Using the Global Arrays Toolkit to Reimplement Python's NumPy for Distributed Computation

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

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

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

Outline

- Python programming language
  - Overview
  - NumPy – N-dimensional arrays and more
  - Cython – C extensions for Python
  - Python for high performance computing
- Global Arrays Toolkit
  - Overview
  - Python bindings for Global Arrays
  - Global Arrays in NumPy (GAiN)

Python

- General purpose language
- Machine-independent, bytecode interpreted
- Object-oriented
- Rapid application development
- Extensible with C, C++, Fortran, Python, Cython
- Introspective
- Dynamically typed i.e. late binding
- Small language spec; large standard library
- Large and active community

Python – Extensible

- User-defined classes can look and behave like Python's built-in types
  - Implement as much or as little as needed
  - For example, `def __len__()` called by built-in `len()`
- "duck typing"
  - "If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck."
  - ```
    def foo(duck):
        duck.quack()
    ```
    - Function foo takes any duck-like object
    - These "ducks" need not derive from same class hierarchy
    ```

Python – High Level Language Features

- Slicing e.g. `a[4],b[2:7],c[:-1],d[1:9:2]`
  - Python indexing is half-open [0,n)
  - Returns a copy
  - Possibly multidimensional e.g. `e[1,2,3:10]`
- List comprehensions
  - `a = [(i**2,i**3) for i in range(10)]`
- Generators
- OO – modules, classes, methods, functions

Python – The Cons

- "Slow" – every line is interpreted, even in a loop
- Global Interpreter Lock aka the GIL
  - Python alone essentially can't utilize multiple CPUs (and frankly doesn't want to)
  - Only an issue for Python threads executing Python bytecode (effectively serial in that case)
  - Python threads on multicore CPU worse than single core
- Simple solutions:
  - Interpret less
  - Avoid the GIL i.e. wrap threaded code in C/C++
  - Stackless Python

Outline

- Python programming language
  - Overview
  - **NumPy – N-dimensional arrays and more**
  - Cython – C extensions for Python
  - Python for high performance computing
- Global Arrays Toolkit
  - Overview
  - Python bindings for Global Arrays
  - Global Arrays in NumPy (GAiN)

NumPy – N-Dimensional Arrays and More

- Primary contribution: the `ndarray` class
  - Contiguous memory segment, either C or Fortran order
  - Metadata: shape, strides, pointer to start
- Slicing returns “views”, not copies
- Universal Functions aka ufuncs
  - Arithmetic, C math library, comparison
    - `c = a + b`
    - `numpy.sub(a, b, out=c)`
  - Special methods reduce, accumulate
    - `b = numpy.add.reduce(a)`

NumPy – Broadcasting

- If a.ndim != b.ndim, prepend 1's to shape of array with fewer dimensions
- Shapes are compatible if for each dim they either match or one of them is equal to 1
  - (3,4,5) and (2,3,4,1) work
  - (5,6) and (5,2) don't work
- Elegant
  - Add vector to rows or columns of matrix
  - Add scalar element-wise

Outline

- Python programming language
  - Overview
  - NumPy – N-dimensional arrays and more
  - **Cython – C extensions for Python**
  - Python for high performance computing
- Global Arrays Toolkit
  - Overview
  - Python bindings for Global Arrays
  - Global Arrays in NumPy (GAiN)

Cython – C Extensions for Python

- A language
  - Start with Python
  - Add static typing
- A compiler
  - Compiles to C code which utilizes Python's C API
  - Optimized for use with NumPy
- Call any external libraries – it's C after all

