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

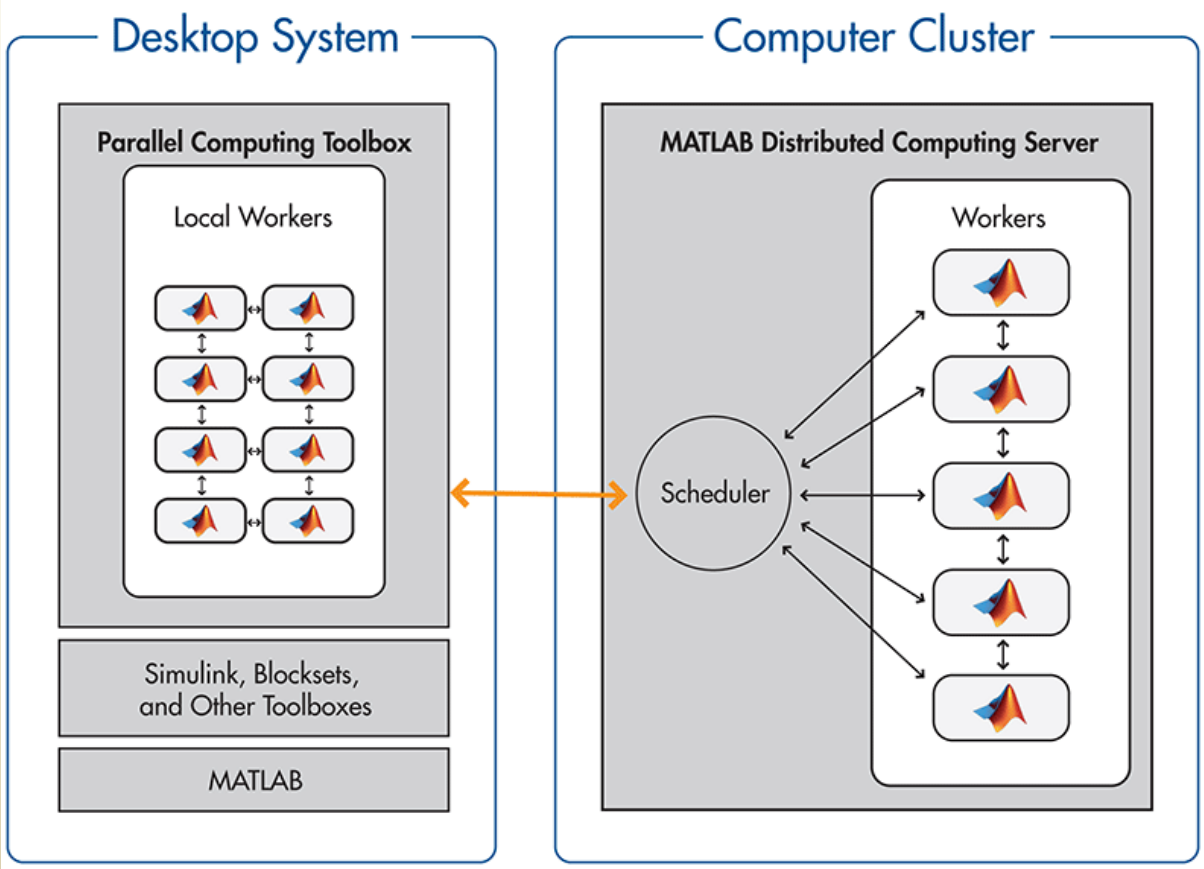
Seminar for the Bioinformatics Practitioners Club, Nov. 3, 2014

Cornell University
Center for Advanced Computing

Overview of Parallel Computing Toolbox (PCT)

Parallel Resources: Local & Remote

PCT



MDCS

PCT Opens Up Parallel Possibilities

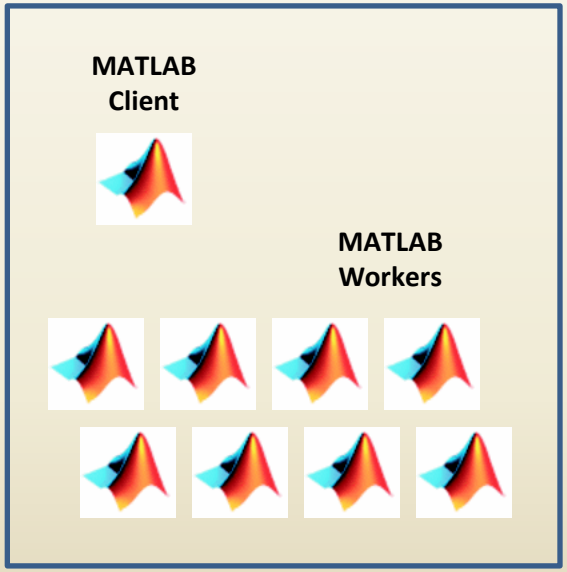
- MATLAB does *multithreading* implicitly in core array ops.
- To exploit parallelism beyond this, a user needs to insert PCT commands. In order of increasing complexity:
  - Parallel for-loops: **parfor**
  - Single program, multiple data: **spmd**, **pmode**
  - Partitioned arrays for big-data parallelism: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

PCT Opens Up Parallel Possibilities

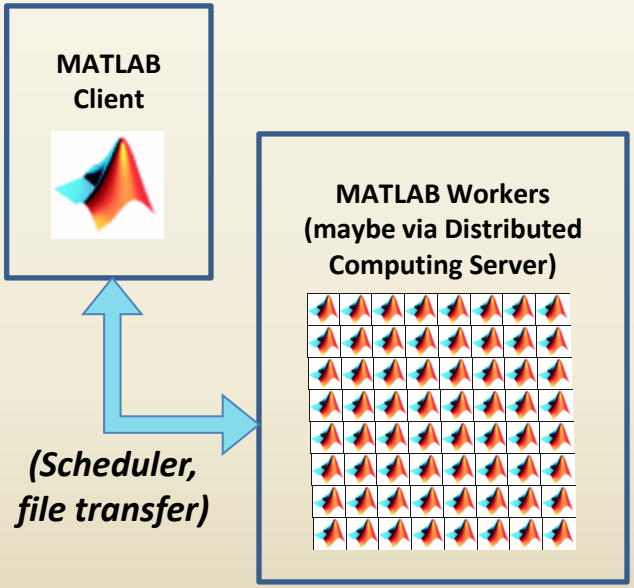
- MATLAB does *multithreading* implicitly in core array ops.
- To exploit parallelism beyond this, a user needs to insert PCT commands. In order of increasing complexity:
  - Parallel for-loops: **parfor**
  - Single program, multiple data: **spmd**, **pmode**
  - Partitioned arrays for big-data parallelism: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

