Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1245 – 1254

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

ParNCL and ParGAL: Data-parallel tools for postprocessing of large-scale Earth science data

Robert Jacob[a,*], Jayesh Krishna[a], Xiabing Xu[a], Tim Tautges[a], Iulian Grindeanu[a], Rob Latham[a], Kara Peterson[b], Pavel Bochev[b], Mary Haley[c], David Brown[c], Richard Brownrigg[c], Dennis Shea[c], Wei Huang[c], Don Middleton[c]

[a]*Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439, USA*
[b]*Sandia National Laboratory, P.O. Box 5800, Albuquerque, NM 87123, USA*
[c]*National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307, USA*

Abstract

Earth science high-performance applications often require extensive analysis of their output in order to complete the scientific goals or produce a visual image or animation. Often this analysis cannot be done in situ because it requires calculating time-series statistics from state sampled over the entire length of the run or analyzing the relationship between similar time series from previous simulations or observations. Many of the tools used for this postprocessing are not themselves high-performance applications, but the new Parallel Gridded Analysis Library (ParGAL) provides high-performance data-parallel versions of several common analysis algorithms for data from a structured or unstructured grid simulation. The library builds on several scalable systems, including the Mesh Oriented DataBase (MOAB), a library for representing mesh data that supports structured, unstructured finite element, and polyhedral grids; the Parallel-NetCDF (PNetCDF) library; and Intrepid, an extensible library for computing operators (such as gradient, curl, and divergence) acting on discretized fields. We have used ParGAL to implement a parallel version of the NCAR Command Language (NCL) a scripting language widely used in the climate community for analysis and visualization. The data-parallel algorithms in ParGAL/ParNCL are both higher performing and more flexible than their serial counterparts.

Keywords: data analysis; data parallelism; postprocessing

1. Introduction

Today's petascale systems, such as those operated by the DOE Leadership Computing Facilities at Argonne (ALCF) and Oak Ridge National Laboratories (OLCF), are being used extensively by the earth sciences [1]. In most high performance computing applications, new knowledge is only gained after significant analysis is done on the output of the petascale simulation, often referred to as "post-processing". In the case of climate modeling, the direct output which may measure in the terabytes for a single multi-million core-hour simulation tells little about the climate. It is only after multivariate time-series analysis (post-processing) is performed on that data and comparisons with other runs and observations that something new can be learned.

*Corresponding author. Tel.: +1-630-252-2983 ; fax: +1-630-252-5986.
E-mail address: jacob@mcs.anl.gov.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.291

The programs currently used to perform this analysis in the earth sciences are often not nearly as flexible or high-performing as the primary applications. They are often single threaded and/or 32-bit and may assume structured grids are being used. In many cases, they either break or require workarounds for the ultra-large unstructured-grid data that is becoming the norm in computational earth sciences. In climate science, they are already a bottleneck [2]. Post-processing of ultra-large data sets present not just a complexity and performance challenge to current tools, but also a memory challenge. The single threaded programs will typically assume that they can read all the data in to memory. This requires further workarounds where the researcher uses command line tools to reduce the size of the data to something that can be held in the memory of a single node.

The hardware to X analyze multi-terabyte gridded output data is available. Both the ALCF and OLCF have dedicated "Data Analysis and Visualization" (DAV) clusters attached to the same disk as the primary compute platform and containing hundreds of "fat" nodes with powerful CPU's, parallel file systems and large amounts of memory (*lens* at OLCF and *eureka* at ALCF). But there is a distinct lack of analysis software that can take advantage of those platforms. The President's Council of Advisors on Science and Technology (PCAST) 2010 review of Networking Information Technology Research and Development (NITRD) [3] said that one of the major challenges in data analysis was "computational models and languages suited for expressing data analysis algorithms that map onto large-scale, parallel systems."

Programs such as Parallel-R [4] provide data-parallel versions of some of its statistical analysis functions. However it does not support operations on a grid. To accurately calculate gradients and other features from output data, it is necessary for the tool to have a representation of the discretization used in the original model. Tools such as GLEAN [5] or DIY [6] provide facilities for data staging and movement in an HPC environment but not the grid-aware data model or domain-specific algorithms we need.

In this paper, we describe a new Parallel Gridded Analysis Library (ParGAL,§4) that is built on a computational model that can map onto large analysis clusters (or petascale systems) and explicitly represent the discretizations used in earth science models. We have focused our initial development on supporting analysis of climate model output. ParGAL provides high-level parallel algorithms that can operate on structured or unstructured grid data in parallel. The library builds on several existing scalable systems (§2) for its data model, algorithm expression and I/O and some modifications were required to these systems(§3). We are using ParGAL to build a data-parallel version of a popular domain-specific analysis and visualization scripting language, the Parallel NCAR Command Language (ParNCL), which will both allow it to scale and operate on multiple grid types. (§5).

