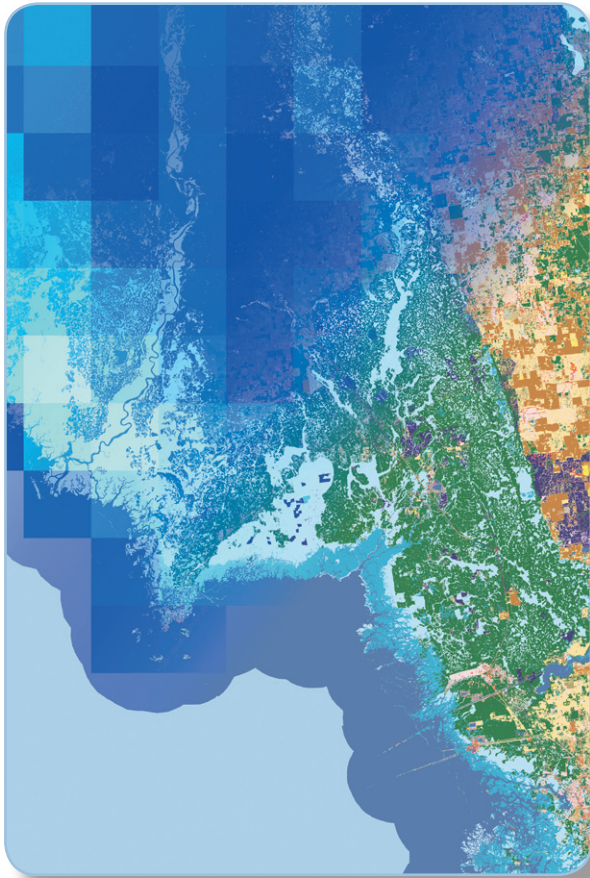


ACCELERATING A GEOSPATIAL APPLICATION USING MATLAB® Distributed Computing Tools

BY NARFI STEFANSSON, KELLY LUETKEMEYER,
AND ROB COMER



Since its genesis as a simple “matrix laboratory,” MATLAB has enabled scientists and engineers to efficiently process large data sets. The Distributed Computing Toolbox and the MATLAB Distributed Computing Engine extend MATLAB with tools for solving computationally intensive problems that exceed the capabilities of a single machine. This article describes a land-cover aggregation and mosaic process implemented with MATLAB distributed computing tools.

Developed by the MathWorks geospatial computing development team in collaboration with Dr. Molly E. Brown of Science Systems and Applications, Inc, sited at the NASA Goddard Space Flight Center, the process addresses the question, “How can we reformulate state-by-state land cover maps, with discrete categories for wetlands, forests, urban areas, and so forth, for comparison with lower-resolution U.S. or global remote sensing data sets?”

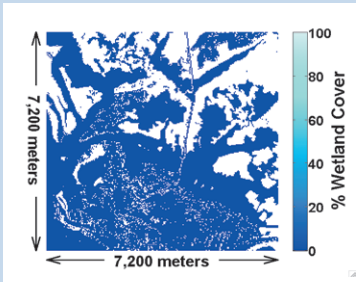
This is an excellent example of a *coarse-grained distributed application*. Also known as *embarrassingly parallel applications*, these types of programs are often associated with Monte Carlo applications and parameter sweeps. The same application runs independently on several nodes, with no communication or data shared between nodes, and the run-time on each node dominates the time needed to start and stop the application.

The Land Cover Aggregation and Mosaic Process

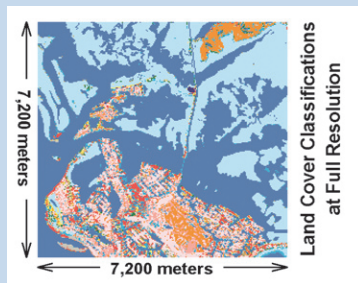
We begin by downloading the National Land Cover Data set (NLCD) from the U.S. Geological Survey (USGS). The data set includes classification grids for each state. A grid consists of 30-meter-square map cells, each assigned one of roughly 20 discrete land-cover classes. Each cell contains either 0% or 100% of any given

Aggregating a Land Cover Class

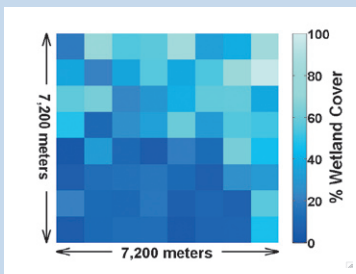
Our aggregation procedure exploits MATLAB array subscripting and utilities in a single, efficient processing step. These images illustrate aggregation of wetlands into 900-meter-square cells, combining the two wetland classes, for an area on the southwest coast of Florida. All three images are in the map coordinate system that the NLCD specifies for Florida.