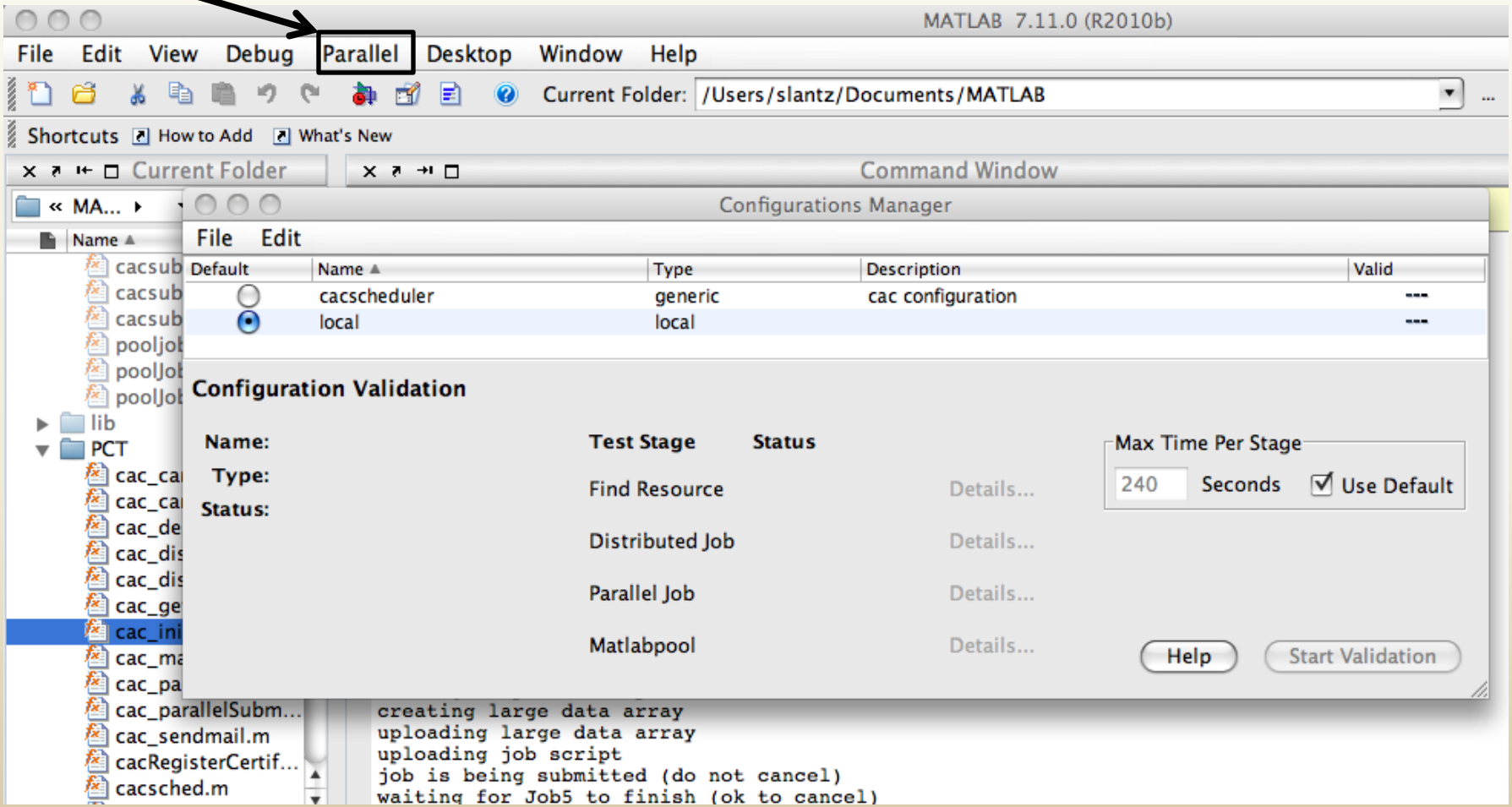
Major PCT Concepts

- **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

Using a Configuration/Profile



findResource/parcluster

- If you download the CAC client-side code, cacsched.m shows you how to call the findResource function.

```
sched = findResource('scheduler','type','generic');
```

- Examine cac_initialize.m to see how the PCT interfaces are tied to to specific functions provided by CAC.

```
set(sched,'SubmitFcn',{@cac_distributedSubmitFcn,clusterHost,Remo
set(sched,'ParallelSubmitFcn',{@cac_parallelSubmitFcn,clusterHost
set(sched,'GetJobStateFcn',@cac_getJobStateFcn);
set(sched,'CancelJobFcn',@cac_cancelJobFcn);
set(sched,'CancelTaskFcn',@cac_cancelTaskFcn);
set(sched,'DestroyJobFcn',@cac_destroyJobFcn);
set(sched,'ClusterOsType','pc');
set(sched,'ClusterMatlabRoot',ClusterMLRoot);
```

