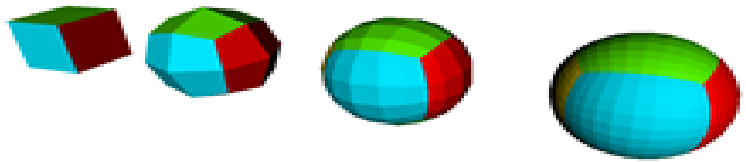
Using Distributed Arrays in UV-CDAT



Alex Pletzer, Dave Kindig, and Srinath Vadlamani (Tech-X Corp.)
Charles Doutriaux (LLNL)

pletzer@txcorp.com

Webex presentation: May 9 2012

Work funded by MoDAVE: DOE/SBIR DE-FG02-08ER85153

Overview

- What are distributed arrays?
- What are distributed arrays good for?
- Parallelism in UV-CDAT
- Distributed arrays in UV-CDAT
  - How to create a distributed array
  - How to access data on other processors
- Ghosted distributed arrays in UV-CDAT
  - A special kind of distributed array for accessing halo data
- Examples

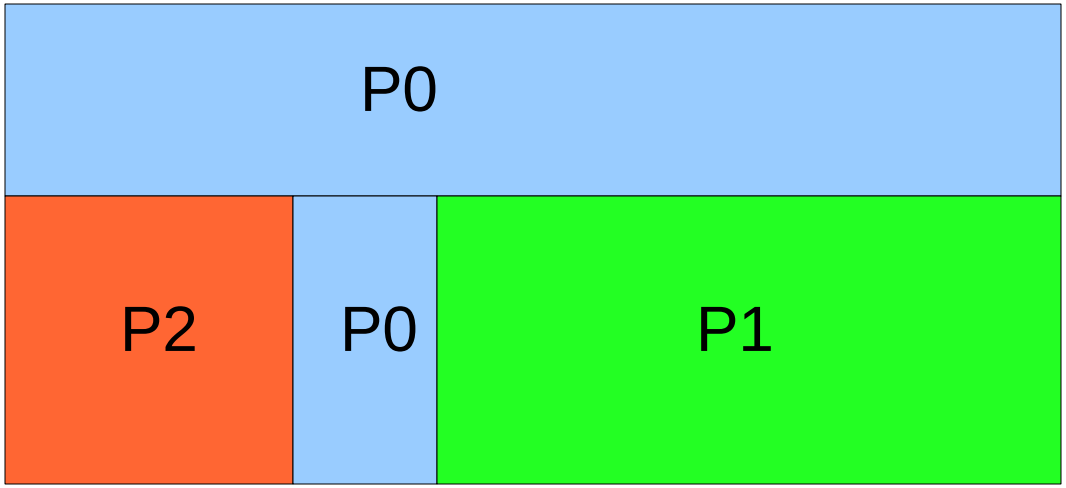
Carry home message

- **You can do parallel computing/postprocessing in UV-CDAT**

```
>>> import distarray
```

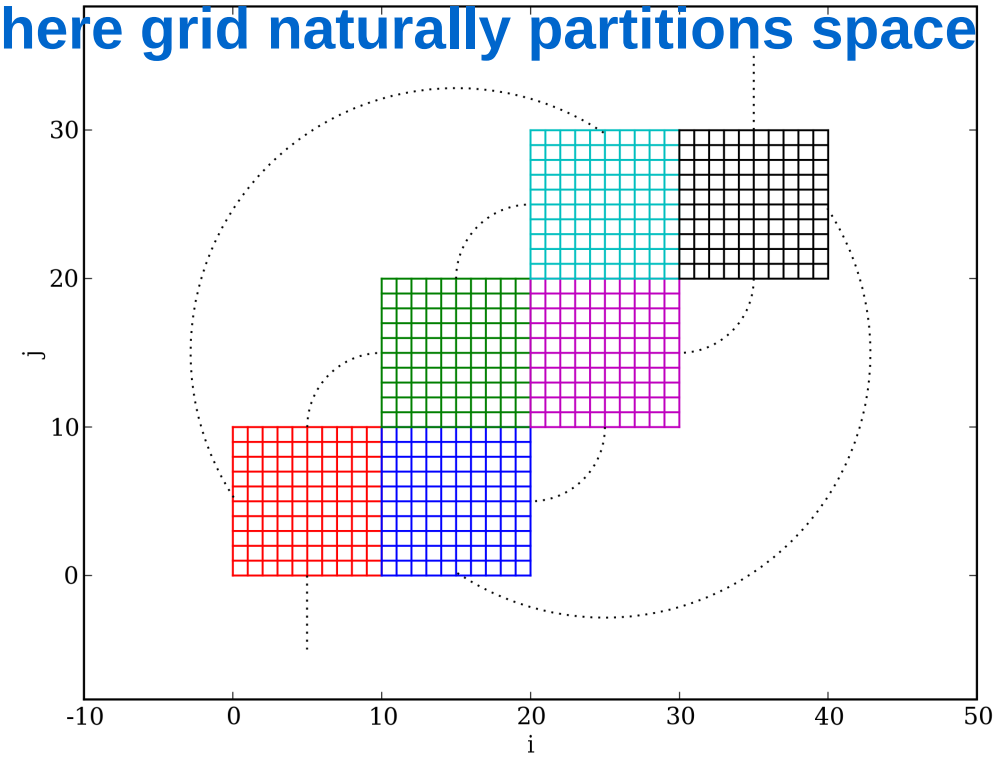
What are distributed arrays?