▲ Binary image showing cells that belong to a wetland class as white and all other cells as deep blue.



▲ The original NLCD classification, with the same classification key as the image shown in Figure 1. The prevalence of pale blue indicates a concentration of woody wetland in the northern half of the area.



▲ The aggregate image, which shows smooth gradations of wetland content with much coarser spatial resolution.

class—for example, either no mixed forest or all mixed forest (Figure 1).

The high density and volume of the NLCD data, plus the fact that each classification grid resides in its own map coordinate system, makes comparison with many national or global data sets impractical. We therefore select a specific class or a group of related classes (aggregated square blocks of cells within the grid for each state) and construct a mosaic that covers the entire conterminous U.S. in a new map coordinate system and with a coarser spatial resolution.

Our aggregation procedure trades spatial resolution for finer data resolution. Instead of quantizing to two levels (0% and 100%), the mosaic stores the percentage of a specific class of land cover as a continuous value in each map cell.

Because each classification grid resides in its own map coordinate system, we must conduct a separate aggregation for each grid and reproject the results to the U.S. map coordinate system used for the final mosaic (Figure 2).

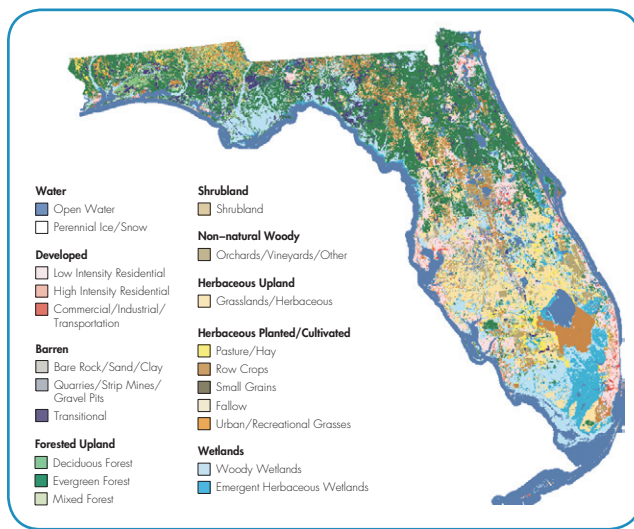


Figure 1. Reduced-resolution, color-coded NLCD classification map for Florida with matching classification key. Both were constructed entirely in MATLAB, as were all the map displays in this article.

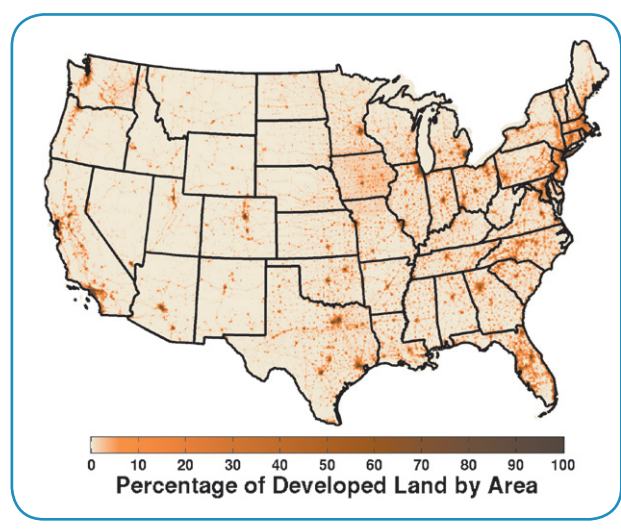


Figure 2. Mosaic showing the density of developed land, as a percentage of each 8-km-square map cell, in the conterminous U.S.

