Cornell University
Center for Advanced Computing

Parallel MATLAB: The Parallel Computing Toolbox, MDCS, and Red Cloud

Steve Lantz
Senior Research Associate
Cornell Center for Advanced Computing

Seminar for HD-HNI, Mar. 6, 2015

Cornell University
Center for Advanced Computing

Overview of Parallel Computing Toolbox (PCT)

PCT Opens Up Parallel Possibilities

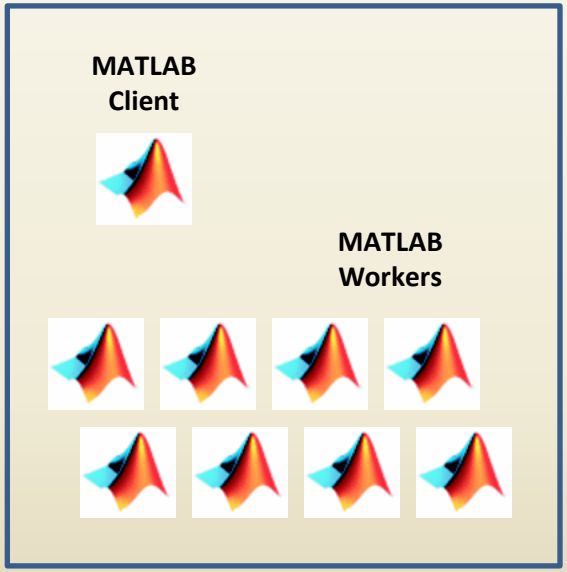
- MATLAB does *multithreading* implicitly in core array ops.
- To exploit parallelism beyond this, a user needs to insert PCT commands. In order of increasing complexity:
  - Parallel for-loops: **parfor**
  - Single program, multiple data: **spmd**, **pmode**
  - Partitioned arrays for big-data parallelism: **(co)distributed**

PCT Opens Up Parallel Possibilities

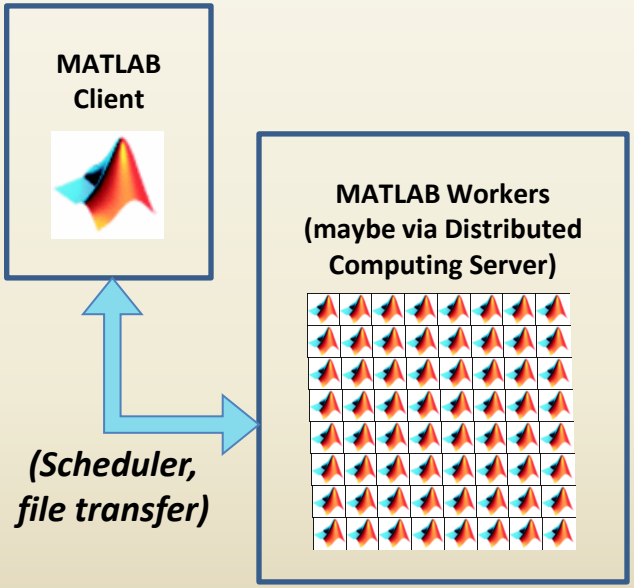
- MATLAB does *multithreading* implicitly in core array ops.
- To exploit parallelism beyond this, a user needs to insert PCT commands. In order of increasing complexity:
  - Parallel for-loops: **parfor**
  - Single program, multiple data: **spmd**, **pmode**
  - Partitioned arrays for big-data parallelism: **(co)distributed**
  - Multiple batch-style runs of a serial function: **createJob**
  - Batch-style run of a parallel function: **createParallelJob** (= pmode), **createMatlabPoolJob** (if parfor/spmd sections)