```
def foo(bar):
    baz = 6.0
    return bar+baz

def foo(float bar):
    cdef float baz = 6.0
    return bar+baz
```

Outline

- Python programming language
  - Overview
  - NumPy – N-dimensional arrays and more
  - Cython – C extensions for Python
  - **Python for high performance computing**
- Global Arrays Toolkit
  - Overview
  - Python bindings for Global Arrays
  - Global Arrays in NumPy (GAiN)

Python for High Performance Computing

- How to make Python fast
  - Observation: most time spent in math kernels
  - Keep exception handling, IO, debugging at high level
  - Spend time on speed only where it's needed
- HPC in Python → Use many Python instances
- MPI available as mpi4py
  - Communicate arbitrary Python types (after pickling)
  - Communicate NumPy arrays of C data types

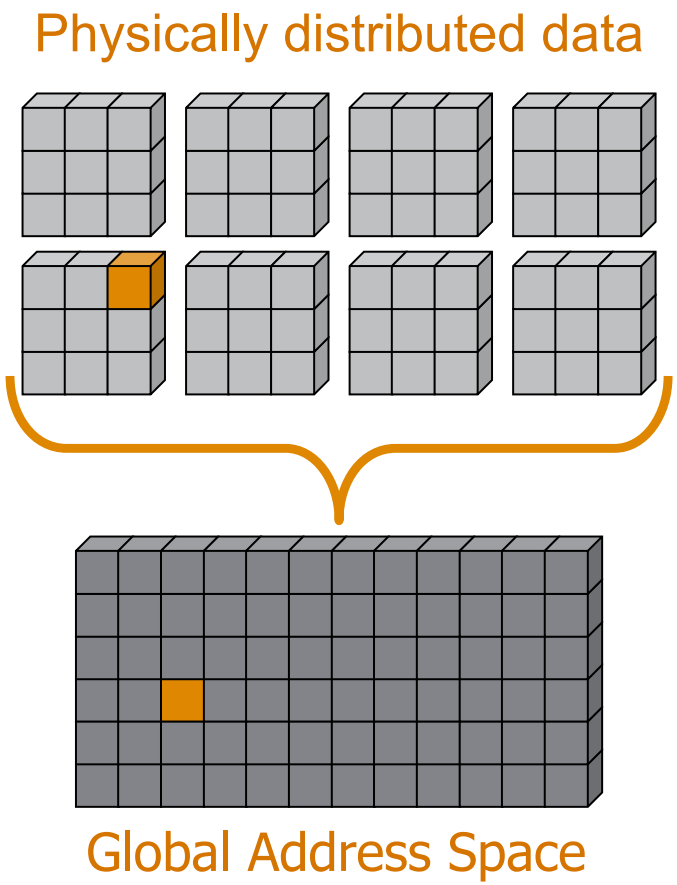
Python for HPC – The Catch

- Python relies on shared libraries
  - Unless you statically compile all extensions into python
  - Not all systems support shared libraries
  - Some systems may be missing certain system calls
- Parallel file systems do not like small files and lots of metadata traffic e.g. BGP
  - W. Scullin at ANL rewrote libdl to use MPI broadcast to load libraries
  - If not, “hello world” may take 0.5-1.5 hours to load

Outline

- Python programming language
  - Overview
  - NumPy – N-dimensional arrays and more
  - Cython – C extensions for Python
  - Python for high performance computing
- **Global Arrays Toolkit**
  - Overview
  - Python bindings for Global Arrays
  - Global Arrays in NumPy (GAiN)

Global Arrays Toolkit



- Distributed dense arrays that can be accessed through a shared memory-like style
- One-sided communication versus message passing
- Global indexing, for example:

