Using the Global Arrays Toolkit to Reimplement NumPy for Distributed Computation

Jeff Daily, Pacific Northwest National Laboratory
jeff.daily@pnnl.gov
Robert R. Lewis, Washington State University
bobl@tricity.wsu.edu
Motivation

▸ Lots of NumPy applications
  ▪ NumPy (and Python) are for the most part single-threaded
  ▪ Resources underutilized
    - Computers have multiple cores
    - Academic/business clusters are common

▸ Lots of parallel libraries or programming languages
  ▪ Message Passing Interface (MPI), Global Arrays (GA), X10, Co-Array Fortran, OpenMP, Unified Parallel C, Chapel, Titianium, Cilk
  ▪ Can we transparently parallelize NumPy?
Background – Parallel Programming

- Single Program, Multiple Data (SPMD)
  - Each process runs the same copy of the program
  - Different branches of code run by different threads

```python
if my_id == 0:
    foo()
else:
    bar()
```
Background – Message Passing Interface

- Each process assigned a rank starting from 0
- Excellent Python bindings – mpi4py
- Two models of communication
  - Two-sided i.e. message passing (MPI-1 standard)
  - One-sided (MPI-2 standard)

```python
if MPI.COMM_WORLD.rank == 0:
    foo()
else:
    bar()
```
Background – Communication Models

Message Passing:
Message requires cooperation on both sides. The processor sending the message (P1) and the processor receiving the message (P0) must both participate.

One-sided Communication:
Once message is initiated on sending processor (P1) the sending processor can continue computation. Receiving processor (P0) is not involved. Data is copied directly from switch into memory on P0.
Background – Global Arrays

Physically distributed data

- Distributed dense arrays that can be accessed through a shared memory-like style
- single, shared data structure/global indexing
  - e.g., `ga.get(a, (3, 2))` rather than `buf[6]` on process 1
- Local array portions can be `ga.access()`’d

Global Address Space
Remote Data Access in GA vs MPI

Message Passing:

identify size and location of data blocks

loop over processors:
  if (me = P_N) then
    pack data in local message buffer
    send block of data to message buffer on P0
  else if (me = P0) then
    receive block of data from P_N in message buffer
    unpack data from message buffer to local buffer
  endif
end loop

copy local data on P0 to local buffer

Global Arrays:

```
buf=ga.get(g_a, lo=None, hi=None, buffer=None)
```

Global Array handle

Global upper and lower indices of data patch

Local ndarray buffer
Background – Global Arrays

- Shared data model in context of distributed dense arrays
- Much simpler than message-passing for many applications
- Complete environment for parallel code development
- Compatible with MPI
- Data locality control similar to distributed memory/message passing model
- Extensible
- Scalable
Previous Work to Parallelize NumPy

- Star-P
- Global Arrays Meets MATLAB (yes, it’s not NumPy, but…)
- IPython
- gpupy
- Co-Array Python
Design for Global Arrays in NumPy (GAIaN)

- All documented NumPy functions are collective
  - GAIaN programs run in SPMD fashion
- Not all arrays should be distributed
  - GAIaN operations should allow mixed NumPy/GAIaN inputs
- Reuse as much of NumPy as possible (obviously)
- Distributed nature of arrays should be transparent to user
- Use owner-computes rule to attempt data locality optimizations
Why Subclassing `numpy.ndarray` Fails

The hooks:
- `__new__()`,
- `__array_prepare__()`
- `__array_finalize__()`
- `__array_priority__`

First hook `__array_prepare__()` is called *after the output array has been created*
- No means of intercepting array creation
- Array is allocated on each process – not distributed
The `gain.ndarray` in a Nutshell

- Global shape and $P$ local shapes
- Memory allocated from Global Arrays library, wrapped in local `numpy.ndarray`
- The memory distribution is static
- Views and array operations query the current `global_slice`
Example: Slice Arithmetic

Observation: In both cases shown here, Array \( b \) could be created either using the standard notation (top) or the “canonical” form (bottom)
Example: Binary Ufunc

Owner-computes rule means output array owner does the work

- `ga.access()` other input array portions since all distributions and shapes are the same
- *call original NumPy ufunc on the pieces*
Example: Binary Ufunc with Sliced Arrays

Owner-computes rule means output array owner does the work

- `ga.get()` other input array portions since arrays not aligned
- call original NumPy ufunc
Example: Binary Ufunc

- Broadcasting works too
- Not all arrays are distributed
How to Use GAiN

Ideally, change one line in your script:

```python
# import numpy
import ga.gain as numpy
```

Run using the MPI process manager:

```
$ mpiexec -np 4 python script.py
```
Live Demo: laplace.py

2D Laplace equation using an iterative finite difference scheme (four point averaging, Gauss-Seidel or Gauss-Jordan).

I’ll now show you how to use GAiN

(This is not the “pretty pictures” part of the presentation -- there’s nothing pretty about raw computation.)
laplace.py Again, but Bigger

laplace.py Strong Scaling

Time (s)

N=10,000
N=100,000

Cores

Proudly Operated by Battelle Since 1965
GAiN is Not Complete (yet)

► What’s finished:
  ■ Ufuncs (all, but not reduceat or outer)
  ■ ndarray (mostly)
  ■ flatiter
  ■ *numpy dtypes are reused!*
  ■ Various array creation and other functions:
    ● zeros, zeros_like, ones, ones_like, empty, empty_like
    ● eye, identity, fromfunction, arange, linspace, logspace
    ● dot, diag, clip, asarray

► Everything else doesn’t exist, including order=''
► GAiN is here to stay – it’s official supported by the GA project (me!)
Thanks! Time for Questions

Using the Global Arrays Toolkit to Reimplement NumPy for Distributed Computation

Jeff Daily, Pacific Northwest National Laboratory
jeff.daily@pnnl.gov

Robert R. Lewis, Washington State University
bobl@tricity.wsu.edu

Where to get the code until pnl.gov domain is restored:

Where to get the code, usually:
https://svn.pnl.gov/svn/hpctools/trunk/qa

Website (documentation, download releases, etc):
http://www.emsl.pnl.gov/docs/global