Two Ways to Use PCT

Interactively - vs. - batch-style



Set up matlabpool – enter PCT commands at console*



*Select local pool **or** remote cluster – submit task script*

**or* *parpool in R2013b*

Major PCT Concepts

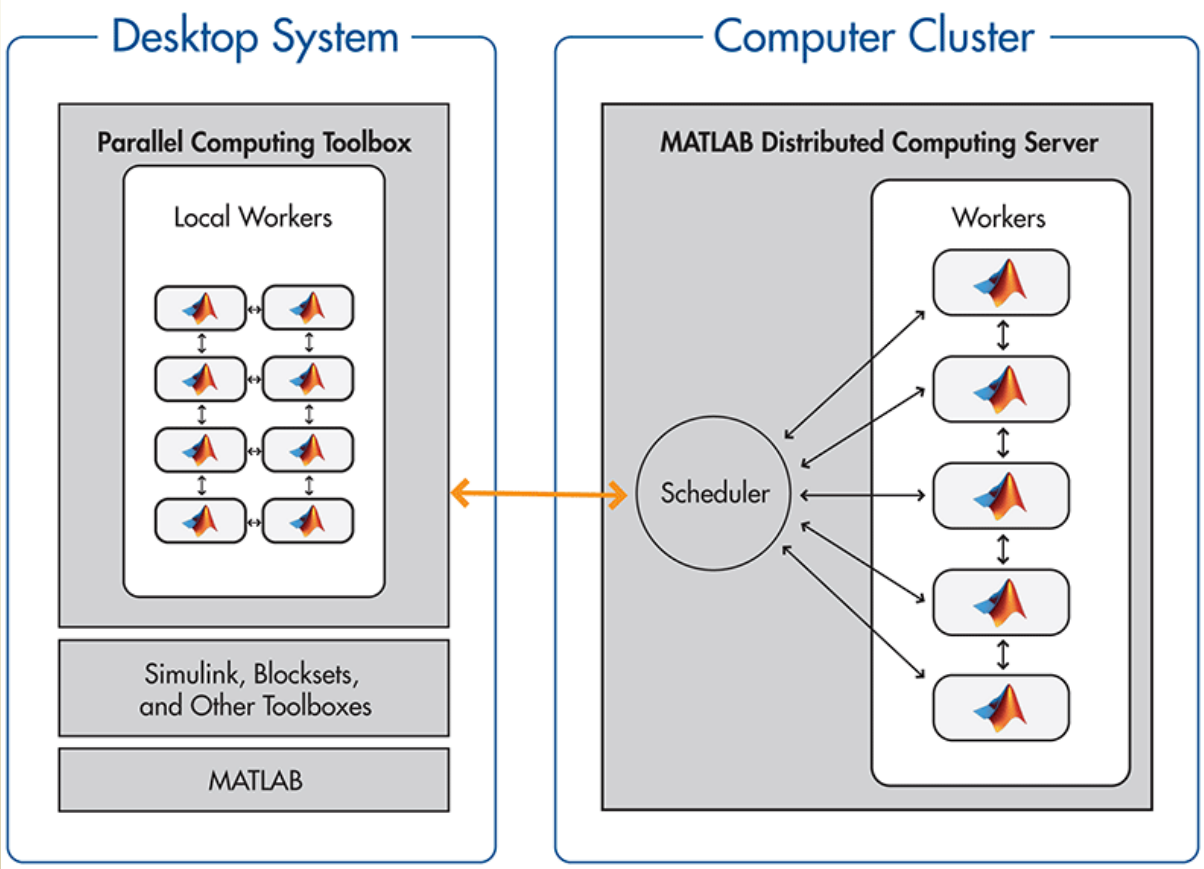
- **matlabpool**: pool of separate MATLAB processes = "labs"
  - Differs from multithreading! No shared address space
  - Ultimately allows same concepts to work on MDCS clusters
- **parfor**: parallel for-loop, iterations must be independent
  - Labs (workers) split up work; load balancing is built in
- **spmd**: single program, multiple data
  - All labs execute every command; labs can communicate
- **(co)distributed**: array is partitioned among workers
  - "Multiple data" to spmd, one array to MATLAB built-ins

What If You Outgrow Your Laptop?

