



Parallel Analysis Tools and New Visualization Techniques  
for Ultra-Large Climate Data Sets (ParVis)

PROGRESS REPORT: 2010-2011

## Summary

ParVis is a new project funded under LAB 10-05: “Earth System Modeling: Advanced Scientific Visualization of Ultra-Large Climate Data Sets”. Argonne is the lead lab with partners at PNNL, SNL, NCAR and UC-Davis.

This report covers progress from when work began (as funding arrived at the various institutions July - September, 2010) through Sept. 30, 2011

A primary focus of ParVis is introducing parallelism to climate model analysis to greatly reduce the time-to-visualization for ultra-large climate data sets. For this report, it is convenient to summarize the work in the first year as two tracks with different time horizons: one track is to provide immediate help to climate scientists already struggling to apply their analysis to existing large data sets. The other track focused on building a new data-parallel library and tool for climate analysis and visualization that will give the field a platform for performing analysis and visualization on ultra-large datasets for the foreseeable future.

***We completed all of our first-year milestones and made our first-year deliverable: a task parallel (using Swift) version of the latest Atmospheric Model Working Group diagnostic package. This version was made available to the community for free via the ParVis website.***

## Progress on short-term improvements

### Swift-based climate model diagnostics

Swift is a system for the rapid and reliable specification, execution, and management of large-scale science and engineering workflows. It supports applications that execute many tasks coupled by disk-resident datasets - as is common, for example, when analyzing large quantities of data or performing parameter studies or ensemble simulations. The diagnostic plots made by c-shell scripts developed by CESM working groups can straightforwardly be recast as many-task parallel applications. Plots from these scripts are used extensively by scientists evaluating climate model integrations.

Prior to the start of ParVis, John Dennis of NCAR converted version 4.1.2 (circa May, 2010) of the Atmospheric Model Working Group diagnostic package to Swift and used it internally for high-resolution diagnostics. We took over this work. In our first year, we updated the swift version to AMWG diagnostic package version 5.1 (May, 2011) and made it available to the community for testing. Figure 1 below shows that Swift can increase performance by an average of 3x. Swift uses either multiple cores in a workstation or multiple cores/processors/nodes on a cluster to execute the parallel workflow.

Some modifications were made to Swift to support ParVis needs. Provisions were made to enable the application script to specify job-specific parameters and pass them through to the underlying resource manager. This enables the AMWG scripts to request large-memory compute nodes for specific application invocations that require them. The ability of the user script to execute MPI applications under dynamically provisioned Swift compute nodes was added to facilitate the integration of Pagoda into the AMWG scripts (see below). Automatic deletion of the un-named temporary disk files used to pass information between intermediate stages of the AMWG diagnostic package was implemented. This feature now deletes such temporary files as soon as they are no longer needed within a script, even where complex dependency patterns exist. This reduces the overall disk storage required by the package, and makes it more feasible to execute within disk-space-constrained environments. Finally, several infrequently occurring race conditions in Swift were uncovered by the execution of the AMWG package, and have been fixed.

## Original vs. Swift Timings for Various Datasets

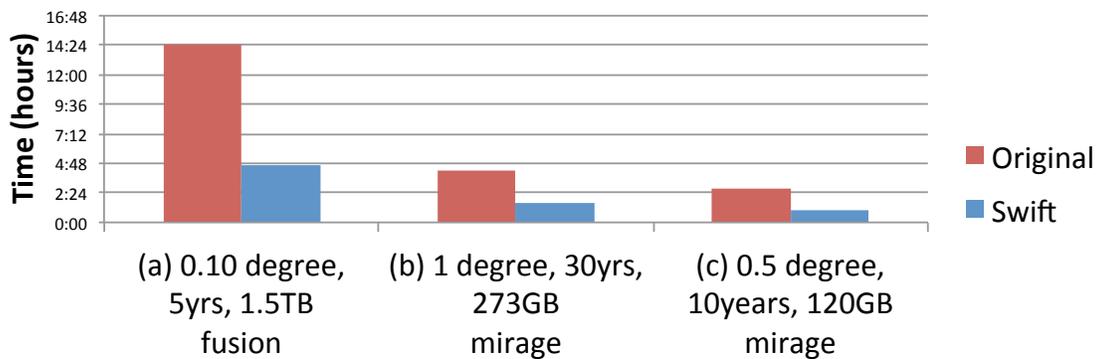


Figure 1 Swift speed-ups for the AMWG diagnostics applied to various data sets.

The diagnostic scripts use a combination of NCO utilities (ncra, etc.) to perform data reduction and NCL to make additional calculations and plots. Pagoda, developed as part of the Colorado State University Global Cloud Resolving Model project, provides parallel versions (using Global Arrays) of the NCO utilities. We explored the performance of the equivalent of ncra (called pgra) on large data sets and had encouraging results (Figure 2). Thanks to our testing, Pagoda developers found that permuting the array order leads to additional scalability. We will explore replacing NCO with Pagoda in the Swift versions of the diagnostic scripts for additional speedup.

We have also developed an all-NCL version of the AMWG diagnostic package. This will be used to compare the performance of Swift-based scripts that use our new ParNCL application with those that use NCO or Pagoda for the data reduction.

