A Distributed Python HPC Framework: ODIN, PyTrilinos, & Seamless

Phase II final report submitted to the DOE SBIR program by:

Enthought, Inc.

Award Number:

DE-SC0007699

Principal Investigator:

Robert D. Grant, Ph.D.

This report is approved for unlimited distribution.

515 Congress Avenue, Suite 2100, Austin, TX 78701

1 Executive Summary

The technical effectiveness and economic feasibility of the methods or techniques investigated or demonstrated: Under this grant, three significant software packages were developed or improved, all with the goal of improving the ease-of-use of HPC libraries.

The first component is a Python package, named DistArray (originally named Odin), that provides a high-level interface to distributed array computing. This interface is based on the popular and widely used NumPy package and is integrated with the IPython project for enhanced interactive parallel distributed computing.

The second Python package is the Distributed Array Protocol (DAP) that enables separate distributed array libraries to share arrays efficiently without copying or sending messages. If a distributed array library supports the DAP, it is then automatically able to communicate with any other library that also supports the protocol. This protocol allows DistArray to communicate with the Trilinos library via PyTrilinos, which was also enhanced during ths project.

A third package, PyTrilinos, was extended to support distributed structured arrays (in addition to the unstructured arrays of its original design), allow more flexible distributed arrays (i.e., the restriction to double precision data was lifted), and implement the DAP. DAP support includes both exporting the protocol so that external packages can use distributed Trilinos data structures, and importing the protocol so that PyTrilinos can work with distributed data from external packages.

How the project is otherwise of benefit to the public: The original goal of the project was to make parallel hardware easier to use and to make HPC more accessible by providing a distributed HPC framework based in the scientific Python ecosystem. The three components developed for this project–DistArray, the Distributed Array Protocol, and Trilinos improvements–combine to provide a NumPy like interface to distributed array computing. This allows non-expert engineers who are familiar with NumPy to leverage the performance and scalability of Trilinos.

2 Goal Comparison

2.1 Accomplished Goals, Objectives, and Milestones

Distributed Array Protocol The distributed array protocol was defined and implemented in both DistArray and PyTrilinos. As a community protocol, input was solicited and obtained from parties outside the project, including developers of IPython, Global Arrays in NumPy (GAiN), and PETSc.
`https://github.com/enthought/distributed-array-protocol`

Trilinos Multidimensional Capabilities The Domi package was developed to provide dense multidimensional arrays which, among other uses, facilitates more direct interaction between Trilinos, NumPy, and DistArray.
`https://trilinos.org/oldsite/release_notes-12.0.html`

PyTrilinos interoperability with NumPy/DistArray We provided a conduit using the Distributed Array Protocol for PyTrilinos to share distributed arrays with NumPy and DistArray and for DistArray to do the same with PyTrilinos (using Domi behind the scenes). We verified that both projects can share arrays via the protocol.

PyTrilinos enhancements We made general improvements to PyTrilinos to make it easier to use and more "Pythonic." This includes extended Tpetra wrappers, updated Epetra wrappers , and improved documentation and usability.
`https://trilinos.org/packages/pytrilinos/`

DistArray minimal functionality We allow the creation of distributed arrays with user-specifiable domain decomposition over one or several dimensions with blocked data distribution. Array indexing and assignment are supported. All distributed ufuncs are implemented for use with identically distributed arrays. A functional communication layer was implemented for communication between global and local nodes. `https://github.com/enthought/distarray`

DistArray local API, minimal functionality An API was implemented for distributed array argument passing to local nodes. A root node can push user specified code to local nodes, and local nodes can run user code with distributed array sections as arguments. Local user code can communicate between local nodes with the communication API between local nodes. `https://github.com/enthought/distarray/tree/master/distarray/localapi`

DistArray global API, expanded functionality We implemented some communication-heavy distributed NumPy methods, including distributed array reduction functions. We expanded array layout support and expanded user control over array domain distribution. We performed performance profiling of the communication backend.
`https://github.com/enthought/distarray/tree/master/distarray/globalapi`

2.2 Milestones not Accomplished

DistArray global API, expanded functionality We did not implement distributed array median or sorting. It is possible to implement these communication-heavy operations, but doing so would have been at the expense of other core functionality that had greater priority.