```
buf = ga.get(handle,
      lo=[2,2], hi=[3,3])
```

rather than buf[7] on task 2

Global Arrays vs. Other Models

- Advantages:
  - Inter-operates with MPI
    - Use more convenient global-shared view for multi-dimensional arrays, but can use MPI model wherever needed
  - Data-locality and granularity control is explicit with GA's get-compute-put model, unlike the non-transparent communication overheads with other models (except MPI)
  - Library-based approach: does not rely upon smart compiler optimizations to achieve high performance
- Disadvantage:
  - Only useable for array data structures

Outline

- Python programming language
  - Overview
  - NumPy – N-dimensional arrays and more
  - Cython – C extensions for Python
  - Python for high performance computing
- Global Arrays Toolkit
  - Overview
  - **Python bindings for Global Arrays**
  - Global Arrays in NumPy (GAiN)

Python Bindings for Global Arrays

- First implementation by Robert Harrison
  - Tested with Python 1.5.2 and a precursor to NumPy
  - Implemented subset of GA 3.3
  - Not maintained after GA 3.3
  - Fundamental idea: GA buffers wrapped by NumPy ndarray
- GA v5.0.x revamped Python bindings
  - Written using Cython
  - Implements complete GA C API

GA+Python

- `ga.get(g_a, lo=None, hi=None, buffer=None)`
  - No "ld" leading dimension – returned ndarray correctly shaped
  - If buffer is None, allocate one
  - `ga.get(g_a)` returns entire array
    - `lo` defaults to zeros, `hi` defaults to array shape