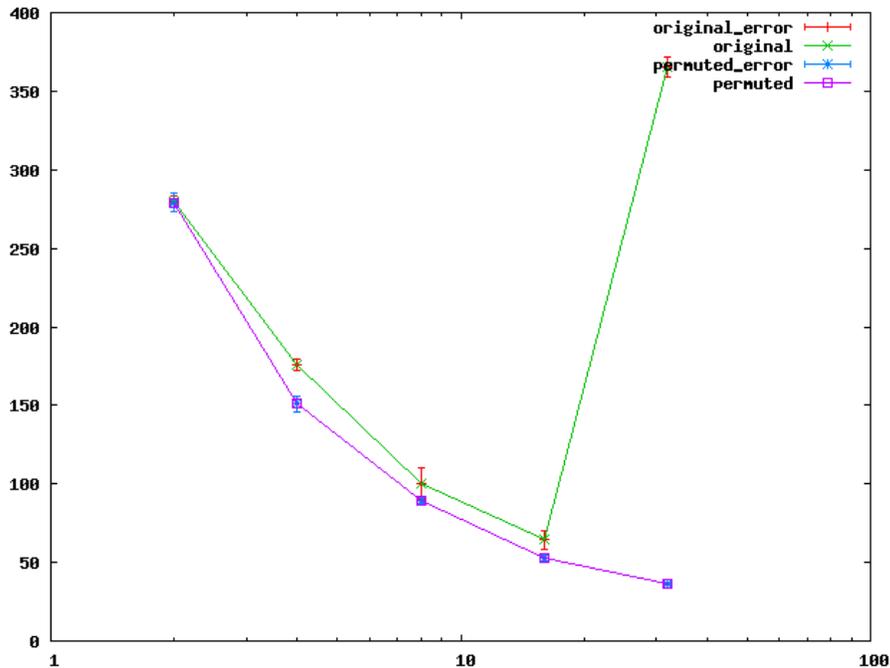


Figure 2 Time (seconds) vs. processor count for processing 42 GB of data with pgra for different array orderings.

We also began work to convert the Ocean Model Working Group diagnostic package. This package currently makes use of several closed-source, non-free graphics packages (such as IDL and MatLab). Since our focus is on free and open-source software, we are first converting the OMWG diagnostic package to use NCL for all plotting. We will then convert it to Swift. This work is being done in collaboration with, and enthusiastic support of, developers of the OMWG diagnostics.

### Improvements to NCL

Two student intern projects were completed during the past year to improve NCL immediately. One project explored if OpenCL could be leveraged to improve the speed of selected NCL functions through parallel processing performed on commodity-level GPUs and multicore CPUs. A second project explored integrating the Earth System Modeling Framework (ESMF) regridding capabilities into NCL, allowing users to regrid to and from various topographically rectangular and unstructured grids. The regridding functionality is a highly requested user feature, and critical in the IPCC AR5 for comparing model runs generated on large, complex, and dissimilar grids. The ESMF work is being integrated into the next release of NCL.

As part of parallelizing NCL we needed to prioritize software development along costly, with respect to the time taken, code paths in the interpreter. Although there are existing NCL functions to time statements in NCL scripts, manually modifying NCL scripts require knowledge of the internal workings of the script as well as the interpreter. Moreover it is cumbersome if we need to modify multiple scripts, like in a diagnostic

package, to determine NCL functions that can be parallelized. Therefore we needed a simple way to profile NCL scripts that is agnostic of the internal workings of the script.

We developed a profiling layer for the NCL interpreter to automatically profile NCL scripts. If turned on at compile time the profiling layer monitors and records script resource usage statistics at runtime. The statistics are stored in a log file for further investigation.

## Progress on long-term tasks

ParVis' long term plan for enabling ultra-large climate data analysis involves developing a new high-performance data-parallel library for performing standard climate calculations on both regular and unstructured grids, the Parallel Climate Analysis Library (ParCAL). ParCAL will in turn be used to build a parallel version of NCL called ParNCL. We are also performing working to take in to account future hardware considerations including exploring compression, cloud computing approaches to analysis and new approaches to 3D visualization of climate data. Significant progress was made on all tasks.

### ParCAL development

Starting from scratch in March 2010, we have a working prototype of ParCAL. We asked the current NCL user community for their priorities and experience with operations that were to slow because of the volume of data. The results have helped us prioritize which functions to implement first in parallel. Three of the most time-consuming algorithms have been implemented and fully tested, including `dim_avg_n` (which calculates averages over an array's dimension, typically time), `max_element` and `min_element`. ParCAL is built upon the Mesh Oriented database (MOAB), Parallel Netcdf (PnetCDF) and Intrepid libraries. Details of dealing with the structured and unstructured numerical grids and I/O will be handled by MOAB and PnetCDF. The combination of MOAB and Intrepid enables the concise and succinct expression of the complex and computationally intensive algorithms such as computing vorticity and divergence.

The architecture of ParCAL is shown in Figure 3.

The **Fileinfo** class provides an abstraction of multiple files and a higher-level interface to manage files. Details such as the management of opening and closing a file, looking up which file contains a specific time step, information retrieval of file metadata etc. are all handled within the class. Underneath, MOAB and PNetCDF are employed to perform parallel read and write of data.