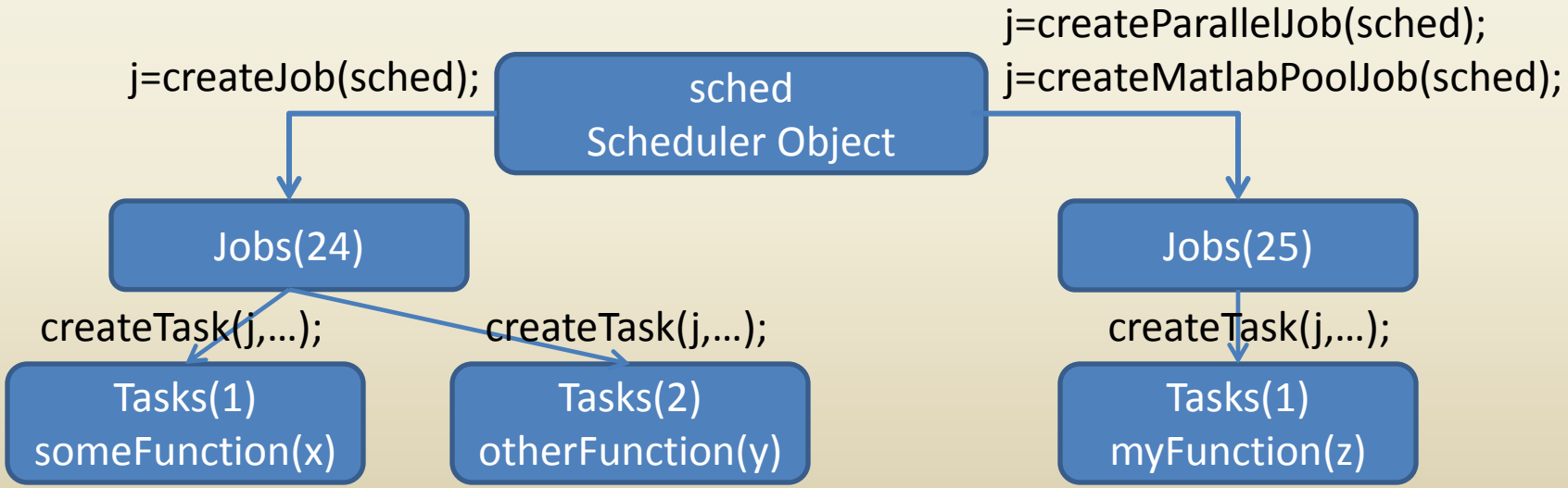
The Great Name Change in R2012a

Previous Name	New Name
findResource	parcluster
scheduler object	cluster object
Configuration	Profile
createJob	createJob (no change)
createParallelJob	createCommunicatingJob (where 'Type' = 'SPMD')
createMatlabPoolJob	createCommunicatingJob (where 'Type' = 'Pool')
createTask	createTask (no change)
getAllOutputArguments	fetchOutputs
destroy	delete
…etc., etc…	See www.mathworks.com/help/distcomp/release-notes.html

Using Batch-Style PCT

Jobs and Tasks

- findResource creates a scheduler object, which allows you to create Jobs. In PCT, Jobs are containers for Tasks, which are where the actual work is defined.



Distributed Jobs

- PCT has 3 types of jobs: distributed, parallel, and pool.
- Distributed jobs have one or more tasks and no communication between tasks.
  - An MDCS scheduler runs each task as a one-core batch job
  - Useful for shifting a series of lengthy tasks to CAC, e.g.

```
job = createJob(sched);
for i = 1:5
    %Each one core job will run [out1,out2] = myfunction(i);
    task = createTask(job,@myfunction, 2, {i});
end
submit(job);
```

Parallel and Pool Jobs

- Parallel and Pool jobs are multi-core or even multi-node.
  - Communication between cores/nodes must be possible.
  - The number of workers (labs) must be given.
  - These jobs have just one task!

```
job = createParallelJob(sched);
%10 core job
set(job, 'MaximumNumberOfWorkers', 10);
set(job, 'MinimumNumberOfWorkers', 10);
%All 10 cores will execute [out1] = parallelFunction(0.25,'lswt');
task=createTask(job, @parallelFunction, 1, {0.25,'lswt'}) ;
submit(job);
```

Parallel Jobs

- All workers (labs) run the task function.
- The task function is responsible for implementing the actual parallelism using “labindex” logic.
- PCT supports MPI-style commands inside parallel jobs.

Size and rank are available from the start of the job.

labindex = MPI_Comm_rank+1
numlabs = MPI_Comm_size+1

Initialization is done for you (no MPI_Init).

```
fprintf('I am %d', labindex);

if labindex == 1
    A = labBroadcast(1,magic(numlabs));
else
    A = labBroadcast(1);
end
```

More on Parallel Jobs

- All basic message-passing methods are available: Send, Receive, Broadcast, Barrier, gop (allreduce or allgather)
- Source and tag are the same as in MPI. MATLAB figures out datatypes for you.
  - labSend(data,dest,[tag]);
  - labReceive(source,tag);
  - labReceive(); % take any

```
if labindex == 1
    labSend(rand(5,5),2,5);
elseif labindex == 2
    randMatrix = labReceive('any', 5);
    labSend(randMatrix,3);
else
   randMatrix = labReceive(2);
end
```

- (Co)distributed arrays are sliced across workers so huge matrices can be operated on. Collect slices with gather.

Pool Jobs

- One worker acts as the proxy for your MATLAB client. This "master" runs the task function.
- The rest of the workers act as the labs in a matlabpool. These labs run parfor/spmd sections of the task function.

```
a(1) = gpuDeviceCount();
spmd
    b = labindex + gpuDeviceCount();
    c = numlabs;
end
for i=1:c{1}
    a(i+1) = b{i};
end  % for 8 GPUs, expected answer is a = [1:8]
```

State of Jobs

- After a job is submitted, “job.state” is just one of several different ways to learn the state of the job.

```
submit(job);
%Periodically check the status (qstat -j jid)
job.state
%Block until the job is finished
waitForState(job);  %==waitForState(job,'finished');
```

- waitForState is a PCT interface to block on job state, which can be problematic if jobs take a long time or fail.
- If more control is desired, check job.state periodically to see if the job finished or failed.