- This is where MDCS comes in: *switch to batch-style.*
- PCT's interfaces allow a third party (e.g., CAC) to write implementations of PCT functions that talk to an MDCS cluster, but look the same to you as when run locally.
- Select parallel resources by using a configuration/profile, or by issuing the findResource/parcluster command.
  - Choose "local" to stay local; choose "cacscheduler" to tie PCT methods to CAC-specific implementations
  - You don't ever call the underlying functions directly

Parallel Resources: Local & Remote

PCT



MDCS

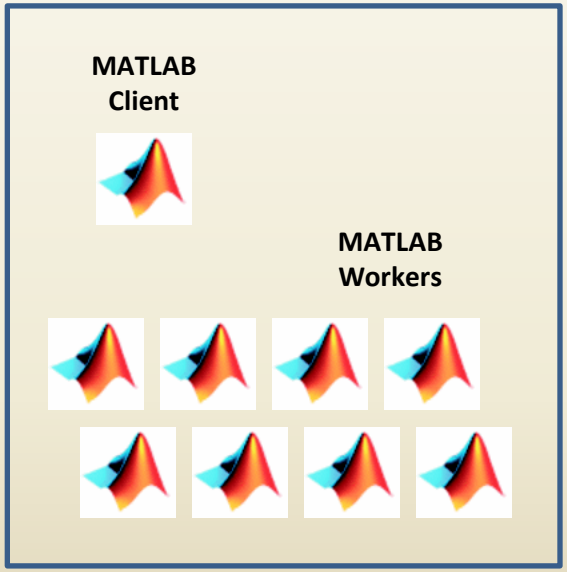
How to Do It Without PCT or MDCS

- Create a MATLAB .m file that takes one or more input parameters (such as the name of an input file).
- Apply the MATLAB C/C++ compiler (mcc), which converts the script to to C, then to a standalone executable.
- Run N copies of the executable on an N-core batch node or a cluster, each with a different input parameter
  - mpirun can launch non-MPI processes, too
- Matlab runtimes (free!) must be available on all nodes
- For process control, write a master script in Python, say