- A big array that is patitioned in sub-arrays
- Each process (P#) owns a sub-array



What are distributed arrays good for?

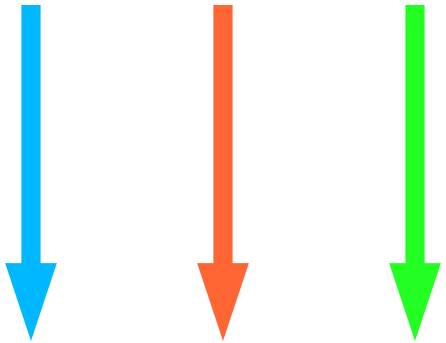
- Divide work among processes
  - Ensemble runs, linear interpolation, finite differencing
- When you don't have enough memory to hold the entire array
  - 0.1 deg: 3600*1800*100*4 = 2.6GB
- Want leverage the cores on your computer
- For convenience
  - the cubed-sphere grid naturally partitions space



Parallelism in UV-CDAT

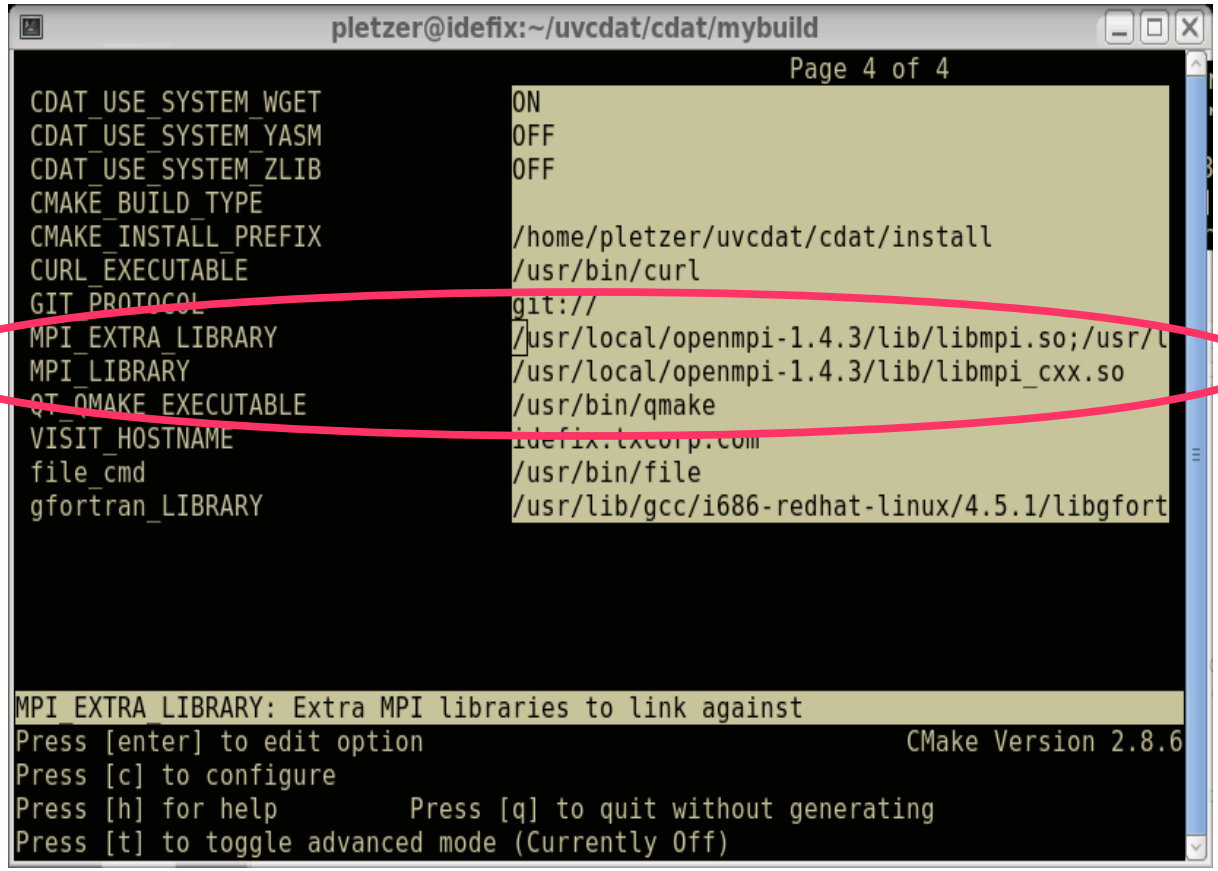
- **UV-CDAT will look for the Message Passing Interface (MPI) library**
  - **Does not assume shared memory**
- **Not implemented: OpenMP, GPU (CUDA, OpenCL), MIC**
- ***The python "threading" module will not help (Python interpreter is not trhead safe)***