Retrieving Results

- Once your job completes, you need to get the results in two steps: (1) download files, (2) load into workspace.
- Download is only needed for MDCS jobs. It is triggered automatically by checking on job.state for a completed job, or by a blocking call to waitForState(job).
- Loading the results requires a separate function call
  - a = getAllOutputArguments(job) returns cell array a{Task,Output}
  - a{1,2} = Task 1, second output

Works the Same Everywhere!

Command Window

```
>> sched = findResource('scheduler','configuration','cacscheduler');
>> j = createJob(sched);
>> t = createTask(j,@rand,1,{3,3});
>> set(t,'CaptureCommandWindowOutput',1);
>> submit(j);
Retrieved R2011b as current matlab version
**************************************

Need assistance? Contact help@cac.cornell.

**************************************


[2012-05-25 18:37:56,411] Using certificat

>> wait(j);
Downloading completed job: Job227.
>> a = getAllOutputArguments(j);
>> a{1,1}

ans =

    0.9173    0.4809    0.4626
    0.6839    0.4612    0.8009
    0.8661    0.1562    0.2155
```

We can control which resource is used to execute the job simply by swapping out the scheduler object!

Command Window

```
>> sched = findResource('scheduler','configuration','local');
>> j = createJob(sched);
>> t = createTask(j,@rand,1,{3,3});
>> set(t,'CaptureCommandWindowOutput',1);
>> submit(j);
>> wait(j);
>> a = getAllOutputArguments(j);
>> a{1,1}

ans =

    0.9173    0.4809    0.4626
    0.6839    0.4612    0.8009
    0.8661    0.1562    0.2155
```

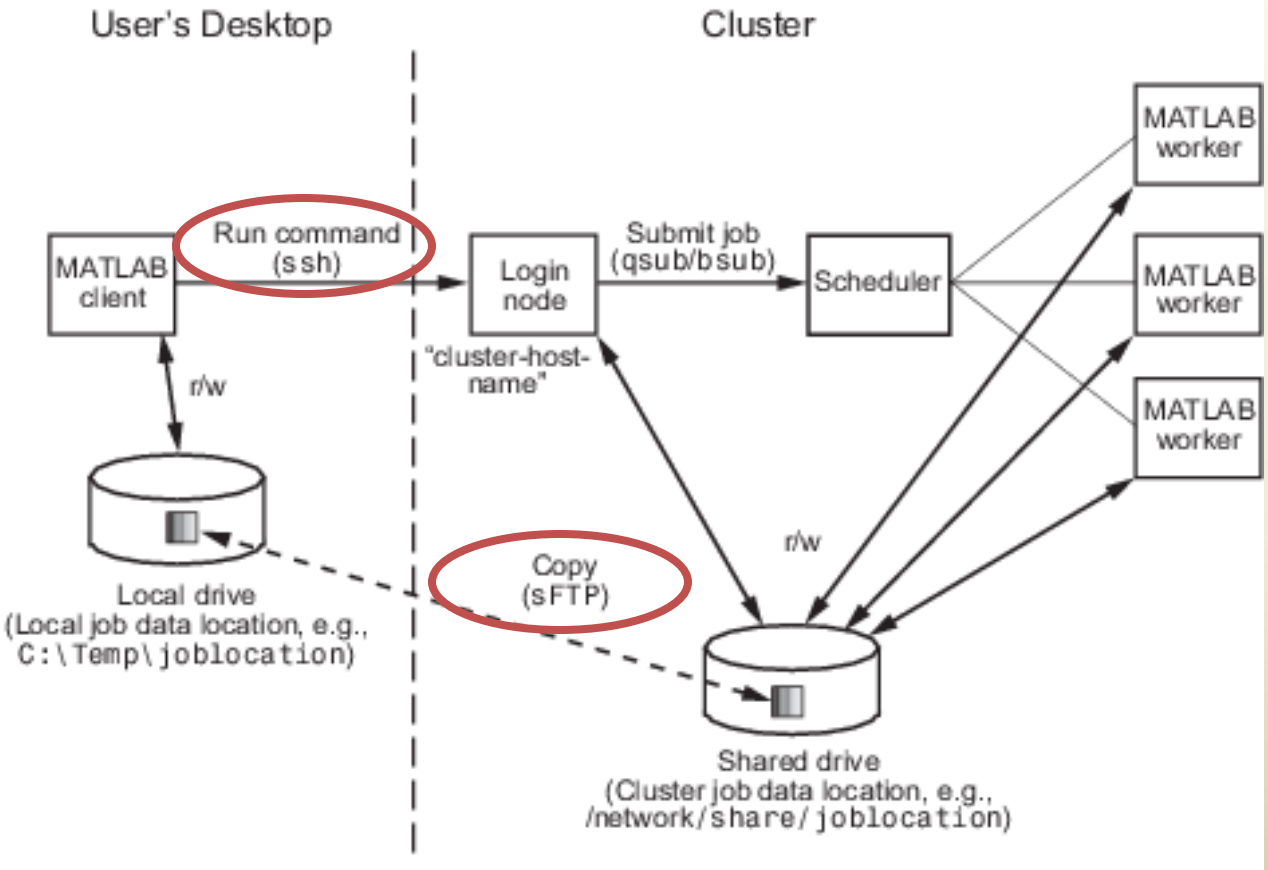
How to Do It Without PCT or MDCS

- 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 machine 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

Cornell University
Center for Advanced Computing

Overview of MATLAB Distributed Computing Server (MDCS) and File Transfer

Connect to MDCS with ssh and sftp



When Is File Transfer Needed?

- If you have a custom function and/or require a datafile:

```
j = createJob(sched);
createTask(j,@rand,1,{3,3});
createTask(j,@myfunction,1,{3,3});
submit(j);
waitForState(j);
a = getAllOutputArguments(j);
```

- The **rand** function is no problem at all, it's built in, but **myfunction.m** does not exist on the remote computer.
- Transfer this file and get it added to the path.