With the **PcVAR** class, access to the mesh data is simplified. If the mesh data is on disk, MOAB will be used to load data into the memory and a marker will be set to facilitate later access. The distinction between variables from a file and variables created by a user is necessary because the implementation needs to know whether to go to disk for the data or just allocate the space in memory.

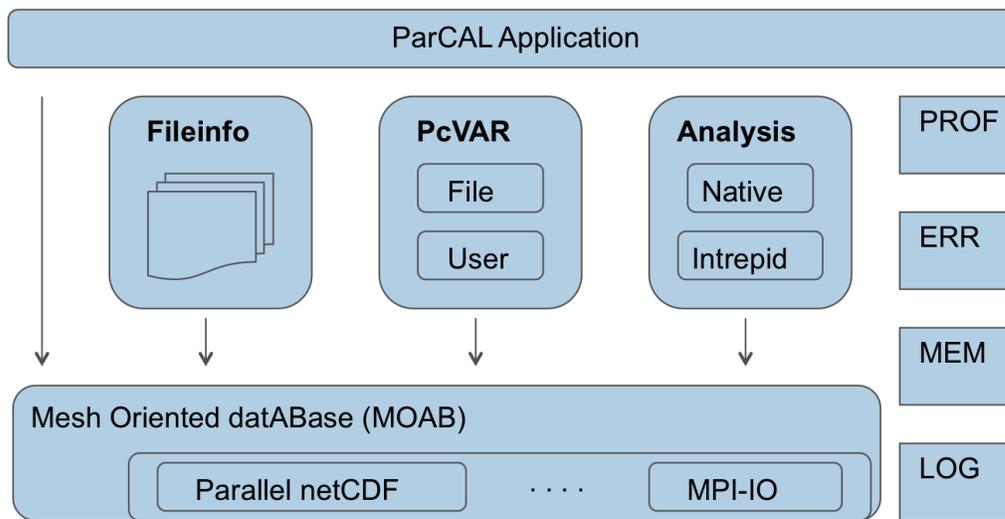


Figure 3: ParCAL Architecture

The **analysis** functionality is divided into two categories, native and Intrepid. Native algorithms are those implemented with only ParCAL or MOAB methods and datatypes while Intrepid algorithms are more complicated analysis functions dependent on Intrepid library.

Miscellaneous Functionality includes four major module: ERR is for dealing with program errors, LOG for logging functionality, PROF for performance profiling and MEM module for memory specific operations.

We are following modern software engineering practices in developing ParCAL. Autotools (autoconf, automake and libtool) have been used for generating automatic configuration scripts. “make”, “make install” and “make check” are all implemented. “make check” invokes unit tests created with the Boost Test Library - Unit Test Framework. All the algorithms implemented so far are fully tested. Cases such as reading only the header information, reading from a single file and from a series of files are all covered. Nightly testing is enabled with the buildbot system. Currently there are eight different configurations being tested every night, such as the trunk test, compilation for debugging, documentation generation, mpich compilation, openmpi compilation, compiled with gcc and intel compiler etc. ParCAL uses the Doxygen system for self-documentation and automatic generation of API documents.

## MOAB/ParCAL development

We have made several changes to MOAB to facilitate the development of ParCAL.

### *Serial, Parallel Import of Climate Data*

A new reader was implemented in MOAB for reading climate data from NetCDF-formatted files. Several new reader options were implemented in this reader:

- *Variable=<...>*: allows specification of one or more variables to read

- *Nomesh*: only read variables from the file; used for time-dependent data when the mesh was already read from a previous timestep
- *Timestep*=<...>, *Timeval*=<...>: for reading files storing multiple timesteps, allows specification of timestep to read, in terms of either timestep number or time value

Climate data files are encountered with data spread over files in various ways, e.g. a single file with multiple variables and timesteps, or one file per timestep. The reader options above provide the flexibility to read data organized in all those ways.

The NC reader has the option of reading in serial or parallel, using the Netcdf or Parallel-NetCDF libraries, respectively. This reader was benchmarked on up to 1024 processors of the Fusion cluster at Argonne.

### **Partition Methods**

Reading a structured mesh in parallel requires partitioning the mesh over processors, such that each processor owns a subset of the overall domain. The partitioning method can strongly affect the parallel load balancing and communication costs in subsequent operations. Four different partitioning methods were implemented in MOAB's structured mesh representation:

1. *Alljorkori*: a 1D partition that partitions the mesh in the j, k, or i dimension, whichever is largest.
2. *Allkbal*: a 2D partition that partitions the j and k dimensions with the goal of forming square (in j and k) subdomains while keeping the number of j and k values constant over all processors.
3. *sqij*: a 2D partition that forms square (in i and j) partitions, with some parts on the end of the i and j dimensions having one fewer i or j value.
4. *sjjk*: like sqij, but over the j and k dimensions.

A reader option, `PARTITION_METHOD`, was implanted to allow run-time selection of the partitioning method when reading a mesh. For the benchmark data used, there were relatively small differences in performance between these partition options, with *sqij* performing slightly better than the others.