2. Components of ParGAL

Many of the features we wanted ParGAL to have, such as a data model for structured and unstructured grids and a way to define operations within and across grids, were already implemented in other systems. Although all the ParGAL code is new, it has been built on several existing pieces of software.

2.1. The Mesh-Oriented Database: MOAB

MOAB is a library for query and modification of structured and unstructured mesh and field data associated with the mesh [7]. MOAB can represent all entities typically found in the finite-element zoo, as well as polygons and polyhedra. Structured mesh is supported as well, with a special interface providing parametric block information [8]. The data model implemented by MOAB references four distinct data types:

- **Entity:** vertices, triangles, quads, etc.
- **Entity Set:** arbitrary collection of entities and other sets
- **Interface:** object through which all other functions are called (i.e., the database)
- **Tag:** information stored on Entity, Entity Set, and Interface objects

This data model has proven remarkably versatile, able to represent most semantic information associated with typical meshes, including boundary conditions, solution fields, geometric associativity, and parallel partitions. Internally, MOAB uses an array-based storage model; this allows efficient access to and iteration over fields associated with mesh entities, including vertex- and element-based variables. MOAB uses the HDF5 library for its native save/restore format [9].

For parallel access, a mesh is represented and queried in MOAB as a serial mesh local to a processor, with information about the parallel nature of the model accessed (and stored) in the form of sets and tags. For convenience, MOABs `ParallelComm` class also has functions that provide this data and functions for performing commonly needed parallel functions. For any entity shared with other processors, MOAB stores both the remote processor rank(s) and the handle(s) of the entity on those processors, on all processors sharing the entity [10]. Mesh models are initialized in parallel by reading mesh from a single file in parallel, using a partition stored as entity sets in the file. A partitioning tool has been implemented by interfacing with the Zoltan partitioning library [11].

The underlying structured grid representation in MOAB stores connectivity information implicitly, for memory efficiency, while storing vertex locations explicitly, for generality.

2.2. Intrepid

Intrepid is a Trilinos [12] package for advanced discretizations of partial differential equations (PDEs) [13]. the abstract framework for compatible discretizations [14] provides the mathematical foundation of Intrepid. This framework prompted reevaluation of conventional software design for PDEs, which usually focuses on a single discretization paradigm. In contrast, Intrepid aims to translate mathematical similarities between finite-element, finite-volume, and finite-difference methods, identified in [14] into software-based similarities. Intrepid has been used to implement numerical methods for PDEs ranging from mimetic least squares for magnetostatics [15] to control volume finite-element methods for semiconductor equations [16].

Intrepid offers a wide range of cell-based tools for the implementation of finite-element, finite-volume, and finite-difference methods for PDEs. The package represents a *middleware* between higher-level software infrastructures and lower-level cell-based numerics, for example, for evaluation of basis functions, coordinate transformations, surface parameterizations, and integration of fields over cells, cell faces, and cell edges. Intrepid is designed to operate locally on batches of cells having the same topology and data type. A key aspect of the design is that Intrepid separates cell topology from the reconstruction (i.e., the field evaluation process). In other words, a reconstruction "basis" and its evaluation points are not tied to a particular cell topology. This design approach allows Intrepid users to "mix and match" cell topologies with reconstruction operators ("bases") and evaluation points, thereby enabling a virtually unlimited generation of new discretization methods from a small number of basic components.

The ability to "mix and match" an extensive range of fields, cells, and evaluation points enables Intrepid to interpret and evaluate virtually any kind of numerical data generated by computer simulations. This makes the package a powerful and flexible tool for data analysis and processing. The ParGAL effort is the first utilization of Intrepid in this application context. ParGAL uses Intrepid to implement forward data operations such as computation of divergence and vorticity from a given velocity field and interpolation between different grids. In addition, ParGAL takes advantage of the discretization capabilities of Intrepid to implement operations such as computation of a stream function and a velocity potential from a given velocity field.

Traditional spherical harmonics approaches for these tasks (often used in other climate analysis tools) require global data. In contrast, because Intrepid is rooted in local cell-based operations, it does not require global data and can compute the stream function and the potential on any limited domain. When combined with MOAB support for parallel mesh-based communication, the local nature of Intrepid operations makes the combination particularly well-suited for parallel analysis of simulation data.