MATLAB Can Copy the Files…

- Setting the *FileDependencies* property tells MATLAB to copy the files for you.
- Specify the directories and files the task will need. All files and directory structure will be copied.
- Not very efficient, though: file transfer occurs separately for each worker running a task for that particular job.

```
>> set(j,'FileDependencies',{'/home/username/src/myfunction.m',...
'/home/username/data/dfile.mat');
```

…Or Copy the Files Yourself

- *FileDependencies* is best for smaller projects with only a couple of files.
- Alternative for larger files:
  1. Copy the file(s) using sftp, or GridFTP
  2. Add the path to the worker sessions
- *PathDependencies* is used to make the task function available at run time.

```
>> set(j,'PathDependencies',{' \\matlabstorage01\matlab\username'});
```

Remote File Storage at CAC

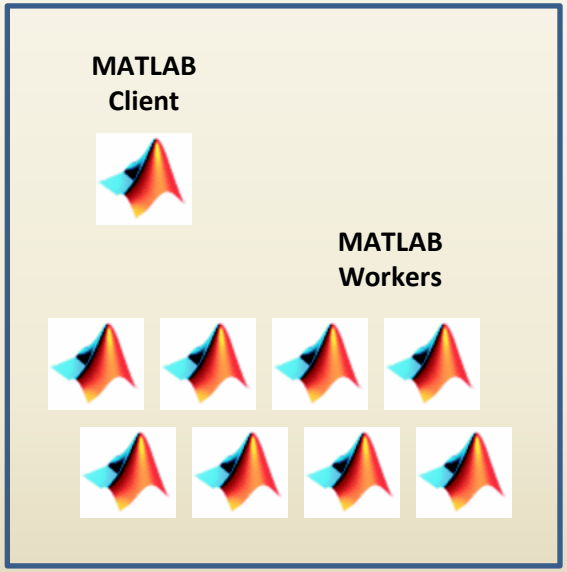
- Subscriptions have 50GB of storage space
  - Intended for MATLAB scripts, job data, etc.
  - Accessible to all MATLAB jobs run by the same user
  - Can be expanded by adding extra storage to a subscription
- General access is provided through GridFTP

```
>> help gridFTP
>> ftp = gridFTP();
>> ftp.list('');
```