Seamless wrapping C We do not support Python code importing C code automatically and calling into the imported library. All Seamless milestones were removed from the project midway to ensure completion of DistArray core milestones and tasks.

Seamless wrapping C++ We do not support Python code importing C++ code automatically, subclassing C++ classes, and generally interoperating with the imported C++ library. All Seamless milestones were removed from the project midway to ensure completion of DistArray core milestones and tasks.

Seamless Python bytecode compilation We do not support compiling Python code via the Seamless compilation infrastructure. All Seamless milestones were removed from the project midway to ensure completion of DistArray core milestones and tasks.

PyTrilinos Wrappers for Second-Generation Packages It was decided that a preferable approach to wrapping multiple linear-solver related packages, would be to wrap the single Trilinos package Stratimikos instead. This package provides access to all Trilinos linear solver packages via a nested parameter list, and would provide the best interface to linear solvers within PyTrilinos. Unfortunately, the status of Stratimikos was not yet conducive to wrappers at the point in time when the project would have required it. Note that for nonlinear solvers, the PyTrilinos.NOX module for nonlinear solvers was expanded to support PETSc linear solvers, and the PyTrilinos.LOCA module was re-implemented after a long period of being disabled.

3 Project Activities Summary

3.1 Original Design

Pure Python prototype The original design began with a pure Python prototype, called ODIN. It demonstrated feasibility and provided an opportunity to explore design questions, including usage and interaction of ZeroMQ, IPython, Cython, and MPI4Py.

Python + ZeroMQ + MPI4Py + Cython The primary packages originally intended to be used for the project were ZeroMQ and its Python bindings in PyZMQ; MPI and its Python interface with MPI4Py; and Cython for compiling and speeding up performance-critical Python code.

Fault tolerance Python users expect to be able to safely explore a library or package, and to this end Python supports both managed memory via reference counting and garbage collection as well as exceptions. Originally the intent was to explore ways to enable robust exception handling in a distributed context via fault tolerant MPI libraries, etc.

Structured data Trilinos package Trilinos was originally designed for unstructured data (picture fields that live on an unstructured mesh). NumPy, and the planned distributed version, supports structured data. This prompted a new Trilinos package, Domi, that supports structured, distributed data, and that can be converted to traditional unstructured Trilinos data structures without copying.

Python wrappers for Tpetra Tpetra is a Trilinos package that provides classes for distribution maps, vectors, multi-vectors, and matrices. It is more complicated than previously wrapped Trilinos packages in that its scalar and ordinal types are specified by C++ templates. The goal was to provide Python wrappers to this package that handle scalar types in the same manner that NumPy does.

PyTrilinos interoperability with the DistArray Protocol Vector and multi-vector classes within Epetra, Tpetra, and Domi are to export the DistArray Protocol, so that other packages can convert them to a native data structure without copying. Also, the Python versions of function and method arguments that accept Epetra, Tpetra, or Domi vectors or multi-vectors are to import any Python object that supports the DistArray Protocol.

3.2 Departures and Course Corrections

IPython Parallel and extending the original DistArray Early on in the project, we evaluated the original DistArray project by Brian Granger. Its development had lapsed, and its implementation and goals were aligned with the same for this project. We decided to use the original DistArray as our starting point to accelerate development. We discussed the decision with B. Granger; he and Min Ragan-Kelly were supportive and provided input.

Scope reduction and reallocation of time from Seamless to DistArray development With personnel changes, the developers best suited for Seamless development for this project were no longer available. We decided to reduce the scope of the project and reallocate the time devoted to the Seamless tasks to DistArray and the Distributed Array Protocol.