2.3. PNetCDF

The climate community makes heavy use of the NetCDF self-describing portable file format and its associated programming interface [17]. Portable in this context means the dataset can be moved from machines with different byte-endianness or datatype sizes without needing to change the client code reading or writing those files. Self-describing means the dataset has enough internal structure that client code can use the associated APIs to determine the kind and quantity of variables contained in the dataset. Further, the NetCDF library provides a means for assigning "attributes" (metadata) to variables, dimensions, and datasets, offering yet more documentation for the data contained therein. Collaborators at different institutions running on different computing resources rely on both the self-describing and portability features of NetCDF in order to understand colleagues' work now and in the future.

For parallel I/O needs, the Parallel-NetCDF project [18] provides a parallel programming interface. Parallel-NetCDF maintains the same NetCDF file format and same concepts of attributes, dimensions, and variables but provides an alternative (though similar) API for parallel programming. This alternative API introduces MPI concepts such as communicators and "info" tuning parameters but retains the spirit of the serial API. Parallel I/O happens through the MPI-IO [19] interface, but the library can abstract away details such as file views and MPI datatypes. Parallel-NetCDF emphasizes "collective I/O," where all processes participate in an I/O operation. Typically, the MPI-IO library can apply several powerful optimizations to a collective I/O workload. Particularly useful are so-called deferred-mode parallel operations, where the application specifies a series of data read operations, then frees Parallel-NetCDF to execute them all asynchronously. This approach allows the library to combine both I/O and communication operations for maximum efficiency.

While Parallel-NetCDF (and serial NetCDF) provide a good interface for regular array access, climate analysis models have grown more sophisticated in the years since these I/O libraries were first designed. Integrating Parallel-NetCDF into MOAB, discussed in Section 2.1, allows us to support these more sophisticated analysis models for more than just structured grids. MOAB provides the richer description of the grids used in climate analysis, and Parallel-NetCDF provides the optimized parallel I/O for that analysis.

3. Modifications to Components

Our building blocks for ParGAL required some modifications to meet the needs of a data-parallel analysis suite for climate model output.

3.1. MOAB

Although MOAB supported structured grids before the development of ParGAL, it did not have support for reading climate-specific data models, nor was the structured grid capability versatile enough to support the full range of climate data analysis algorithms. Enhancements to MOAB were in three primary areas: the representation and initialization of structured grids in parallel; reading data from climate-specific netcdf-format data files; and the representation of spectral element-based mesh and data. Each of these areas is described below.

3.1.1. Representation and Initialization of Structured Mesh

Structured meshes have been used extensively to model the globe. Often, a single 2D structured mesh is used, with periodicity in the horizontal (longitudinal) direction and singularities at the two poles. For parallel representations, the global grid is typically partitioned over processors, with each processor representing a contiguous, rectangular portion of the global *ij* space.

Partitioning a mesh over processors seeks to balance the relative work for each processor, while also minimizing communication between processors. In practice, the ideal method for partitioning a structured mesh over processors depends on the dimensions of the grid in each parametric direction. This can be more challenging when seeking rectangular subdomains, since the number of processors is often not an integral divisor of the number of cells in either parametric direction. Various partitioning strategies were developed, including "alljorkori" (first *j* or *k* or *i* larger than the number of processors partitioned over processors); "alljkbal" (partition *j*, and possibly *k*, seeking square subdomains); and "sqij" / "sqik" (mostly-square subdomains in *i* and *j/k*, with extra rows/columns towards bottom and left/front). By default, we chose the "alljorkori" method, since this method resulted in the largest contiguous reads of data off the disk (data is laid out such that *i* varies fastest). However, this issue will need revisiting after parallel I/O performance is improved.

Depending on the partitioning method used, in parallel the local mesh may or may not be locally-periodic in the horizontal direction. For example, if the "alljorkori" method is used and the *j* direction is chosen for partitioning, each processor will contain a complete horizontal strip of the grid, and therefore the grid will be "locally-periodic" in the *i* direction, i.e. cells on the "right" side of the local subdomain will be connected to vertices on the "left" side of the local subdomain. On the other hand, if the "sqij" partitioning strategy is used, then the grid will be non-locally-periodic, with cells on the "right" side of the subdomain sharing vertices with another processor (even though those vertices may be on the "left" side of the global grid). Initializing structured periodic grids to support

the partitioning strategies described while getting the correct inter-processor grid sharing was crucial to getting the correct results for climate data analysis.

To discover vertex sharing between processors, MOAB has an algorithm based on the assumption of unstructured mesh [10]; this method requires more global communication than is strictly necessary for a structured mesh. MOAB was also modified to compute vertex sharing based only on knowledge of the partitioning method, the global parametric dimensions of the mesh, and information about the starting vertex handle for neighboring processors. Figure 1 shows the impact of this improvement, where the vertex sharing computation time is reduced by an order of magnitude.