### **Resolving Shared Entities**

A parallel mesh representation must be able to not only read the mesh partitioned over processors, but also identify portions of the mesh shared between processors. This is necessary to support operations like summing a field over vertices, i.e. to avoid double counting fields on shared vertices. MOAB provides a function for resolving this sharing, based on global vertex ids (which are established from information in the mesh file). MOAB's initial implementation of shared vertex resolution was for unstructured mesh, and the time spent resolving shared vertices was much greater than the time to read the data. Therefore, we implemented the capability to resolve shared vertices using the structured mesh information. This reduced resolve times by over an order of magnitude. The resulting read times for the benchmark problem were competitive with those of serial climate data analysis tools.

## Intrepid/ParCAL development

Progress on integrating Intrepid in to ParCAL has proceeded along 3 complementary directions.

**Intrepid-MOAB interaction** We have enhanced interoperability between MOAB and Intrepid. Three example drivers were written for solving three different partial differential equation systems with the finite element method. These examples use MOAB to read in the mesh and to provide access to mesh components and their connectivities and use Intrepid to define basis functions on cells, perform integration over cells, and for assembly of the linear systems.

**Data remap** We are developing and evaluating the necessary software infrastructure for data transfer (remap) between grids. Example code has been developed to map data from one grid onto another with bilinear interpolation using MOAB and Intrepid. An example of mapping the function  $f(x) = 1 + \sin(x) \sin(y)$  from a triangular Cartesian grid onto a quadrilateral Cartesian grid is shown in Figure 4. Interpolation examples have also been set up on finite volume latitude/longitude grids at several different resolutions. Evaluation of the example code reveals the need for an efficient, parallel search algorithm that will locate target grid nodes in source grid. Such an algorithm is required to facilitate incorporation of the data transfer into ParCAL.

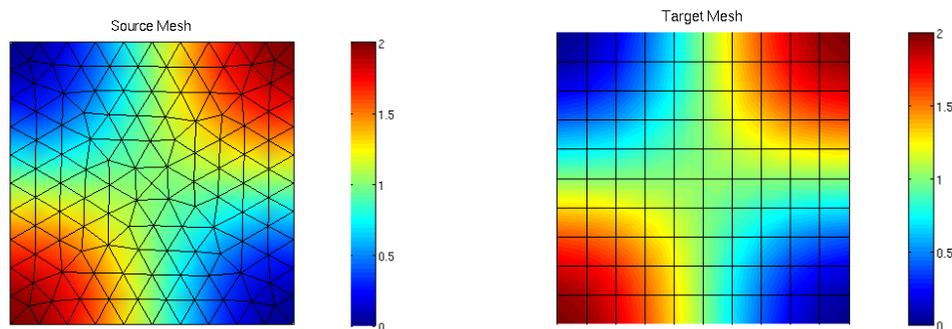


Figure 4 Interpolation of a scalar function from a (left) triangular source grid to a (right) quadrilateral target grid.

**Data processing** Computation of differential operators such as vorticity and divergence in NCL relies on interpolation of the data by globally supported spherical harmonic expansions. We have demonstrated an alternative approach based on piecewise polynomial reconstruction of the data by finite element basis functions which is more parallelizable than the spherical harmonic method. To this end, we wrote code to compute vorticity and divergence given a vector velocity field on a latitude-longitude grid. This code uses nodal basis functions and nodal velocity values to approximate the velocity field over a cell. Vorticity and divergence calculated using both NCL and the Intrepid method are shown in Figures 5 and 6. Initial work has been done to integrate the

code used for these figures into ParCAL.

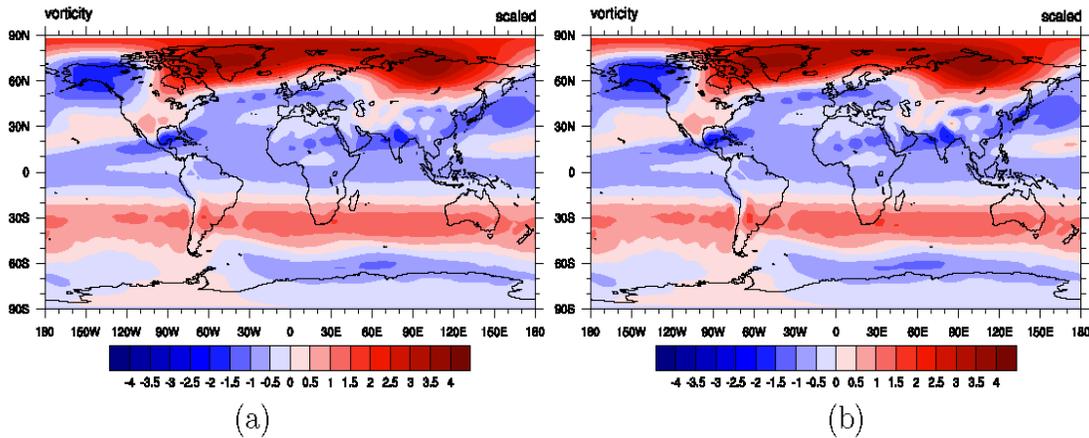


Figure 5 Vorticity plotted with NCL and calculated with the (a) parallel local cell approach of Intrepid and (b) serial spherical harmonic approach of NCL.