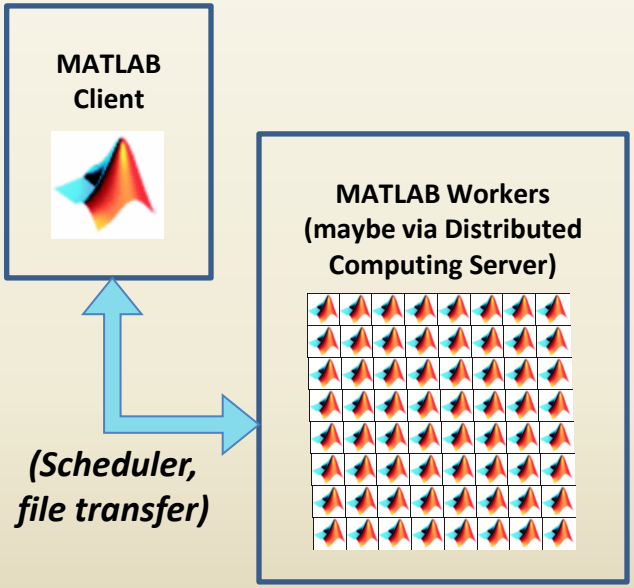
Red Cloud

Two Ways to Use PCT

Interactively - vs. - batch-style



Set up matlabpool – enter PCT commands at console*

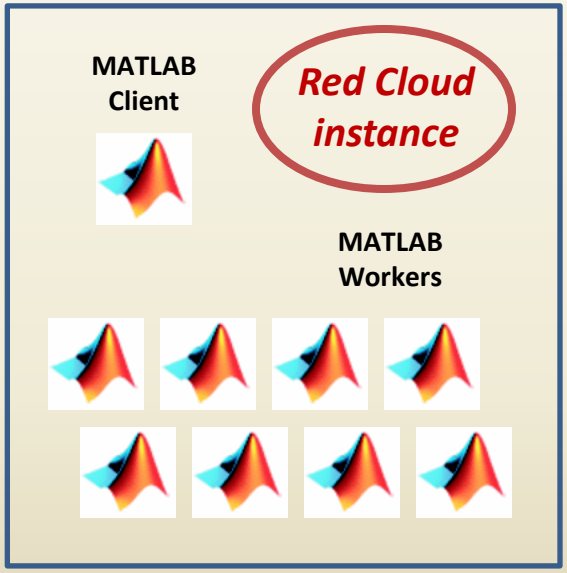


*Select local pool **or** remote cluster – submit task script*

**or* parpool in R2013b*

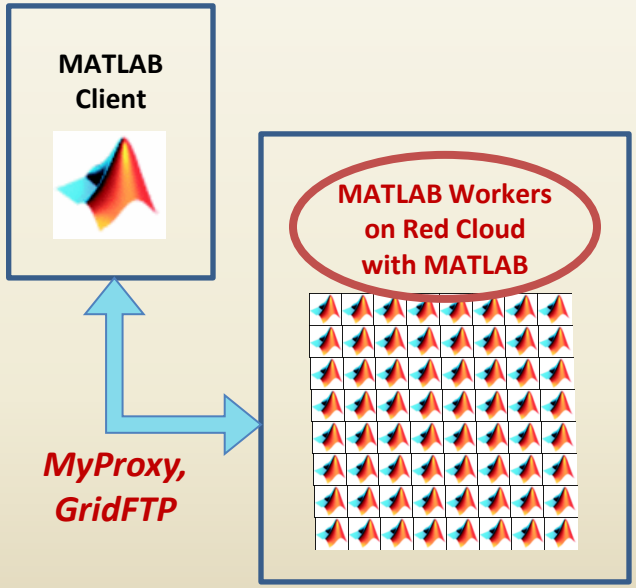
Two Ways to Use PCT at CAC

Red Cloud



Log in to an instance based on an image with MATLAB

Red Cloud with MATLAB

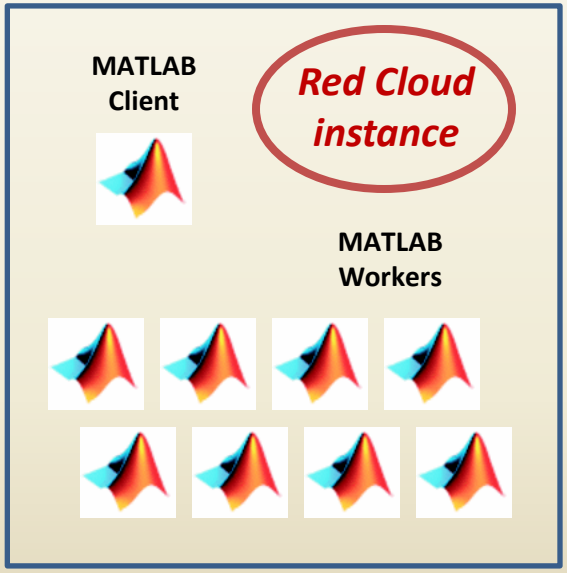


Select CAC as your remote cluster – submit task script

**or parpool in R2013b*

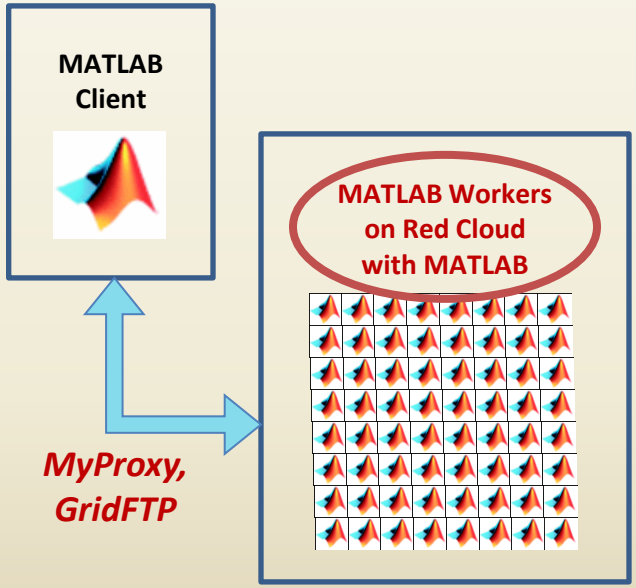
Two Ways to Use PCT at CAC

Infrastructure as a Service



Log in to an instance based on an image with MATLAB

Software as a Service



Select CAC as your remote cluster – submit task script

**or parpool in R2013b*

Red Cloud with MATLAB

- This “Software-as-a-Service” (SaaS) enables a broad research community to run MATLAB on CAC’s high-performance resources in a secure, useable manner.
- Both hardware and software components make up the system. They integrate with the end user’s MATLAB client at different levels.
- All functions are provided by various “services”, meaning you never actually log on to any CAC systems. The client software simply makes requests to CAC systems.

Current System Specifications

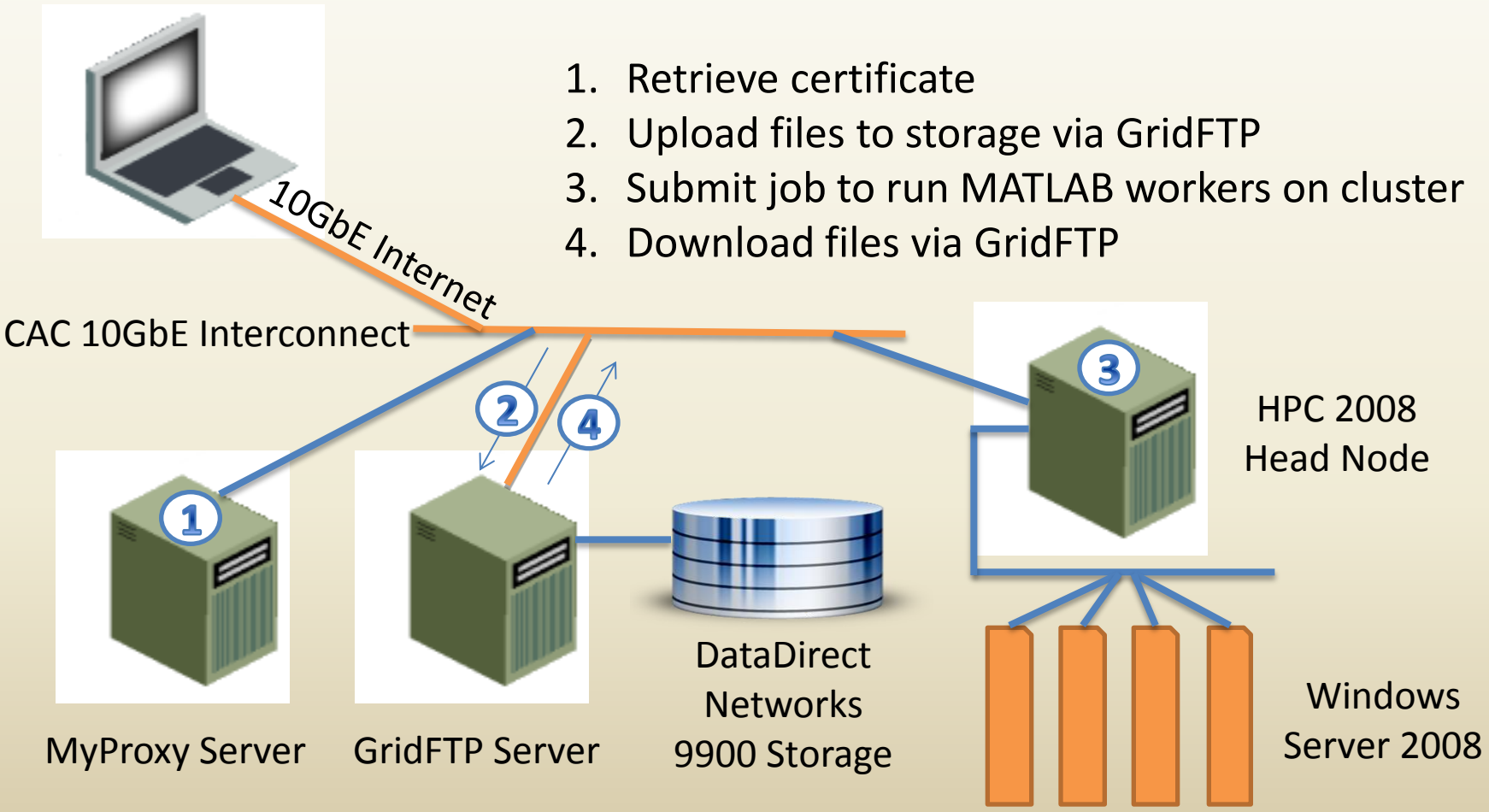
- Microsoft Windows HPC Server 2008 cluster
  - Supports MATLAB clients on Windows, Mac, and Linux
  - Releases R2010b, R2011a, R2011b, R2012a, and R2013a
- 64 Intel cores in 8 Dell C6100 blade servers
  - Per server: 2 4-core Xeon E5620s @ 2.4 GHz
  - In Dell C410x: 8 NVIDIA Tesla M2070s, 1 Tflop/s, 6 GB each
  - 8 GPU-linked cores have 10GB RAM each; others have 2GB
- 8TB DataDirect Networks storage: RAID-6, error correction
  - Accessible by all servers and externally at 10 Gb/s