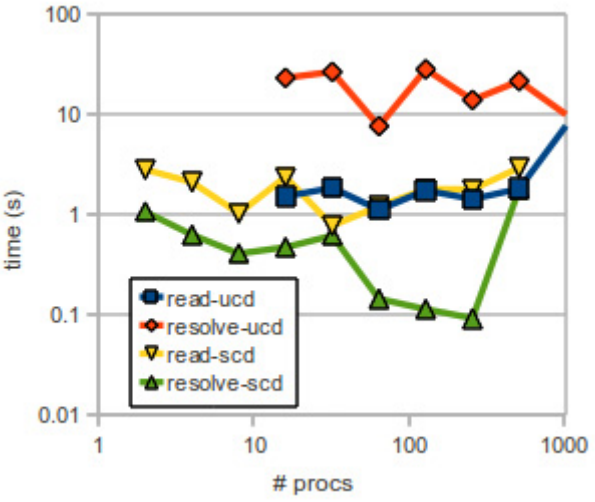

Fig. 1. Timing results showing improvement from deterministic shared vertex resolution.

Finally, although climate data simulations are performed in 3D, in practice the data is saved to disk, and many of the discrete operators are applied, based on a 2D x 1D arrangement of the data. That is, the grid representation is 2D, with a vector of data on grid vertices representing levels of the third dimension. MOAB's netcdf reader was written to reflect this same arrangement of data in MOAB, where the grid is 2D, with 1D tags representing the level-based data.

3.1.2. Efficient Reading of Structured and Unstructured NetCDF Files

Although only 2D structured and unstructured mesh is used to support climate data simulations, the different discretization methods can associated the discrete variables with different mesh entities. For example, Finite Volume-based simulations place variables on cell edges and faces, while Finite Element- and Spectral Element-based methods place variables mostly on cell vertices. MOAB's climate data reader was written to recognize the variations in placement of data, based on the dimension names used in the NetCDF file [1]

3.1.3. Spectral Mesh Representation

The High-Order Method Modeling Environment (HOMME) spectral element (SE) code uses an unstructured quadrilateral grid, with each quadrilateral containing an *NxN* grid of "Gauss-Lobatto" points for degrees of freedom [1]. There are two options typically used to represent SE meshes for data analysis, one which translates each quadrilateral into an *NxN* grid of bilinear quadrilaterals, and one where the true spectral element (and its square grid of data) are represented and analyzed using a true SE discretization. MOAB has been modified to provide both representations, controlled by a read-time option, along with functions for changing between the two at runtime. Options for controlling SE mesh representations have been documented in the MOAB User's Guide. This is a relatively new capability, though, and so the corresponding changes have not been made in the Intrepid-based evaluation of the true SE-based data.

3.2. Intrepid

The Intrepid library is designed to perform operations on batches of cells without reference to the global mesh connectivity. Therefore, the bulk of the work for the Intrepid-dependent algorithms has been to create an intermediate computational layer that takes results from cell-based derivative or evaluation operations and scatters

[1] Conventions for dimension naming in climate NetCDF files are governed by the "CF metadata conventions"[20].

them to arrays based on the mesh connectivity. In the case of the vorticity computation discussed in (§5) additional Trilinos components have also been incorporated to solve the resulting linear system.

Modifications to the MOAB reader that allow variables to be associated with different mesh entities (i.e. vertex, face, cell center) have led to different algorithm formulations for the intermediate computational layer. Currently, cell centered and vertex variable computations are supported, where computations for vertex quantities use the primal mesh defined in MOAB and computations for cell-centered quantities use a dual mesh, which must be inferred from the primal mesh representation.

In addition, the geometry of the native geophysical grid has been taken into account in the metric terms required to compute derivatives. The Intrepid library provides mappings from reference to physical cells given a physical cell in Cartesian coordinates, but for a physical cell in curvilinear coordinates the mapping requires modification. At this time spherical metric terms have been included in the intermediate computational layer for structured latitude-longitude grids and for the cubed-sphere grid used by the HOMME spectral element code.

4. ParGAL Architecture

ParGAL leverages the capabilities of MOAB, PNetCDF, and Intrepid libraries to accomplish efficient, parallel, discretization-accurate data analysis. ParGAL uses a modular design based strictly on the C++ standard [21] for portability. ParGAL is designed to simplify implementation and evaluation of a wide variety of discretization-specific algorithms on a wide range of grid types and very large data sizes.