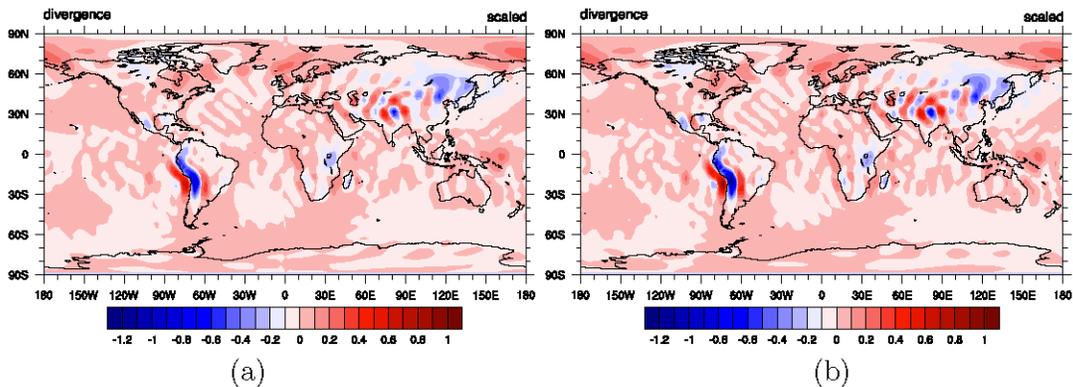


Figure 6 As in Figure 5 but for Divergence.

## ParNCL Development

ParNCL (Parallel NCL) is a parallel version of the NCL interpreter that will perform climate data analysis in parallel. It is being written using the ParCAL library.

The NCL interpreter is written in C and Fortran programming languages. As part of the ParNCL development we modified the NCL interpreter source code to add hooks to ParCAL (Parallel Climate Analysis library) to perform data analysis in parallel. We store the climate data read from NetCDF files using MOAB (A Mesh-Oriented Database). The parallel interpreter is launched like a parallel program written in MPI using the MPI job launcher.

One of the primary design goals of the project is to avoid user modification of the NCL scripts to run it in parallel. We achieved this goal by modifying the interpreter to run and perform data analysis in parallel. However this required several changes to the existing

data structures and interfaces inside the interpreter.

The overall architecture of ParNCL is outlined in Figure 7. A brief description of the design changes is given below.

- NCL Object
  - The interpreter has an object class hierarchy to represent NCL types (byte, integer, float etc.), data (one-dimensional, multidimensional etc.), variables (file variable, list variable etc.) and other NCL objects (graphics object etc.). The parent class of all objects is the NCL object. Since several types of NCL objects can potentially be a distributed object, we modified the NCL object data structure to optionally represent a distributed NCL object.
  - Since the values of these objects are potentially distributed, we modified the objects to store a handle to this storage. We also added functionality to gather this distributed data when needed.

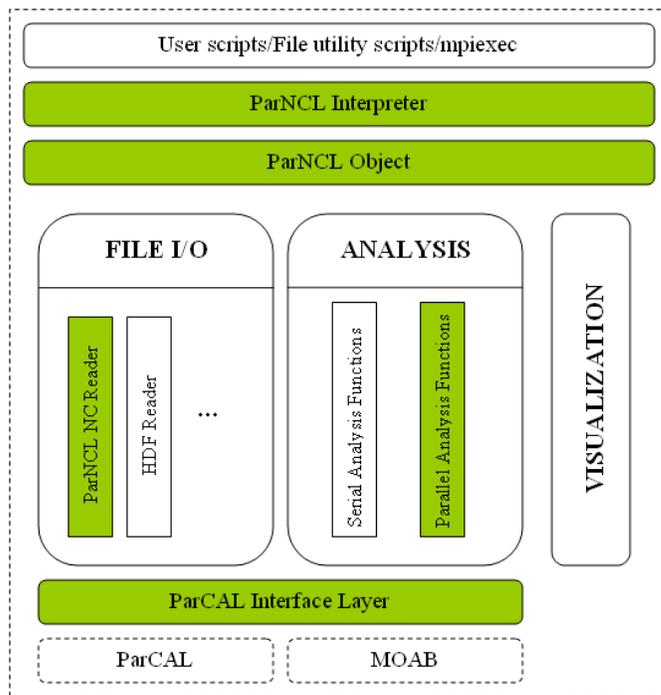


Figure 7 ParNCL Architecture

- File Object interface
  - The interpreter has a generic file object interface that provides access to the various file formats supported by NCL. This generic interface was modified to allow creating, opening, reading and writing multiple files in parallel.
- Parallel NetCDF reader
  - The File I/O module in the interpreter contains a reader for each file type (HDF4, HDF5, NetCDF4, Grib etc.) supported by NCL. We added a new file reader module for reading NetCDF data in parallel. ParNCL

interacts with MOAB to read and

write NetCDF data in parallel through ParCAL.

- ParCAL Interface layer
  - ParNCL interacts with ParCAL and MOAB to perform data analysis in parallel. However these two libraries are written in C++ and the NCL interpreter is written in C. Therefore we added a lite interface layer to translate between NCL and ParCAL data structures.
- Parallel database

- The climate data in NetCDF files is read using MOAB, a mesh oriented database, and stored inside this parallel database. ParNCL interacts directly and indirectly via ParCAL with MOAB to read, write data in parallel.