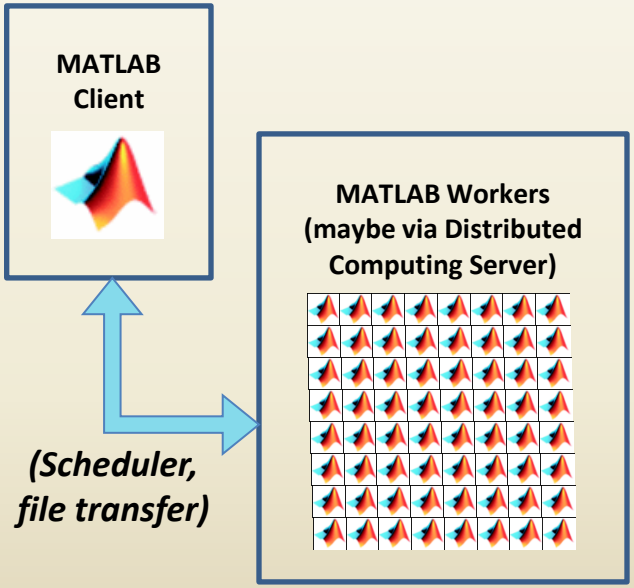
Red Cloud

Two Ways to Use PCT

Interactively - vs. - batch-style



Set up matlabpool – enter PCT commands at console*

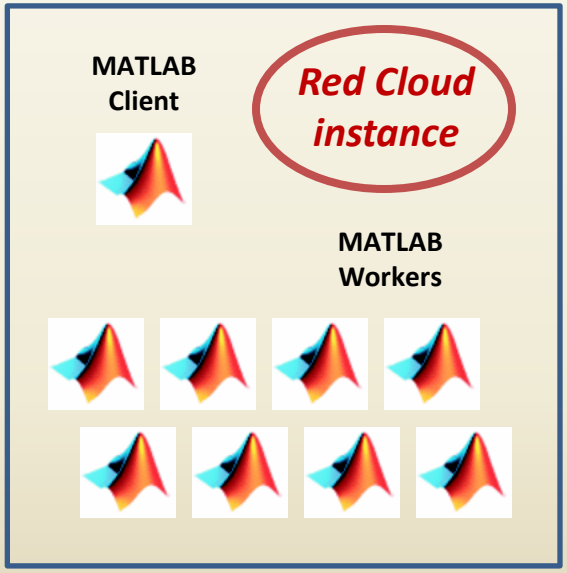


*Select local pool **or** remote cluster – submit task script*

**or* parpool in R2013b*

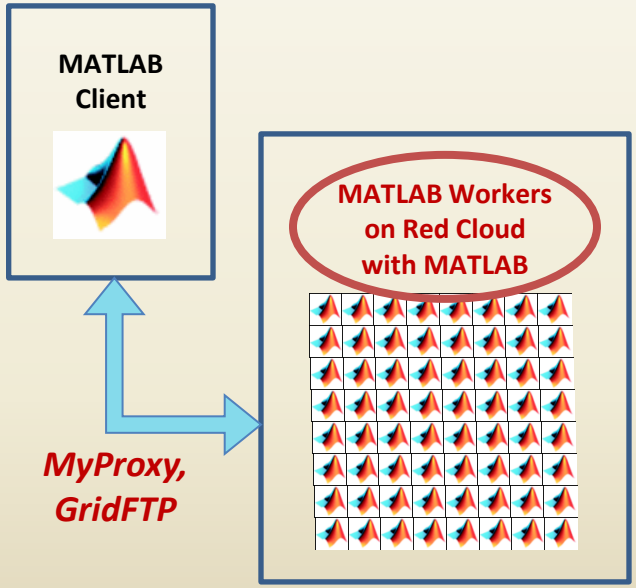
Two Ways to Use PCT at CAC

Red Cloud



Log in to an instance based on an image with MATLAB

Red Cloud with MATLAB



Select CAC as your remote cluster – submit task script

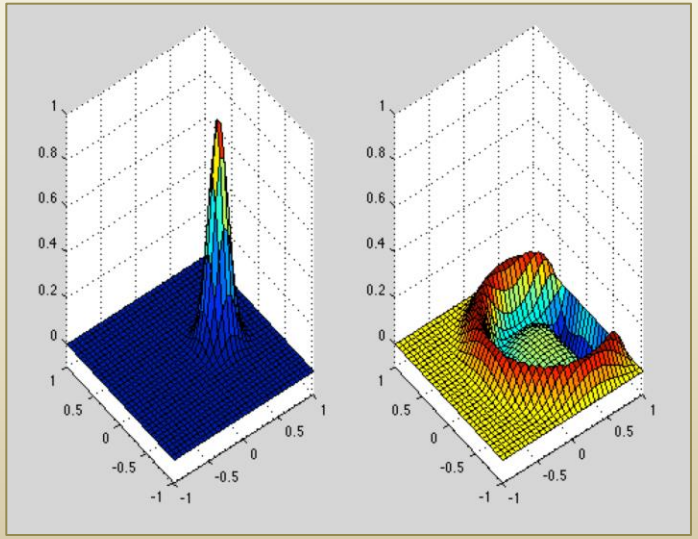
**or parpool in R2013b*

Case Study:
GPGPU and MATLAB PCT

A Word About GPUs

- Red Cloud with MATLAB features 8 nodes with dedicated NVDIA Tesla M2070 GPUs capable of 1 Tflop/s each!
- MATLAB PCT has built-in GPU functions that provide an easy way to program the GPUs without learning CUDA

Stop by after the lecture to see a demo of how to run a wave simulator on Red Cloud's NVIDIA GPUs



GPGPU in MATLAB: Fast and Easy

- Initial benchmarking with large 1D and 2D FFTs shows excellent acceleration on 1 GPU vs. 8 CPU cores
  - Including communication: up to 10x speedup
  - Excluding communication: up to 20x speedup
- MATLAB code changes are trivial
  - Move data to GPU by declaring a gpuArray
  - Methods are overloaded to use internal CUDA code on gpuArrays