ParGAL's interfaces encapsulate details about file reading, parallel partitioning, and mesh-based parallel communication, so that the algorithm designer can focus on analysis. ParGAL comprises four main components:

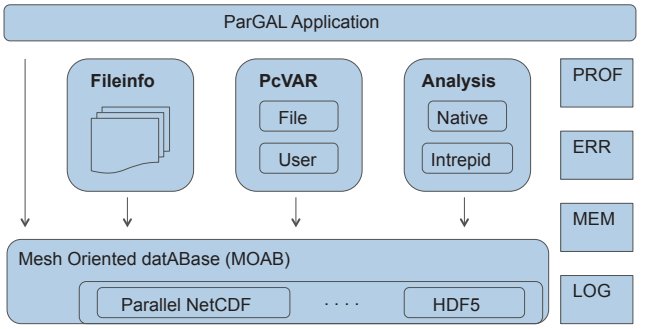

Fig. 2. ParGAL Architecture.

Fileinfo, PcVAR, Analysis, and Support. Figure 2 illustrates the architecture of ParGAL and the interaction among the components. The details of each component are given below.

The Fileinfo class provides an abstraction of a single file or multiple files and a higher-level interface to hide lower-level details of file management, including opening and closing a file, looking up which file contains a user-specified time step, and retrieving information about file metadata. It also expands the capability of the lower-level libraries used. The current MOAB NetCDF/Parallel-NetCDF reader stores file metadata for a single series of files. With Fileinfo, multiple instances of the class can be used to store file metadata for multiple different file series.

ParGAL is designed to work with various large-scale structured and unstructured numerical grids. A MOAB mesh instance serves as the database or container for most of the "heavy" data, while ParGAL provides a higher-level index and summary of that data. PcVAR is built on top of MOAB to encapsulate the details of variable data access. For instance, it keeps track of whether a specific time step is loaded. If not, MOAB will be used to load data into the memory, and a marker will be set to facilitate later access. Otherwise, the marker will be returned. The distinction between variables from a file and variables created by a user is necessary because the results of some analysis routines need to be stored and the lower-level implementation needs to know whether to go to disk for the data or just allocate the space in memory if it is the first time the data is accessed.

The Analysis module contains the analysis routines implemented in ParGAL. Most of them will take PcVAR's as input arguments and output results either into a scalar or another PcVAR, similar to how C++ STL's

Table 1. ParGAL Function Table

Algorithm	Description
max_element	return the maximum element of a variable.
min_element	return the minimum element of a variable.
dim_avg_n	computes the average of a variable's given dimension at all other dimensions.
dim_max_n	computes the maximum of a variable's given dimension at all other dimensions.
dim_min_n	computes the minimum of a variable's given dimension at all other dimensions.
dim_median_n	computes the median of a variable's given dimension at all other dimensions.
vorticity	calculates vorticity from a velocity field on a rectilinear or cubed-sphere grid. Intrepid is used to calculate the partial derivatives assuming a bilinear approximation of velocity on a grid cell.
divergence	calculates divergence from a velocity field on a rectilinear or cubed-sphere grid. Intrepid is used to calculate the partial derivatives assuming a bilinear approximation of velocity on a grid cell.
gather	gather the value of a variable to root 0.

generic algorithm works. The analysis functionality is divided into two categories, native and Intrepid-based. Native algorithms, implemented with functionality provided by ParGAL or MOAB, involve mostly straightforward data-parallel arithmetic operations on mesh-based fields, while Intrepid algorithms are used for more complex discretization-based algorithms. Table 1 shows the algorithms that we have implemented so far and their functionality. With the current ParGAL design, the various native and Intrepid-based algorithms can be implemented succinctly.

Four support modules provide basic library functions. ERR is for program errors. We are using C++ exception handling mechanisms, and the exception thrown also contains the file name and source line number where the exception is thrown. The LOG module provides logging functionality, the PROF module is for performance profiling, and the MEM module is for memory-specific operations.

5. Application of ParGAL: ParNCL

ParGAL and its components provide powerful tools to build data-parallel analysis algorithms for many types of gridded data. A developer must have some knowledge of both C++ and parallel programming. To make the powerful features of ParGAL more accessible to Earth scientists, we have used ParGAL to create a data-parallel version of the NCAR Command Language (NCL). NCL [22] is a free interpreted language that is widely used for data analysis and visualization, especially in the climate community. NCL offers a wide array of data analysis operations ranging from simple math operations such as finding the minimum element in an array to sophisticated domain-specific operations. The two-dimensional plots rendered by NCL are publication quality and highly customizable (climate scientists use two-dimensional figures instead of three-dimensional visualizations because the aspect ratio of their system is very small). Earth scientists collectively have developed thousands of lines of NCL scripts to perform postprocessing on the output from models and to analyze and visualize data. We have developed a parallel version of the NCL interpreter, ParNCL, that performs data analysis in parallel using ParGAL and MOAB.

