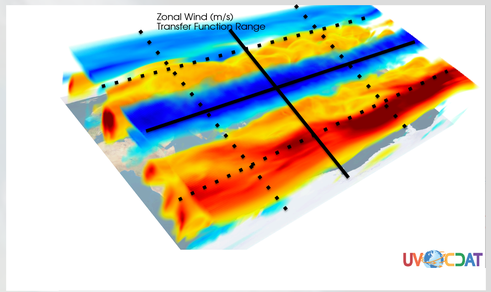


Combination of different parallelism mechanism can be used to increase performance of CDMS. In this workflow, Numba's "Just-In_time"(JIT) capability is used to accelerate mathematical computation.

The middle section found in figure to the left shows a multiprocessing pool of workers. Tasks are dispatched in an embarrassingly parallel manner either to perform computation, such as regridding, or to process different sections of a big array. This mutiprocessing code will take advantage of all cores available on the computer.

Finally, CDMS can also take advantage of netCDF parallel I/O to write in parallel. It is now possible to combine all these methods and increase software performance.

UV-CDAT



CDMS using MPI is implemented as part of the Ultrascale Visualization Climate Data Analysis Tools (UV-CDAT). CDMS can now utilize a parallel I/O interface which allows for both single and multi-threaded I/O, as well as "quilted" I/O. CDMS can also pass flags to the netCDF4 library to select chunking and/or deflation options if desired by the user. All parallel I/O commands called by UV-CDATA are conducted by CDMS using the python module mpi4py, which is linked to openMPI. CDMS calls a python extension to interface with the "C" parallel netCDF4 programming interface. Python is an excellent language for developing parallel code. It is a great language for prototyping and for small scripts. The python module "mpi4py" offers most functionality of C or Fortran implementations of MPI. The main difference between "mpi4py" and MPI in C or Fortran is that mpi4py is object oriented. As well, using MPI functionality, CDMS can parallelize by dividing the data and sending work to different processors. The figure above shows how UV-CDAT can divide the data and send each tiles to different processors to perform some computation. The number of nodes is decided by the user depending on the resources available in his or her cluster.

Future work

Benchmarking:
Different parallelism architecture will give different computation speed. It is important to select the algorithm that will make the most of compute resources. Choosing an embarrassingly parallel approach can be appropriate when many files need to be processed.

What is the best approach or architecture for parallelism in remote-sensing? More work needs to be done to answer this question and a very good benchmark application needs to be used. Most of the time being lost is in disk I/O and a developer needs to be very careful when using profilers so that the speed returned by the benchmark is actual computation and not I/O latency.

Cython:
Cython is already compiled with Scipy which is now part of UV-CDAT. A good use of Cython could speed up computations.

Some CDMS code is being updated to make better use of latest Python modules such as Numpy. A lot of new interface is becoming available and taking advantage of them can also greatly speed up computations.

For Additional Information

Nadeau1@LLNL.gov
Williams13@LLNL.gov

This work is supported by the ESGF consortium: DOE, NASA, NOAA, IS-ENES, NCI.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DEAC52-07NA27344

2015 Conference