Services and Security

- **File transfer service**
  - Move files through a GridFTP (specialized FTP) server to a network file system that is mounted on all compute nodes
- **Job submission service**
  - Submit and query jobs on the cluster (via TLS/SSL); jobs are executed by MATLAB workers on the compute nodes
- **Security and credentials**
  - Send username/password over a TLS encrypted channel to MyProxy; get a short-lived X.509 certificate granting access

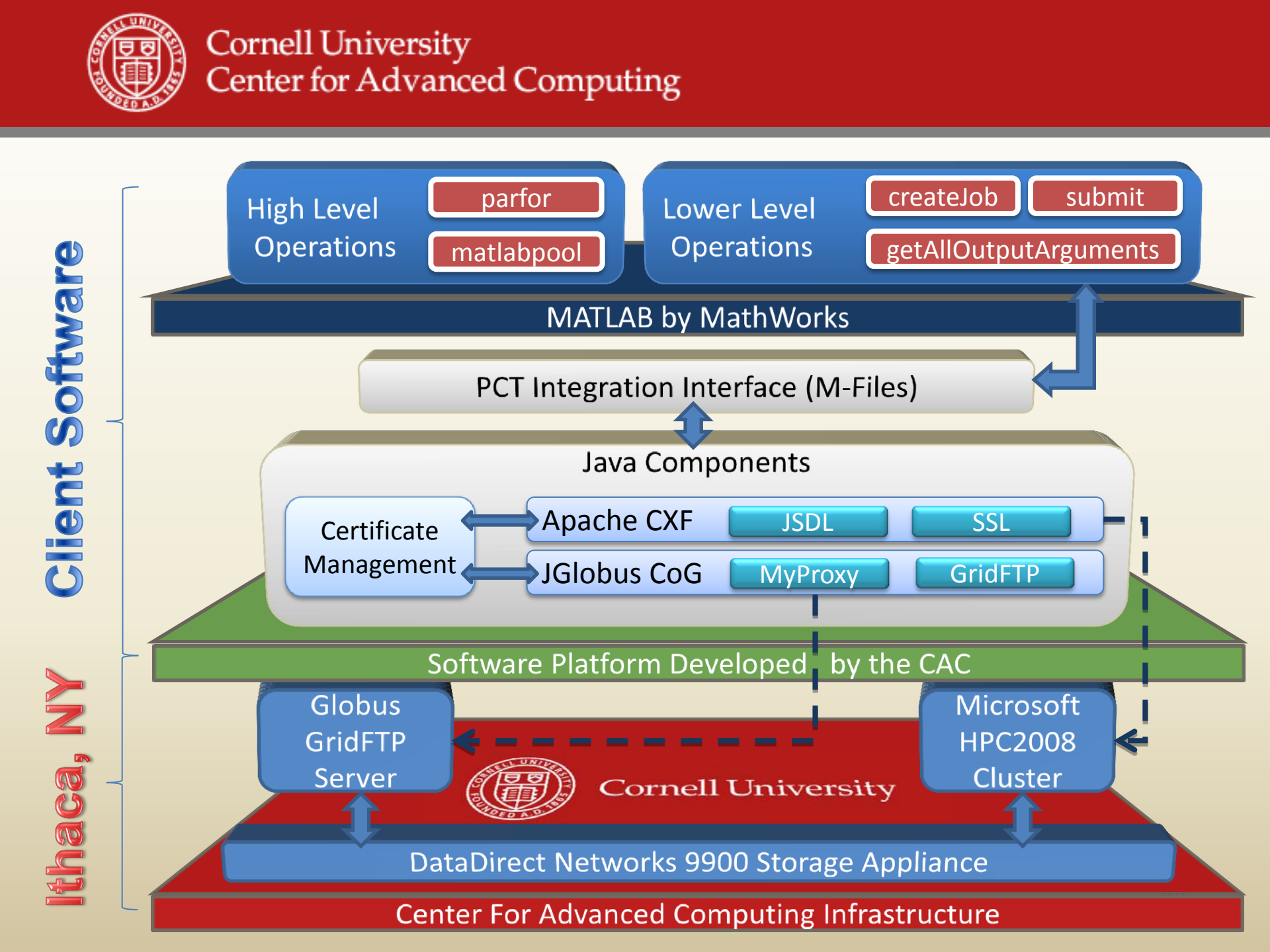
Hardware View

1. Retrieve certificate
2. Upload files to storage via GridFTP
3. Submit job to run MATLAB workers on cluster
4. Download files via GridFTP



Software View

- File movement and job submission interactions are largely hidden by software integrated with MATLAB.
- CAC's client code for MATLAB is a mix of Java and M-files that enable access to the HPC cluster directly from your MATLAB client through the PCT "generic scheduler" interface.
- Client code communicates as needed with server-side software at CAC to run distributed and parallel jobs on the HPC cluster's 64 CPU cores and 8 GPUs.