Several data structures in the NCL interpreter were extended to parallelize the interpreter. A new parallel file reader was added to the interpreter to read data in parallel. We also added an interface layer to facilitate access to the ParGAL and MOAB libraries. ParNCL interacts with MOAB to access data stored in NetCDF files and create new data corresponding to user variables specified in an NCL script. The interpreter uses the ParGAL functions to perform data analysis on the data stored in the parallel database provided by MOAB.

ParNCL supports all the data analysis operations implemented in ParGAL. Apart from these operations the interpreter supports simple math operations such as adding, subtracting, and scaling multidimensional NCL variables and global operations such as calculating the sine of all the values in a multidimensional NCL variable. The

interpreter also supports subscripting arrays of data using the NCL variable subscripting rules. So far, we have not modified any visualization algorithms in the NCL interpreter. Once the data analysis is complete, the single threaded visualization algorithms are used to plot the results.

The multidimensional variables read from the NetCDF files are stored in a parallel mesh database provided by MOAB. A PcVAR variable is created for each of these multidimensional variables and stored with it. This PcVAR variable is used for all data analysis operations that use ParGAL.

5.1. Comparison of ParNCL Vorticity Calculation

In this section we compare the performance of ParNCL with NCL on a typical analysis function. As discussed above, ParNCL uses ParGAL and performs data analysis in parallel while NCL performs the data analysis using a single thread. We compare the performance of the single-threaded NCL interpreter with ParNCL using the NCL function uv2vrF(), a function that computes the vorticity given the U and V wind components on a fixed rectangular grid.

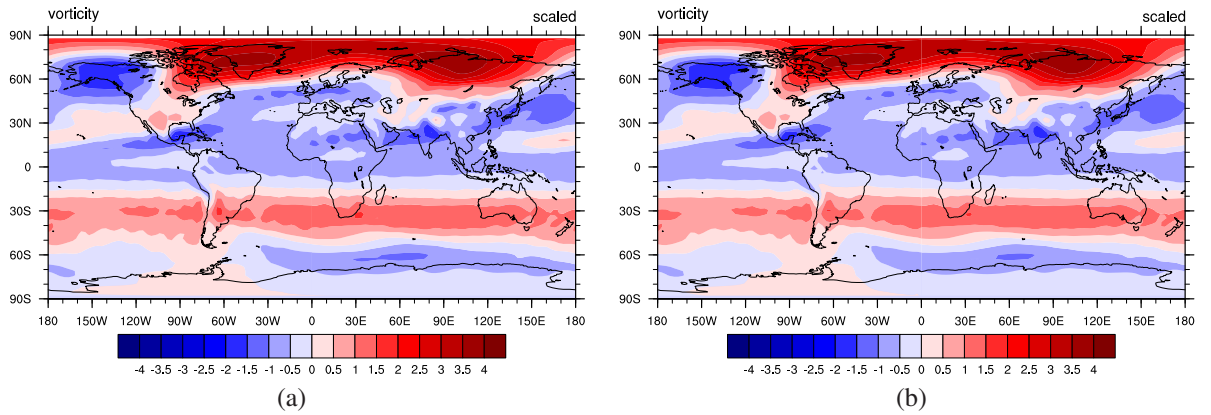

Fig. 3. NCL visualization of vorticity calculated from the same U and V field by (a) the original NCL routine and (b) the ParGAL routine

The native NCL vorticity function uses spherical harmonic analysis [23], which requires global data on a structured spherical grid and provides an accurate representation of vorticity. However, it is not applicable to data on limited domains or on unstructured grids and is not easily parallelizable. In contrast, the algorithm in ParGAL has been developed by using a finite-element approach that is highly parallelizable, works equally well on global and limited domains, and is easily extensible to unstructured grids. In the ParGAL approach a formal L^2 projection is used to approximate the vorticity from a nodal velocity field. This method generates a simple linear system whose components are obtained by integrating over cells, thereby eliminating the pole singularity in the case when nodes are located at the poles. The implementation of the algorithm uses Intrepid to provide basis function definitions, numerical quadrature rules, and cell-based numerical operations. The linear system is solved by using an iterative solver from the AztecOO package [24] and a multilevel preconditioner from the ML package [25], both part of the Trilinos framework [12].

Figure 3 shows that the two algorithms produce nearly identical results visually. To compare performance, we measured the time taken to compute vorticity with both the original spherical harmonic NCL function and the ParGAL function as called by our parallel version of NCL, ParNCL. We used a four time-steps from an atmospheric general circulation model with a horizontal grid of 768x1152 points and 26 vertical levels. The two-dimensional vorticity field is calculated separately on each level with one call to the function.