Implementing the Process— Sequential Version

This application lends itself to distributed computing because most of the processing is conducted state by state, providing an obvious basis for coarse-grained parallelism.

Distributed computing applications can be more difficult to implement than sequential applications. By first developing good sequential code, we benefit from all the strengths of the MATLAB interactive environment¹. Once the sequential code is stable, we distribute the computations.

The functions in the sequential version of the code reflect the natural computational sequence:

1. Set up the correct referencing projection and initialize the output array.
2. Loop over the list of states:
 - a. Import the NLCD map grid for the current state.
 - b. Create an aggregated land cover map grid for that state.
 - c. Reproject the aggregation results to the U.S. map coordinate system.
3. Assemble the individual reprojected state land cover images into a single mosaic image.

This structure has several advantages: it reduces the complexity of task distributions, limits each function's input and output data, and lets us maintain the sequential and distributed versions of the code with minimal code redundancy.

Transitioning to the Distributed Model

Our sequential computations involve repeating the same procedure multiple times, using data from different states. We use either a single, large task or up to 48 smaller tasks, one for each contiguous state. To maintain flexibility, we specify the number of tasks via an input parameter to the distributed version of the code, but we use 48 tasks for our timing measurements.

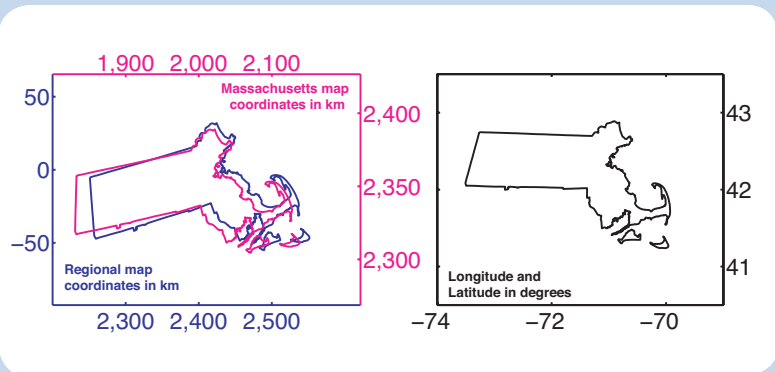
Reprojecting Each State

Once the aggregation process for an individual state is complete, the resulting raster grid must be reprojected (transformed geometrically) to the U.S. map coordinates and copied into the mosaic.

Applying a geometric transformation to a raster grid or image usually involves applying the inverse transformation to map each output cell center back to the input space and then interpolating a new value. In our example, the inverse transformation occurs in two steps: unprojecting cell-center locations in the mosaic from U.S. map coordinates to the latitude-longitude system, then projecting them into the state's map coordinates using the Mapping Toolbox functions `projinv` and `projfwd`

```
[lat, lon] = projinv(regionProj, usX, usY);  
[stateX, stateY] = projfwd(stateProj, lat, lon);
```

Finally, we scale and shift `stateX` and `stateY` to their corresponding row and column offsets in the state grid, which we pass to the Image Processing Toolbox function `tformarray` to interpolate the data values to insert in the U.S. mosaic.



The outlines of Massachusetts show that the reprojection process involves small local rotations and scale change as well as a shift in origin.

After dividing the states into groups, one for each task, we use the Distributed Computing Toolbox to create the job and its tasks and submit the job to the cluster to perform the land-cover computations. Whenever a worker finishes a task, the client MATLAB obtains the results from the job manager and inserts them into the mosaic image.

When distributing the computations, we consider the following points:

- Average run-time for each state is approximately 90 seconds. Because task overhead is minor relative to task run-time, we have a coarse-grained application even when using 48 tasks.
- None of the machines is set up as a file server. To optimize cluster performance, we copy all 30 GB of land cover data to each worker so that only a few tens of kilobytes need to be transmitted with each task.
- We do not balance loads; states are distributed among the tasks alphabetically, not according to size.