DistArray MPI-only communication option for performance improvements After developing a minimal viable DistArray version, performance profiling revealed the IPython parallel based ZeroMQ communication to be the dominant bottleneck both in terms of latency and throughput. We decided to add an MPI-only based communication backend to improve performance. This backend consistently improved latency by a factor of 10, and throughput by a factor of 3-5.

PyTrilinos Interoperability with MPI4Py Certain details of the implementation of DistArray and the desire to be interoperable with PyTrilinos, forced PyTrilinos to move from supporting MPI_COMM_WORLD only to supporting sub-communicators. To maximize interoperability with established HPC Python packages, PyTrilinos now uses the sub-communicators from MPI4Py to enable this capability (rather than wrap its own).

PyTrilinos Interoperability with PETSc The PyTrilinos wrappers for the NOX non-linear solver package were expanded to allow users to specify a PETSc linear solver at the core of a Newton algorithm.

4 Products Developed

4.1 Publications

Parallel Processing in Python Minisymposium at SIAM PP 2014

PyCon 2015 Poster Presentation on DistArray K.W. Smith, R.D. Grant, and W.F. Spotz, *DistArray: Distributed Numpy.* Poster presented at PyCon 2015, Montreal, Quebec, Canada, April 2015.

SciPy 2015 Presentation on DistArray R.D. Grant, *DistArray: Distributed Arrays for Python.* Presentation at SciPy 2015, Austin, TX, July 2015.

Trilinos Users Group 2015 Presentation W.F. Spotz, *Cool Stuff You Can Do with PyTrilinos, DistArray, and Jupyter.* Presentation at the 2015 Trilinos Users Group meeting, Albuquerque, NM, October, 2015.

4.2 Project Web Sites and Related Web Resources

DistArray Web Site `http://distarray.readthedocs.org/en/latest/`

DistArray Source Code Repository `https://github.com/enthought/distarray`

Distributed Array Protocol Web Site `http://distributed-array-protocol.readthedocs.org/en/latest/`

Distributed Array Protocol Source Code Repository `https://github.com/enthought/distributed-array-protocol`

Trilinos Web Page `https://trilinos.org/`

PyTrilinos Web Page `https://trilinos.org/packages/pytrilinos`

Domi Web Page `https://trilinos.org/packages/domi`

4.3 Networks and Collaborations Fostered

The Distributed Array Protocol in particular has fostered significant collaboration between the project participants and others interested in computing with distributed arrays.

Jeff Daily, developer of GAiN (Global Arrays in NumPy) Jeff Daily is a scientist at the Pacific Northwest National Laboratory, a key developer of the Global Arrays Toolkit, and is responsible for the Python support for Global Arrays. He participated in the design of the Distributed Array Protocol.

Development team, IPython/Jupyter Brian Granger is a major contributor to the IPython and Jupyter projects and is the original author the DistArray project, which we have taken as an starting point for our work. Min Ragan-Kelly is also a major contributor to these projects and is heavily involved in the IPython Parallel subproject, which DistArray relies upon. Both have provided input into the design of DistArray and the Distributed Array Protocol.

Lisandro Dalcin, lead developer of PETSc4Py We are currently in discussions with Lisandro Dalcin regarding support for the Distributed Array Protocol in PETSc4Py and possible extensions to the protocol in a future version.

4.4 Technologies/Techniques

The Distributed Array Protocol The Distributed Array Protocol (DAP) is a process-local protocol that allows two subscribers, called the "producer" and the "consumer" or the "exporter" and the "importer", to communicate the essential data and meta-data necessary to share a distributed-memory array between them. This allows two independently developed components to access, modify, and update a distributed array without copying. The protocol formalizes the metadata and buffers involved in the transfer, allowing several distributed array projects to collaborate, facilitating interoperability. By not copying the underlying array data, the protocol allows for efficient sharing of array data.