MPI execution model:
start to finish



UV-CDAT will build mpi4py if MPI is found

>>> import mpi4py

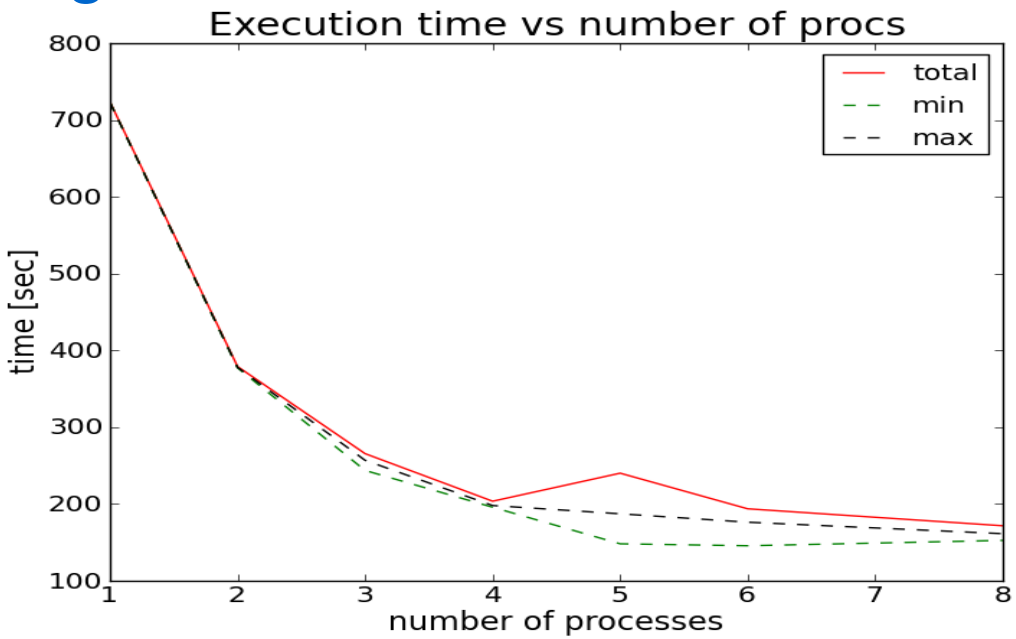


mpi4py: developed by Lisandro Dalcin

For embarrassingly parallel jobs, run your script with....

```
$ mpiexec -n 8 python <my_script.py>
```