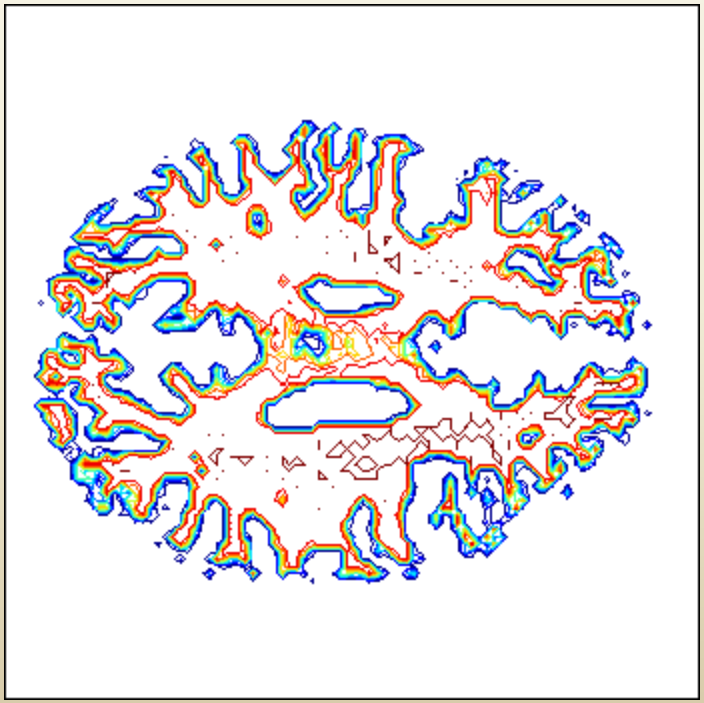
```
g = gpuArray(r);
f = fft2(g);
```

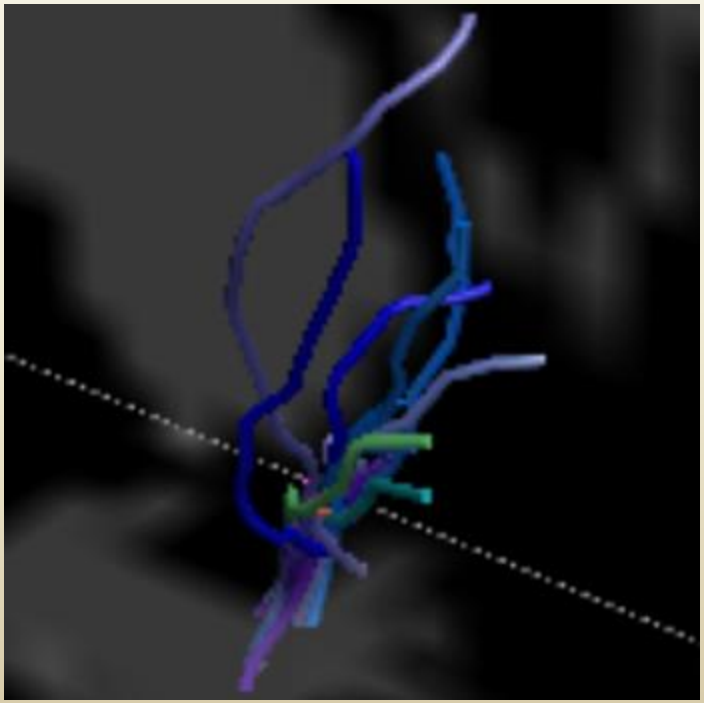
Analysis of MRI Brain Scans

- Work by Ashish Raj and Miloš Ivković, Weill-Cornell Medical College
- Research question: Given two different regions of the human brain, how interconnected are they?
- Potential impact of this technology:
  - Study of normal brain function
  - Understanding medical conditions that damage brain connections, such as multiple sclerosis, Alzheimer's, TBI
  - Surgical planning

