CDMS Overview

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Numeric modeled multidimensional data array. All elements of the array are of the same type.</td>
</tr>
<tr>
<td>Complete</td>
<td>Allocated time value, with write-in representation (year, month, day, hour, minute, second).</td>
</tr>
<tr>
<td>Dictionary</td>
<td>A collection of objects, indexed by key. All dictionaries in CDMS are extended strings, e.g., [key=value].</td>
</tr>
<tr>
<td>Float</td>
<td>Floating-point value.</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer value.</td>
</tr>
<tr>
<td>List</td>
<td>An ordered sequence of objects, which need not be of the same type. New members can be inserted or appended. Elements are deivered with square brackets, e.g., [2, 3, 5, 'v'].</td>
</tr>
<tr>
<td>Month</td>
<td>No calendar reference</td>
</tr>
<tr>
<td>Numba</td>
<td>A float time value, with time representation (years since baseline). Defined in the cdms module. A complete time is not defined.</td>
</tr>
<tr>
<td>Tuple</td>
<td>An ordered sequence of elements. Tuple elements can be inserted or appended. Elements are deivered with parenthesis, e.g., (2, 3, 5, 'v').</td>
</tr>
</tbody>
</table>

2. “code” module
   - Converting a units string, of the form “units since baseline”, to a floating-point value based on the common calendars used in climate simulation. Basic arithmetic and comparison operators are also available.
   - 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 regridder)
     - From one set of pressure levels to another
     - From one vertical (latitude) cross-section to another vertical cross-section.
   - 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).

CDMS Expansion

- 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, latitude, and longitude. As models become more precise in their computation, the volume of data generated becomes bigger and difficult to handle due to the limit of computer 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 complex approach is to processorize the data to each node for computation and each node will save the result 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 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 mixture of these parallelized methods.

Numba gives the possibility to speed up part of CDMS with high performance functions written entirely 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.

- Use of multiprocessing (pool of workers)
  - Pool offers a convenient mean of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism).
  - 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 code module can be replaced by Numba 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 mechanisms 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 regidding, or to process different sections of a big array. This multiprocessing 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.

Future work

- 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.

For Additional Information

Nadeau1@LLNL.gov
Williams1@LLNL.gov