Performance results were obtained from the *Fusion* cluster at Argonne National Laboratory. Each compute node in the cluster has two Nehalem 2.6 GHz Pentium Xeon processors (8 cores) and 36 GB of memory and uses the InfiniBand QDR network for communication. We compared NCL version 6.0.0 Beta with ParNCL's beta release. Figure 4 shows that the ParGAL/ParNCL vorticity algorithm is already faster than NCL even at one processor. This timing includes all file I/O. The flattening of the curve after 32 processors comes from the parallel I/O, which we have not optimized. The algorithm without I/O scales nearly perfectly (not

shown). While 32 processors is small in the petascale age, introducing distributed-memory algorithms for regular use on any number of processors will be a paradigm shift for most Earth science analysis workflows. Even at only 16 processors, however, the total time is already reduced by over an order of magnitude to a time that is no longer prohibitive for exploratory analysis. The Intrepid-based computation of vorticity in ParNCL is more efficient than the algorithm called by NCL because it does not require projection into global spherical harmonics; even though the Intrepid-based method requires solution of a global system, it is still efficient, and parallelizable, because of the use of modern solvers with multigrid capability.

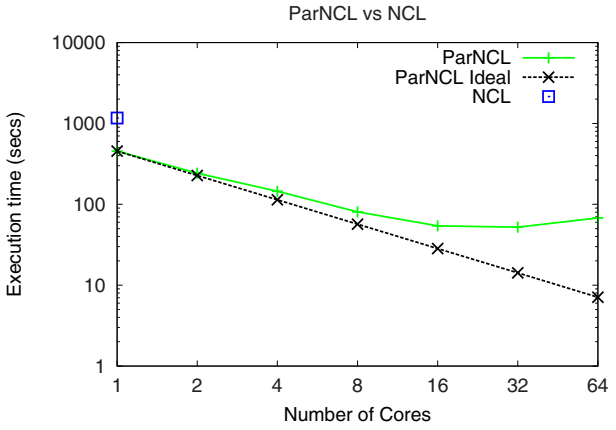

Fig. 4. Total Execution time for reading 4 timesteps of data and calculating the vorticity field on each level of a 768x1152x26 grid vs. number of cores

6. Conclusion

Postprocessing analysis of petascale model output is a crucial component of the scientific process in the Earth sciences. ParGAL is a library for performing many analysis functions that introduces both the ability to employ data parallelism and operate on both structured and unstructured grids. To build ParGAL, we have leveraged several well-engineered software libraries that provide key capabilities in the area of parallel mesh and mesh-based data, parallel I/O, and mesh-based discretizations. Using libraries for this purpose not only simplifies construction of an integrated data analysis capability but also makes the postprocessing operations similar (both mathematically and algorithmically) to the operations in the original simulation. Our early results comparing performance with that of a well-established visualization and analysis package are encouraging. ParGAL provides high-level functions that hide the details of the distributed-memory parallelism from the end user, potentially allowing the familiar script-based analysis approach to scale in parallel. NCL has over 300 built-in functions, and our plan is to implement data-parallel versions of the most widely used ones in ParNCL. We will also add the ability to work with additional file formats and grid types. We believe ParGAL/ParNCL and its core set of functions will significantly improve the ability of Earth scientists to gain knowledge from their large datasets.

Acknowledgements

This work is part of the Parallel Analysis Tools and New Visualization Techniques for Ultra-Large Climate Data Sets (ParVis) project supported by the Earth System Modeling Program of the Office of Biological and Environmental Research of the U.S. Department of Energy's Office of Science under contract DE-AC02-06CH11357. The project is co-sponsored by the U.S. National Science Foundation via contributions from the National Center for Atmospheric Research, Boulder, CO. We gratefully acknowledge the computing resources provided on "Fusion," a 320-node computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] J. M. Dennis, M. Vertenstein, P. H. Worley, A. A. Mirin, A. P. Craig, R. Jacob, S. Mickelson, Computational performance of the ultra-high resolution capability in the community earth system model, Int. J. High Perf. Comp. Appl. 26 (5) (2012) 5–16.