Connecting Two Types of MRI Data

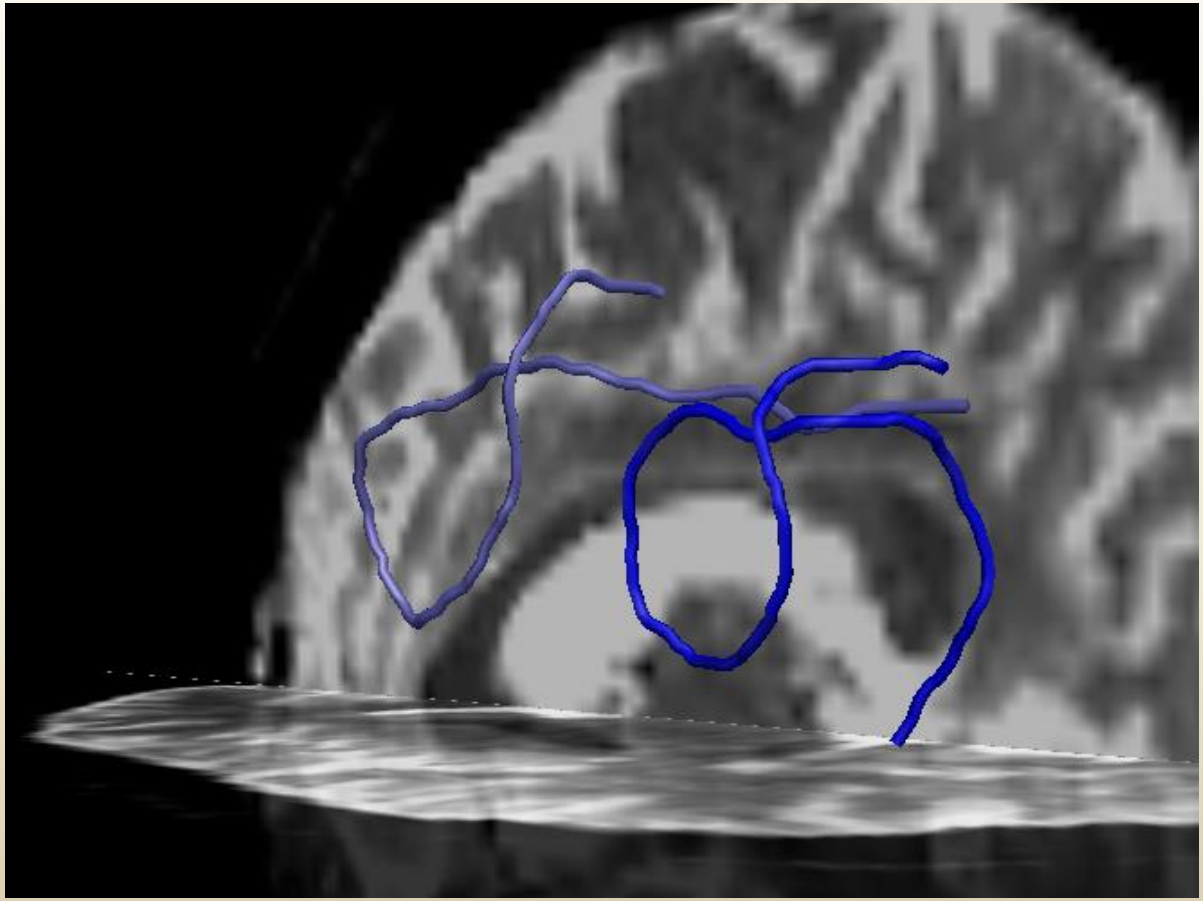
- 3D MRI scans to map the brain's white matter
- Fiber tracts to show lines of preferential diffusion