  - Limitation: passed buffer must be contiguous
    - But only number of elements must match, not the shape
- `ga.access(g_a, lo=None, hi=None)`

GA+Python cont.

- `ga.put(handle, buffer, lo=None, hi=None)`
  - Buffer can be any "array-like"
    - `ga.put(handle, [i**2 for i in range(10)])`
  - Limitation: buffer must be contiguous (but can be diff. shape)
- No explicit "patch" routines
  - `GA_Copy(…)` and `NGA_Copy_patch(…)`
  - `ga.copy(g_a, g_b, alo=None, ahi=None, blo=None, bhi=None, trans=False)`

GA+Python – No explicit data types

- Single “dot” function
  - C: `GA_Ddot()`, `GA_Zdot()`, etc
  - Python: `ga.dot(…)`
- Single “gop” function
  - C: `GA_Dgop()`, `GA_Zgop()`, etc
  - Python: `ga.gop(buffer, op)`
    - Takes any “array-like”
    - Op is one of ‘+’, ‘*’, ‘min’, ‘max’, ‘absmin’, ‘absmax’
    - Or spell op out: `result = ga.gop_add([1,2,3,4])`
    - Or just use mpi4py…

Outline

- Python programming language
  - Overview
  - NumPy – N-dimensional arrays and more
  - Cython – C extensions for Python
  - Python for high performance computing
- Global Arrays Toolkit
  - Overview
  - Python bindings for Global Arrays
  - **Global Arrays in NumPy (GAiN)**

Global Arrays in NumPy (GAiN)

- Idea: NumPy for HPC using Global Arrays
- Similar idea already attempted by commercial outfit
  - "Star-P" by Interactive Supercomputing (now owned by MS)
  - Client/server model
    - Client is MATLAB or Python
    - Server is Fortran/C/C++ using MPI, ScaLAPACK, Trilinos
    - Data and task parallelism models