## Data Compression for Ultra-Large data sets

We developed a general framework based on the characteristics of high resolution climate data, which allow us to develop a range of algorithms that offer a tradeoff of: Compression ratio versus overhead, compression ratio versus the level of parallelism, and compression ratio versus information loss in lossy compression.

Our compression schemes contain two phases: the first phase predicts the next value based on the previous values, the second phase encoded the next value with entropy-based encoding. We have compared our scheme against other schemes and also evaluated various design choices. We have implemented a prototype and gathered some preliminary results. Experiments are done with various chunk sizes because chunk size can affect compression ratios and degree of parallelism-

Small chunk size: low compression ratio, high degree of parallelism

Big chunk size: high compression ratio, low degree of parallelism

We have evaluated our prototype (Figure 8) with an Intel Xeon 2.26Ghz, 6GB memory Dell workstation. We are using GCRM high resolution (about 15km) climate data. Compression speed is around 3Gbits per second when the data is in the main memory. It approaches the same order of magnitude of data retrieval from disks.

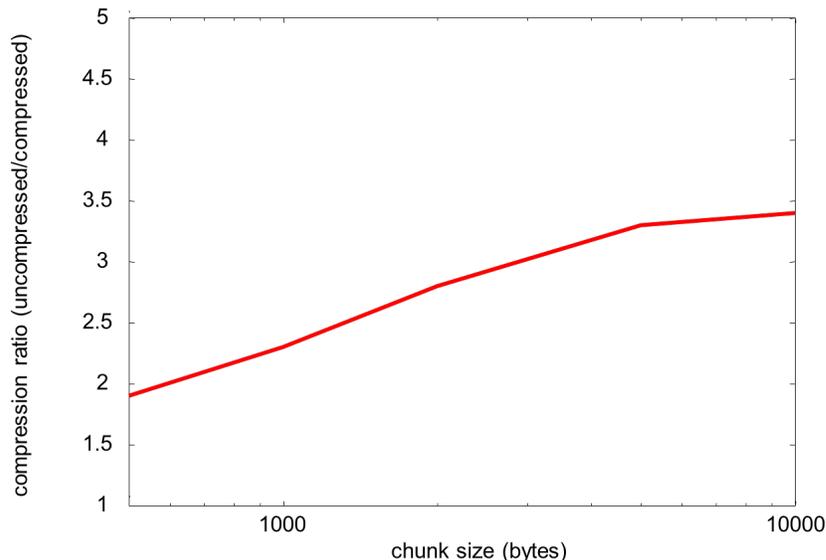
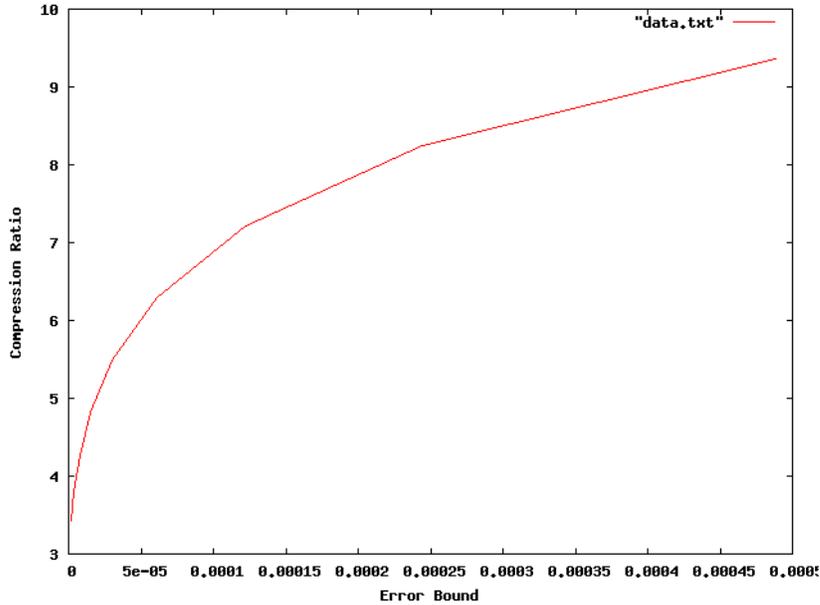


Figure 8 Lossless compression ratio as a function of chunk size.

We have also experimented with lossy compression. Our scheme allows one to bound the error for each value. Preliminary results show that we can achieve a compression ratio around 10 when the error bound is 0.1% (See Figure 9). Our lossy compression



scheme also allows multilayer compression. Depending on the required precisions, different layers can be used. For high precision, use more layer. For low precisions, less layers also means less I/O overhead.

Figure 9 Lossy compression ratio as function of error bound.

## Climate analysis and visualization in the cloud

MapReduce is a functional programming approach that has been used with great success by Google and other commercial entities on cloud computing platforms. MapReduce comprises two stages: A **Map ()** stage in which input data are transformed into a set of *(key, value)* pairs, and a **Reduce ()** stage in which the *(key, value)* pairs produced by the Map stage are aggregated and analyzed, resulting in a set of answers that are also *(key, value)* pairs in which each key appears only once. MapReduce is a potent programming model because its framework offers support for automatic parallelism with fault-tolerance; the MapReduce user need only implement the requisite **Map ()** and **Reduce ()** functions, which are executed within the framework.