Need for Computational Power

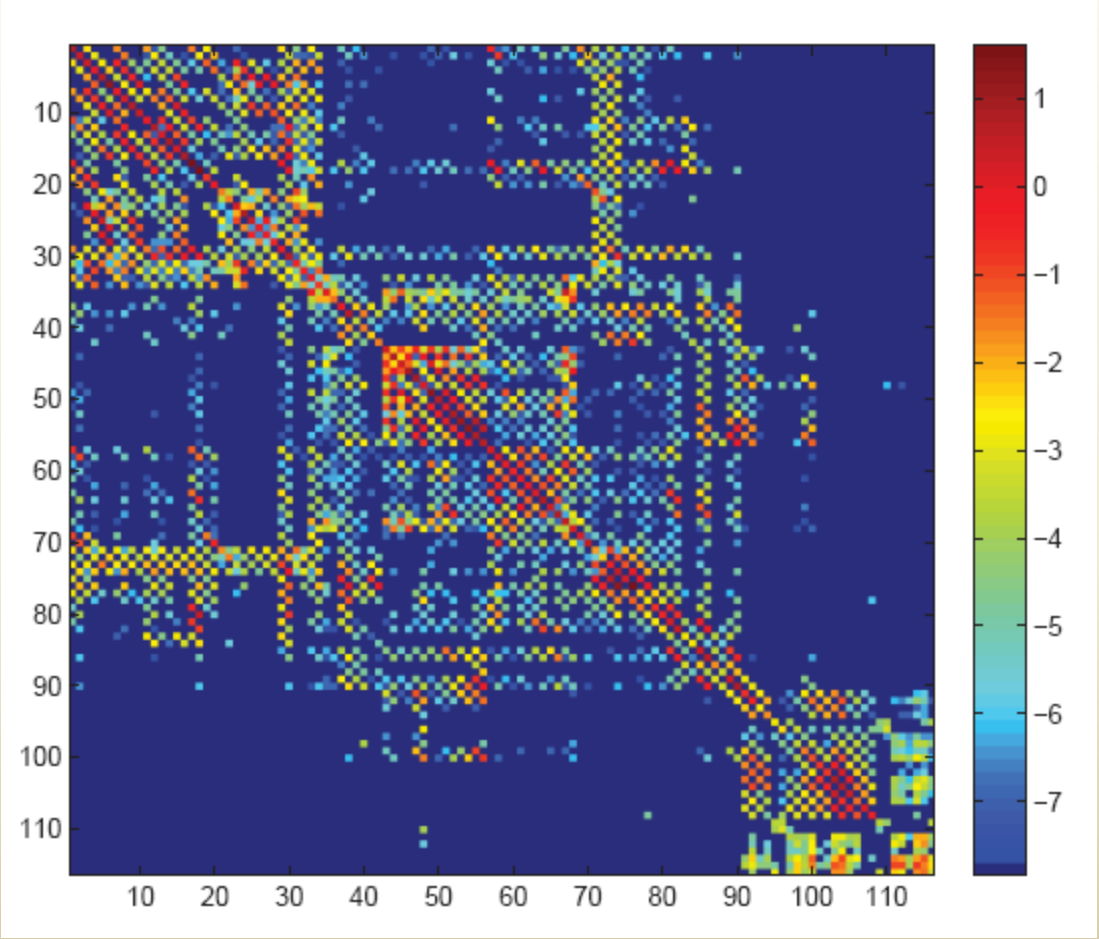
- Problem: long, spurious fibers arise in first-pass analysis
- Solution: use MATLAB to re-weight fibers according to importance in connections



Examples of improbable fibers eliminated by analysis

Data Product: Connectivity Matrix

- Graph with 360K nodes, 1.8M edges, optimized in 1K iterations
- The reduced digraph at right is based on 116 regions of interest