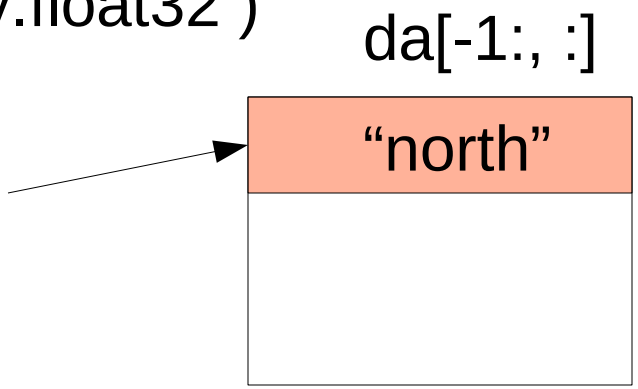
- **Linear interpolation speedup on a 8-core workstation (3D)**
- **Load balancing is the limit**



Distributed array to access remote data

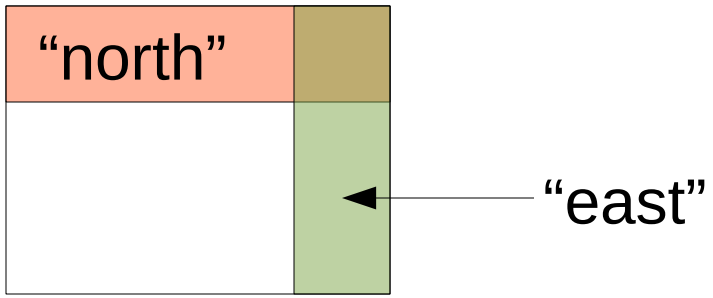
- Each process exposes a “slab” of data (window) to all other processes
- Access the remote data windows using “get” method

```
import distarray
da = distarray.daZeros( (4,5), numpy.float32 )
rk = da.rk   # MPI rank
sz = da.dz  # number processes
northSlab = ( slice(-1, None, None),
              slice(0, None, None) )
da.expose(northSlab, winID='north')
 ...
da[:] = ...          # set data
otherRk = ...     # set src rank
northData = da.get( otherRk, winID='north')
```



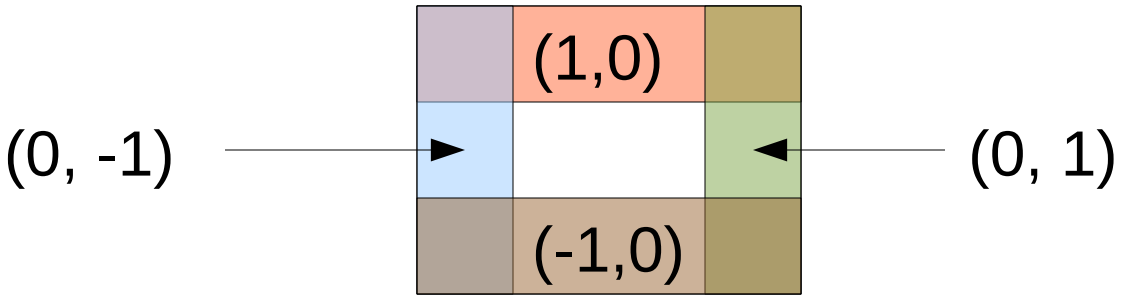
There can be as many slabs as desired

- Each slab gets a unique ID (a string, a tuple, an integer, a “key”)
- Slabs can be overlapping
- A slab can occupy the entire data range
- Supports N-dimensional arrays
- Strides are allowed, non-contiguous data are copied to a buffer
- The get method is a remote memory access
- *All methods are collective*