This approach is highly applicable to climate data analysis. The narrative of a typical climate data analysis is: From input data defined in a  $d$ -dimensional ( $d$  typically  $\leq 5$ ) space—three spatial dimensions, time, and variable—create a  $(d-s)$  set of  $s$ -dimensional subsamples from this multidimensional space, and for each subsample, compute statistics or other diagnostics, resulting in a  $(d-s)$ -dimensional set of results. Thus, **Map ()** is the act of sample formation and **Reduce ()** is the act of diagnostic/statistic calculation. The “keys” in the *(key, value)* pairings generated by **Map ()** and acted on by **Reduce ()** can be either a  $(d-s)$ -tuple, or as simple as a virtual linearization of the  $(d-s)$ -dimensional index space identifying the subsamples. A set of **Map ()** and **Reduce ()** archetypes have been identified for typical climate data analyses, examples for spatiotemporal averaging of a timeseries of 2D slices are presented in Table 1.

TABLE 1. SPECIMEN MAPREDUCE ARCHETYPES FOR AVERAGING

<b>Operation / Yield</b>	<b>Map</b>	<b>Intermediate Key</b>	<b>Reduce</b>
Grid-point timeseries average of 2D field / 2D averaged field	Identify gridpoint timeseries	Gridpoint index space tuple or virtual linearization thereof	Average on timeseries ID key
Global Average / Scalar timeseries plot	Identify time slices	Time index	Average for each 2D slice by ID key
Zonal Average / (zonal,time) contour plot	Identify zonal band for each time slice	(zone,time) index tuple or virtual linearization thereof	Compute average for each band by ID key
Zonal and Time Average / Scalar zonality plot	Identify zonal bands	Zonal index	Compute average of for each ID key

A prototype implementation of a MapReduce climate data analysis toolset has been implemented and tested on a stand-alone (uniprocessor) Hadoop system. Hadoop was chosen because it is a highly portable (from laptops to clusters) open-source software framework. A more compelling reason for choosing Hadoop is that it supports *streaming*; that is, one can implement stand-alone Map and Reduce executables and obviate the need for coding Map/Reduce functions in Java or confronting language interoperability issues. The kernels developed thus far are written in Python. Python is a language renowned for ease-of-use in rapid prototyping. Python leverages a large collection of community-developed mathematical and statistical packages (e.g., numpy and scipy). Finally, Python obviates the requirement of implementing in Java MapReduce input (output) readers (writers) for climate data formats as Python offers widely used packages supporting netCDF, hdf, and GRIB can be leveraged.

The prototype system supports only the netCDF file format. It does this by using the pupynere (<http://pypi.python.org/pypi/pupynere/>) API, which operates on the assumption that netCDF files it is presented conform to the netCDF file format resulting from the netCDF reference implementation. Thus, the prototype does not require the netCDF library to be installed on the target system. The present set of kernels support spatiotemporal averaging of univariate data samples. Performing averaging along a different dimension—zonal averaging—requires only a change in Map function to emit a different set of keys to be paired with the data values.

Execution of these kernels is accomplished within the streaming Hadoop framework. Performance analysis has been deferred until some of the prototypes are running on a Hadoop cluster, but the main determinants of scaling will be how amenable a particular application is to multiple Map and Reduce processes; that is, the Map function will parallelize well for analyses comprising large numbers of source files and the Reduce function's scaling will be determined by the degree of granularity (i.e., the number of unique "keys" resulting from a the applications Map operation). This is the next development step, to be followed by performance studies of large-scale test problems.

Optimization will be pursued as needed. Future development steps will involve interoperability with ParCal.

### 3D visualization and analysis

We have been continuously enhancing the capability of our 3D visualization software ICAV. One direction is adding the ability to render 3D hexagonal mesh data such as encountered in the GCRM at high quality and speed. Hexagonal meshes have become increasingly popular, and an efficient visualization solution for large data will soon be in high demand. Production software tools such as VisIt renders hexagonal mesh data by resampling the mesh into tetrahedral cells first. The problem with this approach is the excessive amounts of tetrahedral cells generated and the potential imprecision of rendering tetrahedral cells that are treated as linear elements. We have experimented with a few different approaches including different ways to resample the mesh into one that can be stored and processed more efficiently with a GPU. After extensive testing, we found that remeshing does not promise a scalable solution, and have thus decided to attempt direct rendering of the hexagonal mesh. For this study, we had used a toy dataset so far. We have obtained a large GCRM dataset that will enable us to better evaluate our work with respect to quality, accuracy, and scalability. We have also upgraded our ICAV with a cleaner implementation of the rectilinear-grid volume renderer and integration with NCL images (Figure 10).