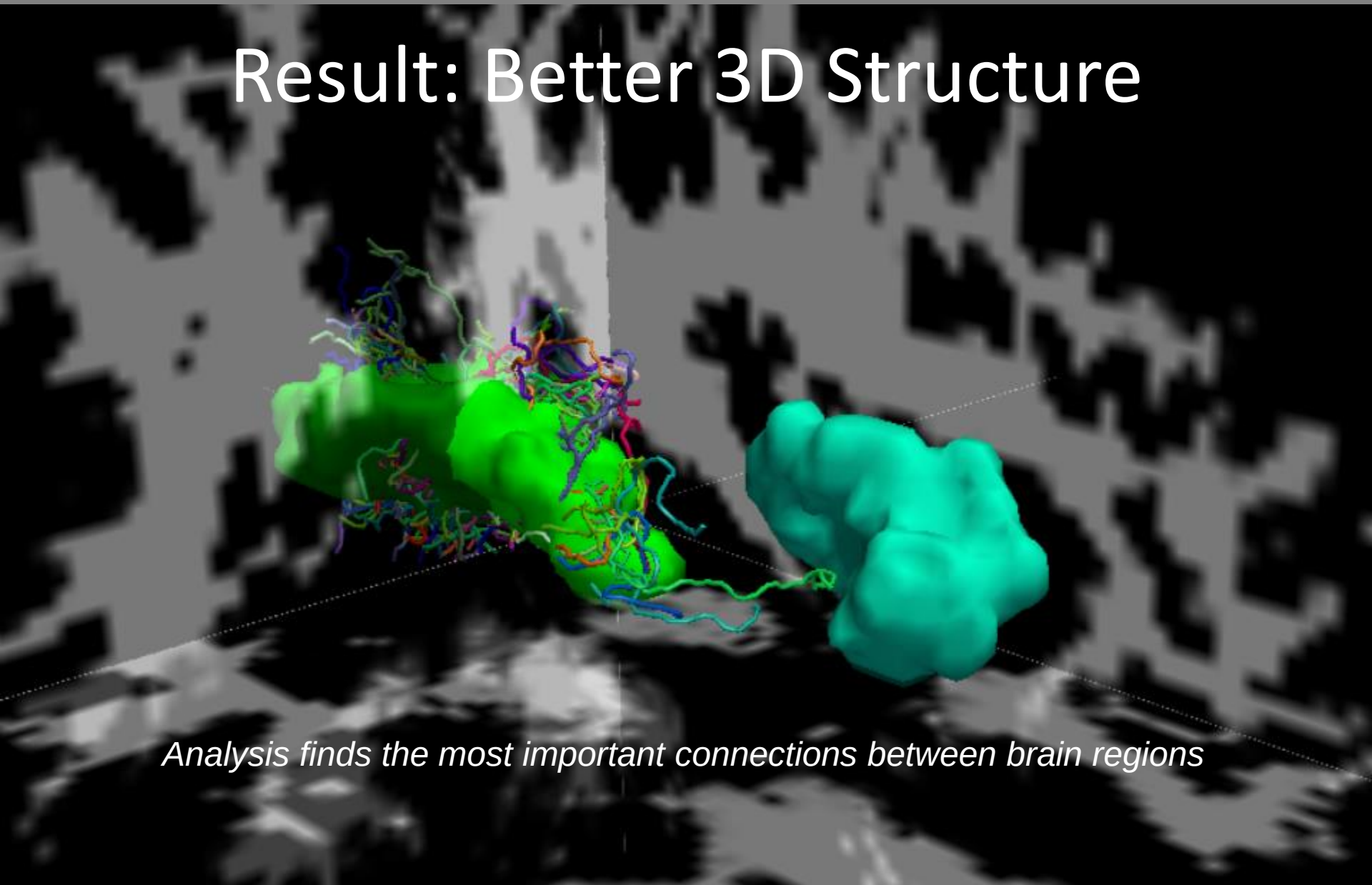
MATLAB Loves Matrices!

- Original code was written using **structs**
  - Advantage: little wasted space; handles variable-length lists of edges connected to a voxel (1–274) or fiber (2–50)
  - Disadvantage: poor data locality, because structs hold lots of extraneous info about voxels/fibers
  - Disadvantage: unsupported on GPU in MATLAB
- Better to store data in **matrices**!
  - Column-wise operations are often multithreaded
  - Matrix operations are often vectorized on CPUs or GPUs

GPU Optimization

- Since R2011b, min and sum will work on gpuArrays!
- Keep big, simple matrices with top-heavy columns
  - Reason 1: GPU doesn't deal well with nested for-loops
  - Reason 2: Want vectorized, SIMD ops on millions of items
- Resulting code is actually *less* complex
  - Keep data in a few huge arrays
- Best result (after a few tricks): **0.15 sec/iteration**
  - 350x speedup over un-optimized, matrix-based version
  - 2500x speedup over initial struct-based version

Result: Better 3D Structure



Analysis finds the most important connections between brain regions

Are GPUs Really That Simple?

- No. Your application must meet four important criteria.
  1. Nearly all required operations must be implemented natively for type GPUArray.
  2. The computation must be arranged so the data seldom have to leave the GPU.
  3. The overall working dataset must be large enough to exploit 100s of thread processors
  4. The overall working dataset must be small enough that it does not exceed GPU memory.

PCT and MDCS: The Bottom Line

- PCT can greatly speed up the analysis of large datasets
- GPU functionality is a good addition to the arsenal
- Yes, a learning curve must be climbed…
  - General knowledge of how to restructure code for parallel and vector computing
  - Specific knowledge of PCT functions
- But speed matters!…
  - MRI image analysis, e.g., is transformed from a research curiosity into a diagnostic tool for real-time, clinical use