Ghosted dist arrays will set the slabs for you

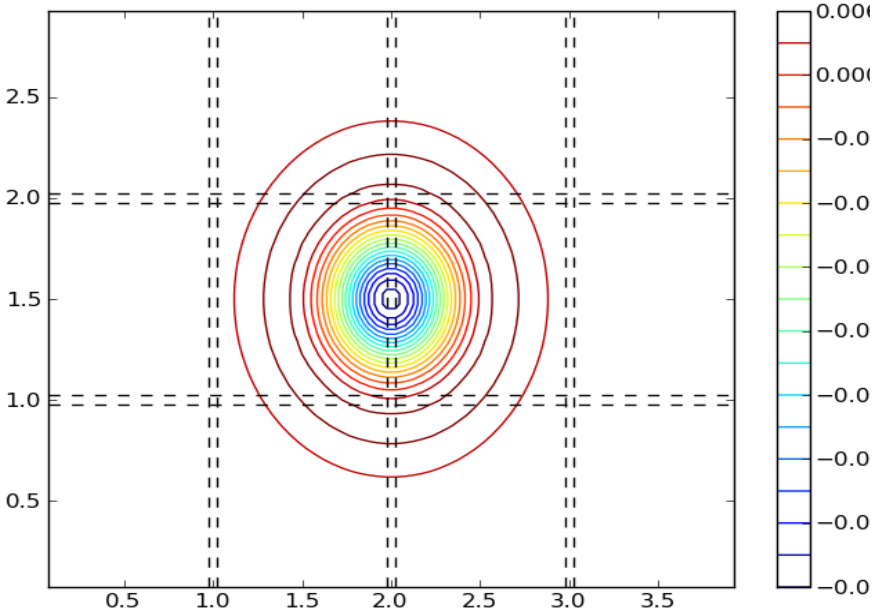
- Each slab gets a unique tuple, e.g. (1, 0) for north, (0, 1) for east, etc.

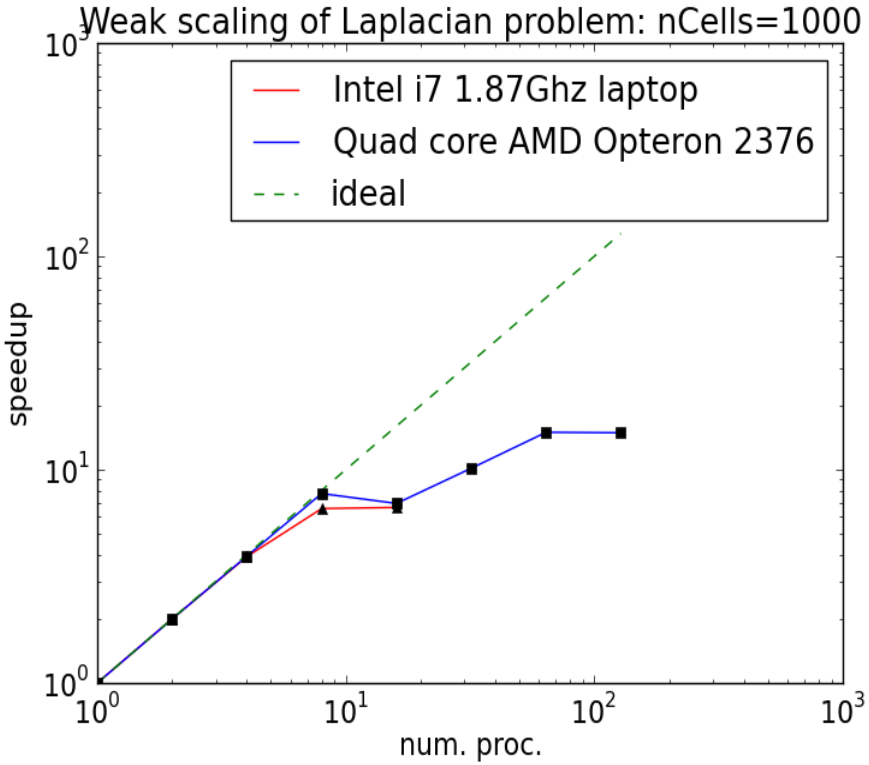



```
gda = distarray.ghZeros( (4,5), numpy.float32,
                  ghostWidth=1 )
...
northData = gda.get(otherRk, winID=(1, 0)) # north
southData = gda.get(otherRk, winID=(-1,0)) # south
```

Example: computing the Laplacian of a function

- Function is a Gaussian
- Regular domain decomposition
- Need neighboring data





Summary

- Pull paradigm, the consumer triggers the communication (requires MPI-2)
- MPI made easy (No MPI_Init, MPI_Finalize, ...)
- Distarray is an extension of numpy array
  - Inherits the behavior of numpy arrays (operations, slicing, etc...)
  - Supports common data types (float64, int32,...)
- More integration with cdms2 arrays may be desirable
  - Should we inherit from cdms2 array?
- May want to add domain decomposition functionality
- Users are required to free the windows (da.free())