    - Python support later dropped – MATLAB only

GAiN Prototype

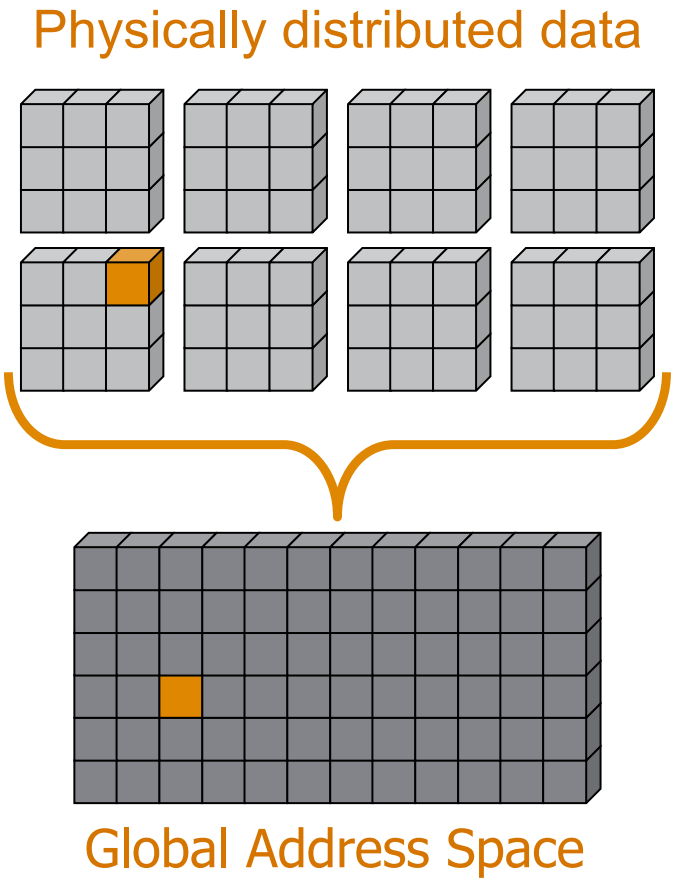
- Goals
  - Distributed arrays
  - Mixed numpy.ndarray and distributed ndarray operations
  - Drop-in replacement for NumPy
    - Implications
      - All calls are collective
      - No user-level notion of ranks
    - `import gain as numpy`
- First attempt: subclass the ndarray
  - Problem: Could not disable memory allocation by NumPy
  - Problem: Insufficient 'hooks' to intercept function calls with mixed numpy.ndarray and distributed ndarray

GAiN Prototype cont.

- Second attempt: replace NumPy
  - Must reimplement most of NumPy API
  - Must intercept all functions
- By end of MSc
  - Only implemented ufuncs and few array creation functions
  - Tested at small scale, up to 16 nodes
  - But there are approx. 300+ more functions/methods to go…

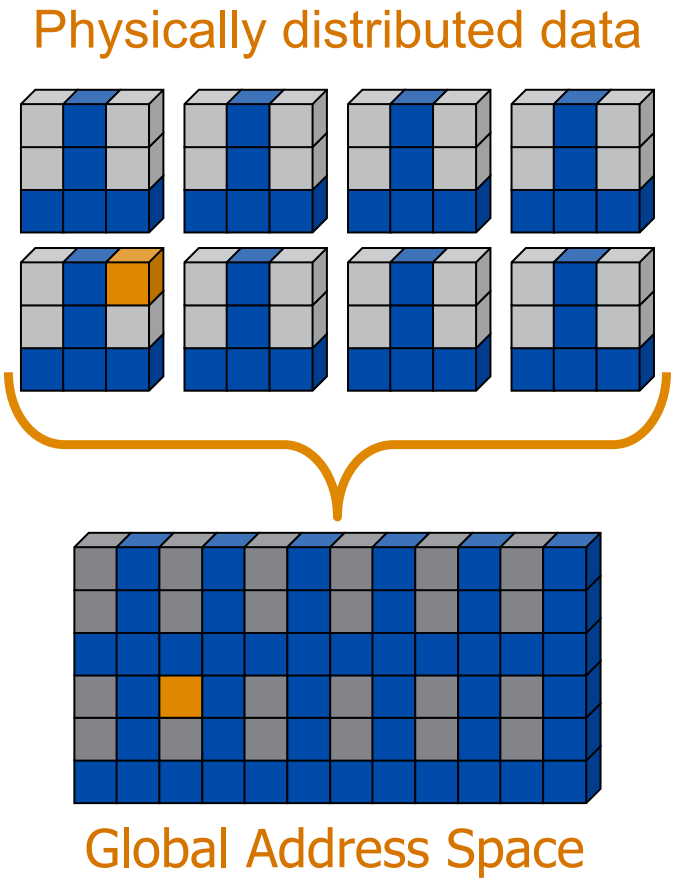
GAiN in a Nutshell

- One “global” ndarray
- p ndarray pieces
- Lots of index translation



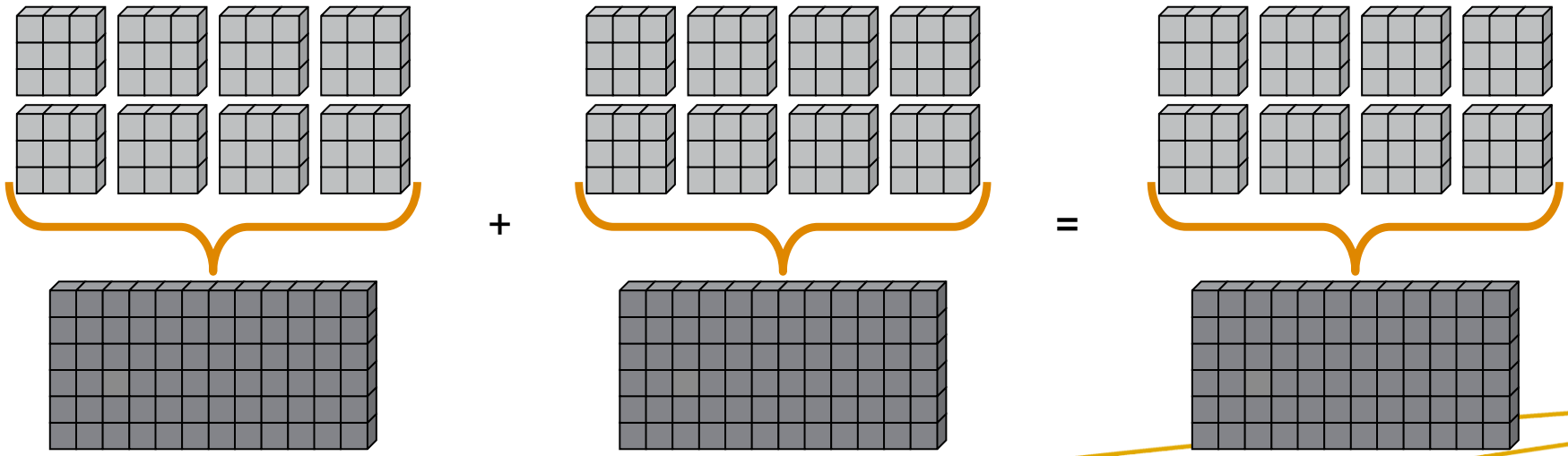
GAiN Slicing

- Shown: a.shape == (6,12)
- b = a[::3,::2]
- b.shape == (4,6)
- Each ndarray piece knows its local index space and global index space



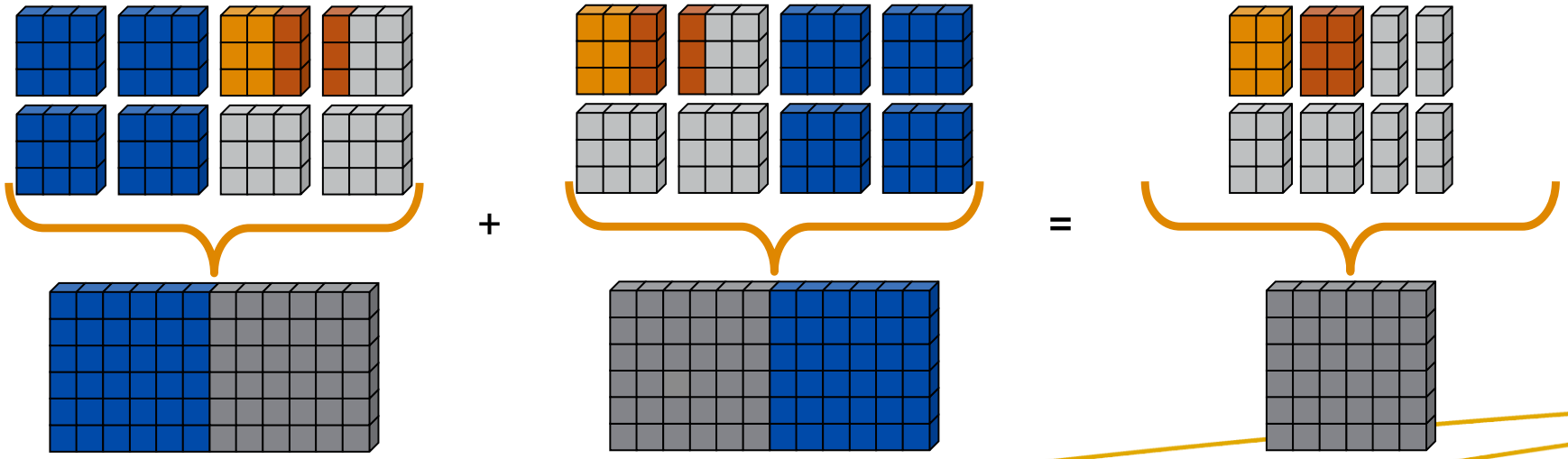
GAiN Binary Ufuncs

- `c = gain.add(a,b)`
  - If output is distributed
    - access(), get(), or slice corresponding pieces from a and b
    - ***Call original numpy ufunc on the pieces***



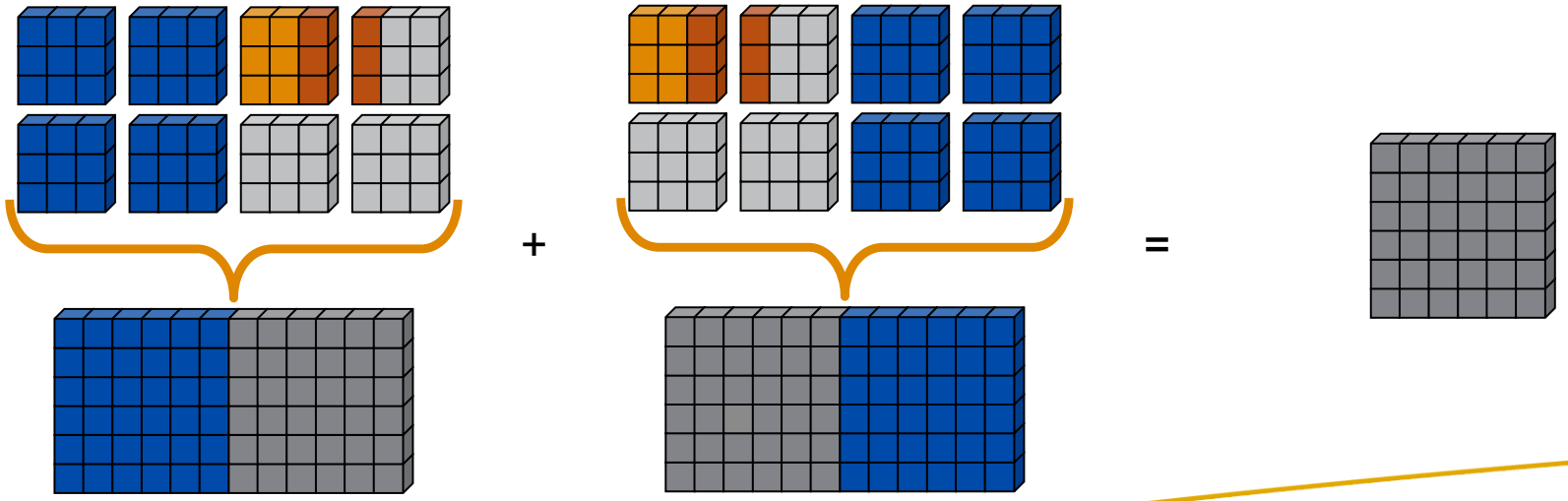
GAiN Binary Ufuncs cont.

- `c = gain.add(a,b,out=c)`