Memory Mapping

The NLCD data files are quite large, even for an individual state. MATLAB memory-mapping features enable us to treat a state-wide data grid like a MATLAB array, without having to hold it all in memory. We construct a `memmapfile` object

```
m = memmapfile(filename, 'Offset', 0, 'Format', ...
    {'uint8', [arrayWidth, arrayHeight], 'nlcdArray'});
then pass m.Data.nlcdArray to our aggregation function as if it were an
ordinary MATLAB uint8 array
[aggregatedData, mask] = aggregate(m.Data.nlcdArray, ...
    nlcdCategory, scaleFactor);
```

Hardware Platform and Performance Results

We run our application on a cluster consisting of four 2.8 GHz Pentium IV machines, each with 512 MB of memory and running Windows XP, connected with Gigabit Ethernet. The MATLAB Distributed Computing Engine is installed on each machine, and a worker from the

engine is started. One machine also runs the job manager. The client machine is a laptop with just 512 MB of memory.

We improve the run-time of the land cover computations significantly by using all four machines instead of just one: A single task running on only one worker takes 4,651 seconds, while 48 tasks running on all four workers take just 1,280 seconds—a remarkable reduction in run-time for very little additional development time. <<

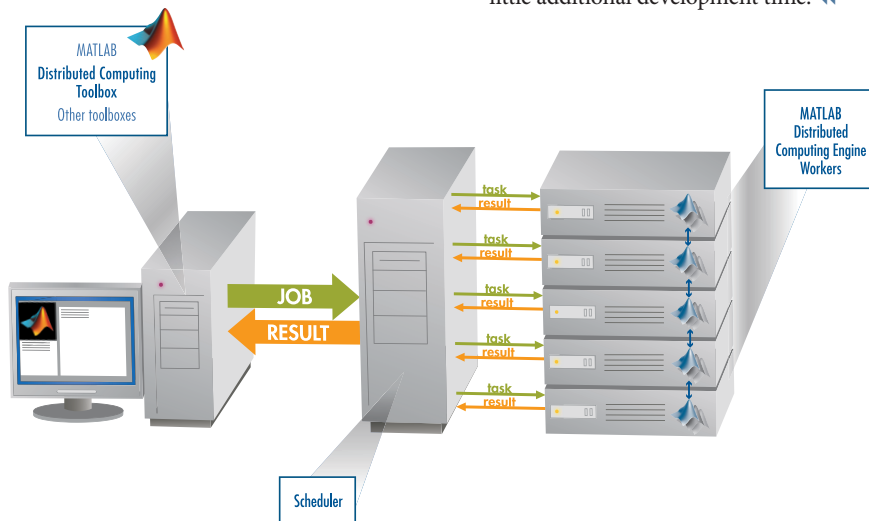


Figure 3. The interaction between the client machine, where the Distributed Computing Toolbox is used to define jobs and tasks, and the MATLAB Distributed Computing Engine.

1 Kepner, Bond, Kim, Reuther and Travinin. "Parallel MATLAB: Current Achievements and Future Challenges." Ohio Supercomputing Center MATLAB Workshop, November 8, 2004. *Supercomputing 2004*.

Distributed Computing Glossary

Client—Computer on which we enter MATLAB commands

Job—Large MATLAB or Simulink® operation to be performed on a computer cluster

Tasks—Segments of a job: the units that the workers receive for execution

Job manager—Process that coordinates and asynchronously distributes tasks to the workers

Worker—Noninteractive MATLAB instance that processes tasks assigned by the job manager and sends the results back to the job manager

Dynamic Licensing

The MATLAB Distributed Computing Engine runs any algorithms that use toolboxes or blocksets for which the client is licensed. This eliminates the need to buy multiple toolboxes and blockset licenses for the worker nodes.

RESOURCES

- ▶ **Distributed Computing Toolbox**
www.mathworks.com/res/distribtb
- ▶ **Mapping Toolbox**
www.mathworks.com/res/mapping
- ▶ **Image Processing Toolbox**
www.mathworks.com/res/image
- ▶ **Webinar: Distributed Computing with MATLAB & Simulink**
www.mathworks.com/res/dcwebinar
- ▶ **USGS Land Cover Characterization Program**
<http://landcover.usgs.gov>
- ▶ **NASA Goddard Space Flight Center**
www.nasa.gov/centers/goddard/home