The DAP is intended to build on the concepts and implementation of the existing Python Revised Buffer Protocol (PEP-3118, `https://www.python.org/dev/peps/pep-3118/`). The protocol uses PEP-3118 buffers and subscribing Python objects such as memoryviews and NumPy arrays as components.

This initial version of the DAP is defined for the Python language. Future versions of this protocol may provide definitions in other languages.

Major use-cases supported by the protocol include:

- Sharing large amounts of array data without copying.
- Block, cyclic, and block-cyclic distributions for structured arrays.
- Padded block-distributed arrays for finite differencing applications.
- Unstructured distributions for arbitrary mappings between global indices and local data.
- Multi-dimensional arrays.
- Different distributions for each array dimension.
- Dense (structured) and sparse (unstructured) arrays.
- Compatibility with array views and slices.

The DAP is used in this work to facilitate distributed array transfer between DistArray and PyTrilinos, but we have worked with several others in its design, and we expect it to prove useful for collaborating with other projects as well.

4.5 Software

DistArray DistArray aims to bring the ease-of-use of the widely used NumPy package to data-parallel high-performance computing. It provides distributed multi-dimensional NumPy arrays, distributed ufuncs, and distributed IO capabilities. It can efficiently interoperate with external distributed libraries like Trilinos. DistArray works with NumPy and builds on top of it in a flexible and natural way.

DistArray

- supports NumPy-like slicing, reductions, and ufuncs on distributed multidimensional arrays;
- has a client-engine process design — data resides on the worker processes, commands are initiated from master;
- allows full control over what is executed on the worker processes and integrates transparently with the master process;
- allows direct communication between workers, bypassing the master process for scalability;
- integrates with IPython.parallel for interactive creation and exploration of distributed data;
- supports distributed ufuncs (currently without broadcasting);
- builds on and leverages MPI via MPI4Py in a transparent and user-friendly way;
- has basic support for unstructured arrays;
- supports user-controllable array distributions across workers (block, cyclic, block-cyclic, and unstructured) on a per-axis basis;
- has a straightforward API to control how an array is distributed;
- has basic plotting support for visualization of array distributions;
- separates the array's distribution from the array's data — useful for slicing, reductions, redistribution, broadcasting, and other operations;
- supports redistribution for block-distributed (and non-distributed) DistArrays;
- implements distributed random arrays;
- supports `.npy`-like flat-file IO and hdf5 parallel IO (via h5py); leverages MPI-based IO parallelism in an easy-to-use and transparent way;
- supports the distributed array protocol, which allows independently developed parallel libraries to share distributed arrays without copying, analogous to the PEP-3118 new buffer protocol;
- is well tested on both OS X and Linux; and
- is well documented on the web and with examples in the source tree.

PyTrilinos PyTrilinos provides Python wrappers to selected packages within the Trilinos project. Trilinos provides a wide variety of packages that support high-performance modeling and simulation. PyTrilinos tends to support those packages related to linear solvers, preconditioners, nonlinear solvers, continuation algorithms, and eigensolvers. Upgrades to PyTrilinos as a result of this project include

- Python wrappers to Tpetra, which provides unstructured vectors, multi-vectors, and matrices with user-specifiable scalar data types, much like NumPy;
- inclusion of a new Trilinos package named Domi that supports distributed multi-dimensional data structures that are compatible with existing Trilinos data structures;
- Python wrappers that provide an interface to Domi, including user-specifiable scalar data types, as in NumPy;
- Epetra, Tpetra, and Domi vectors and multi-vectors now export the DistArray Protocol, so that other Python packages can convert these data structures to native types without copying;
- PyTrilinos function and method arguments that expect Epetra, Tpetra, or Domi vector or multi-vectors that import those Python objects that support the DistArray Protocol;
- a re-enabling of the LOCA continuation algorithm package;
- enabling the NOX nonlinear solver package to use PETSc linear solvers at the core of NOX's Newton algorithms;
- improved PyTrilinos usability, memory management, and robustness;
- improved PyTrilinos documentation.