[2] W. Washington, et al., Scientific grand challenges: Challenges in climate change science and the role of computing and the extreme scale, http://science.energy.gov/~/media/ber/pdf/Climate_report.pdf (2008).
[3] J. P. Holdren, E. Lander, H. Varmus, Report to the president and congress: Designing a digital future: federally funded research and development in networking and information technology, http://www.nitrd.gov/pcast-2010/report/nitrd-program/pcast-nitrd-report-2010.pdf (2010).
[4] N. Samatova, M. Branstetter, A. Ganguly, R. Hettich, S. Khan, G. Kora, J. Li, X. Ma, C. Pan, A. Shoshani, S. Yoginath, High performance statistical computing with Parallel R: Applications to biology and climate modeling, in: Journal of Physics: Conference Series SciDAC 2006, Vol. 46, 2006, pp. 505–509.
[5] V. Vishwanath, M. Hereld, M. E. Papka, Toward simulation-time data analysis and I/O acceleration on leadership-class systems, in: 2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV), 2011, pp. 9–14.
[6] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H.-W. Shen, T.-Y. Lee, A. Chaudhuri, Scalable parallel building blocks for custom data analysis, in: 2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV), 2011, pp. 105–112.
[7] T. J. Tautges, R. Meyers, K. Merkley, C. Stimpson, C. Ernst, MOAB: a Mesh-Oriented database, SAND2004-1592, Sandia National Laboratories, report (Apr. 2004).
[8] T. J. Tautges, MOAB wiki, http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB.
URL http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB
[9] Hierarchical data format version 5, http://www.hdfgroup.org/HDF5 (Sep. 2011).
URL http://www.hdfgroup.org/HDF5
[10] T. J. Tautges, J. Kraftcheck, N. Bertram, V. Sachdeva, J. Materlein, Mesh interface resolution and ghost exchange in a parallel mesh representation, in: Workshop on Large-Scale Parallel Processing, held at the IEEE International Parallel and Distributed Processing Symposium, IEEE, Shanghai, China, 2012.
[11] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, C. Vaughan, Zoltan data management services for parallel dynamic applications, Computing in Science and Engineering 4 (2) (2002) 9097.
[12] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, K. S. Stanley, An overview of the trilinos project, ACM Trans. Math. Softw. 31 (3) (2005) 397–423. doi:http://doi.acm.org/10.1145/1089014.1089021.
[13] P. Bochev, H. Edwards, R. Kirby, K. Peterson, D. Ridzal, Solving pdes with intrepid, Scientific Programming 20(2) (2012) 151–181.
[14] P. Bochev, M. Hyman, Principles of mimetic discretizations, in: D. N. Arnold, P. Bochev, R. Lehoucq, R. Nicolaides, M. Shashkov (Eds.), Compatible Discretizations, Proceedings of IMA Hot Topics Workshop on Compatible Discretizations, Vol. IMA 142, Springer Verlag, 2006, pp. 89–120.
[15] P. B. Bochev, K. Peterson, C. M. Siefert, Analysis and computation of compatible least-squares methods for div-curl equations, SIAM Journal on Numerical Analysis 49 (1) (2011) 159–181. doi:10.1137/090772095.
URL http://link.aip.org/link/?SNA/49/159/1
[16] P. Bochev, K. Peterson, X. Gao, A new control volume finite element method for the stable and accurate solution of the drift-diffusion equations on general unstructured grids, Comput. Methods Appl. Mech. Engrg. 254 (2013) 126–145.
[17] R. Rew, G. Davis, Netcdf: an interface for scientific data access, Computer Graphics and Applications, IEEE 10 (4) (1990) 76 –82. doi:10.1109/38.56302.
[18] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, Parallel netCDF: A high-performance scientific I/O interface, in: Proceedings of SC2003: High Performance Networking and Computing, SC '03, IEEE Computer Society Press, Phoenix, AZ, 2003, pp. 39–. doi:10.1145/1048935.1050189.
URL http://www.sc-conference.org/sc2003/paperpdfs/pap258.pdf
[19] MPI-2: Extensions to the message-passing interface, The MPI Forum (July 1997).
URL http://www.mpi-forum.org/docs/docs.html
[20] B. Eaton, J. Gregory, B. Drach, K. Taylor, S. Hankin, J. Caron, R. Signell, P. Bentley, G. Rappa, H. Höck, et al., Netcdf climate and forecast (cf) metadata conventions (2003).
[21] B. Stroustrup, The C++ Programming Language, 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
[22] NCL – NCAR Command Language (version 6.0.0) [software]. Boulder, Colorado: UCAR/NCAR/CISL/VETS, http://dx.doi.org/10.5065/D6WD3XH5 (2012).
[23] J. C. Adams, P. N. Swarztrauber, Spherpack 3.0: A model development facility, Monthly Weather Review 127 (1999) 1872–1878.
[24] M. A. Heroux, AztecOO user guide, Tech. Rep. SAND2004-3796, Sandia National Laboratories (2007).
[25] M. Gee, C. Siefert, J. Hu, R. Tuminaro, M. Sala, ML 5.0 smoothed aggregation user's guide, Tech. Rep. SAND2006-2649, Sandia National Laboratories (2006).