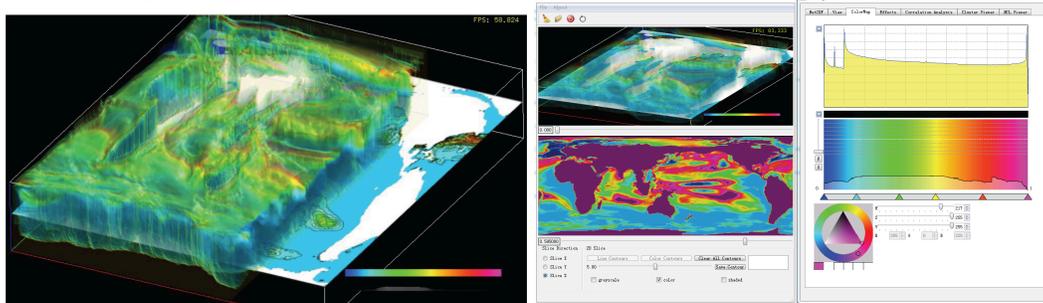


Figure 10 Coupling of volume rendering of the KAPPA isopycnal diffusion coefficient across the globe couple with a corresponding NCL image.

Users can now freely orient the visualization, control color and transparency of the volume, cut away and couple with NCL images (Figure 11).

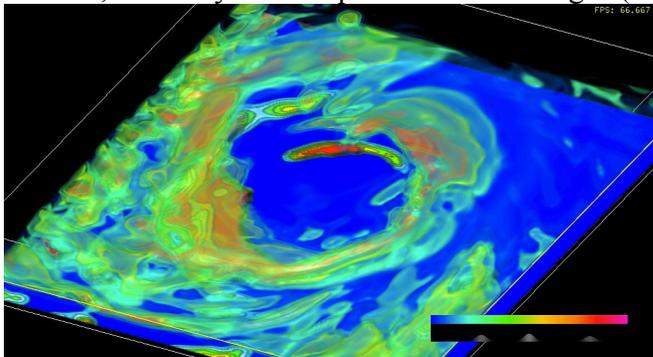


Figure 11 Ice mixing ratio from a hurricane simulation.

## Project Management

### Project organization and resources

The PI is responsible for coordinating effort among the various tasks and insuring progress is made on deliverables. The project is spread over 5 institutions and a “lab lead” at each is responsible for coordination of the ParVis members at their respective institutions. The leads are: Robert Jacob (ANL), Pavel Bochev (Sandia), Karen Schuchardt (PNNL), Don Middleton (NCAR) and Kwan-Liu Ma (UC-Davis).

All team members participate in biweekly conference calls devoted to updates and discussion of near-term development. The ANL web and audio service provider, AdobeConnect, is used to facilitate sharing presentations and recording notes from the call. Two mailing lists hosted by Argonne are also used by the team: one for general discussion (parvis) and another for development details and code check-in messages (parvis-dev).

We have biannual all-hands meetings. Our kickoff meeting was held Sept 29-30, 2010, at Argonne National Laboratory and the second meeting was April 11-12, 2011, at NCAR. A brief third meeting was held in conjunction with the DOE ESM PI meeting on Sept 21, 2011.

The PI keeps the ParVis advisory panel (David Randall (CSU) and William Gustafson (PNNL), Gokhan Danabasoglu (NCAR), Cecilia Bitz (University of Washington) and David Lawrence (NCAR)) advised of progress and solicits feedback from them.

The MCS division at Argonne provides resources for software development (svn repository, bug tracking and test/development machines). We have also obtained an allocation of computer time on Argonne’s Fusion cluster for testing on tens to hundreds of processors. ParVis developers have been given access to the Eureka analysis/viz cluster at the Argonne Leadership Computing Facility through the INCITE project led by Warren Washington (time on Eureka is not charged to the project)

### Communication with the broader community

We maintain a website (<http://trac.mcs.anl.gov/projects/parvis>) to both host software we make available for the community and provide notes and material for ParVis team members. Most of the content is world readable except for the repository and the ticket system. As ParCAL matures, we will open up the repository for anonymous checkout of the source code. We also maintain a one-way mailing list (parvis-ann) that anyone can subscribe to for announcements about ParVis and ParVis software.

We introduced ParVis to the community at the 16<sup>th</sup> CESM Workshop in June, 2011. There was an overview poster in the poster session as well as a talk (“Introducing ParVis”) during the Software Engineering Working Group meeting. We also emailed the

co-chairs of all the CESM working groups to alert them to the project and its goals. A second talk and poster was given at the DOE ESM PI meeting in September of 2011. The ParVis PI along with the PI's of the other visualization projects submitted a successful session proposal for the Fall AGU meeting that will further educate the community about our efforts.

## Interaction with other projects

We have had discussions with the other LAB10-05 projects on how to collaborate. We identified a possible interoperability path through the VTK data model that is common to all the software under development.

Under other funding, NCL developers have implemented NCL as a backend service for the Live Access Server (LAS) to perform simple visualizations and analysis computations on the Earth System Grid (ESG), a data distribution portal that houses scientific data collections---including CMIP5---at sites around the globe. This path will allow ParNCL to also be used as a backend for LAS on ESG.

Members of the BER "Ultra High Resolution Global Climate Simulation" project (PI: Jim Hack, ORNL) have contacted us about using the Swift-based AMWG diagnostics to help analyze their data. We are working with developers of HOMME, a dycore scheduled to be the default in the next release of CAM and the main atmosphere model in the CSSEF project, to ensure its grids can be read efficiently by ParCAL.