  - If output is distributed
    - access(), get(), or slice corresponding pieces from a and b
    - ***Call original numpy ufunc on the pieces***



GAiN Binary Ufuncs cont.

- `c = gain.add(a,b,out=c)`
  - If output is distributed
    - access(), get(), or slice corresponding pieces from a and b
    - ***Call original numpy ufunc on the pieces***



From NumPy to GAiN

- Not all NumPy programs work as GAiN programs
  - IO, including file, stdin, stdout
    - Replace sys.stdout with custom work-alike
  - Database access
- Possible to use IPython
  - Enhanced interactive Python interpreter
  - Client/Server model for parallelism

What Isn't Finished

- Almost too much to mention
  - order='F' i.e. Fortran ordering of data
  - Sorting routines
  - Linear algebra module
  - Random module
- Comprehensive test suite
- Don't Panic
  - We don't replace NumPy, we enhance it.
  - Symbiosis: reuse the NumPy routines to implement their distributed versions

What the Future Holds

- Scaling studies
- Finish first complete implementation by July
- SciPy 2011, July 11-16 Austin, Texas
  - GA+Python and GAiN tutorial
  - Paper presentation
  - Development sprint
- Attract new users
- Develop ‘real’ applications
- Finish performance analysis by SC’11

Using the Global Arrays Toolkit to Reimplement Python's NumPy for Distributed Computation

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

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

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*