Cornell University
Center for Advanced Computing
Client Software
Ithaca, NY
High Level Operations
parfor
matlabpool
Lower Level Operations
createJob
submit
getAllOutputArguments
MATLAB by MathWorks
PCT Integration Interface (M-Files)
Java Components
Certificate Management
Apache CXF
JSDL
SSL
JGlobus CoG
MyProxy
GridFTP
Software Platform Developed by the CAC
Globus GridFTP Server
Cornell University
Microsoft HPC2008 Cluster
DataDirect Networks 9900 Storage Appliance
Center For Advanced Computing Infrastructure

A Note on the Platform

- The compute nodes that run your MATLAB jobs are running Windows HPC 2008 (64 bit).
  - Your client need not be running on a Win64 platform.
  - Files requiring compilation might need to be recompiled on the HPC cluster; a utility is provided for mex files, e.g.
  - MATLAB is resilient to paths with the wrong direction of slashes, but the difference can cause problems.
    - C:\Users\naw47\myfiles\this.dat ← Windows path
    - /home/naw47/myfiles/this.dat ← Mac, Linux path

Support

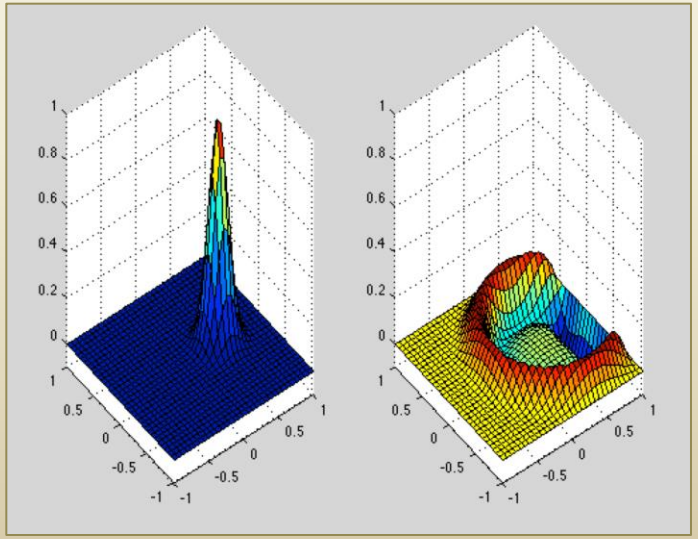
- A subscription comes with basic support to help you get started (contact help@cac.cornell.edu).
  - The basic rate allows CAC to recover hardware and software maintenance costs.
- You have the option to add more extensive consulting support to your subscription.
  - Troubleshooting
  - Guidance on optimizing your application
  - General help with parallel MATLAB

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

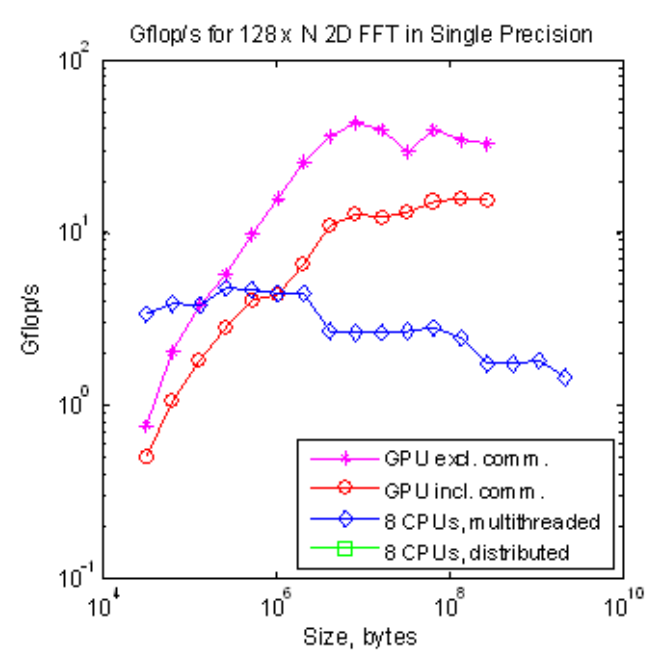


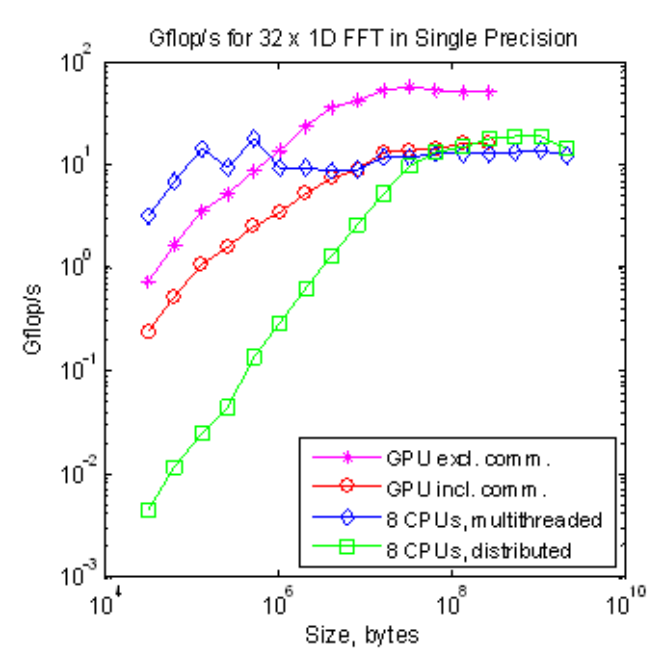
GPGPU in MATLAB: Fast and Easy

- 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);
```

GPU Excels at Large FFTs





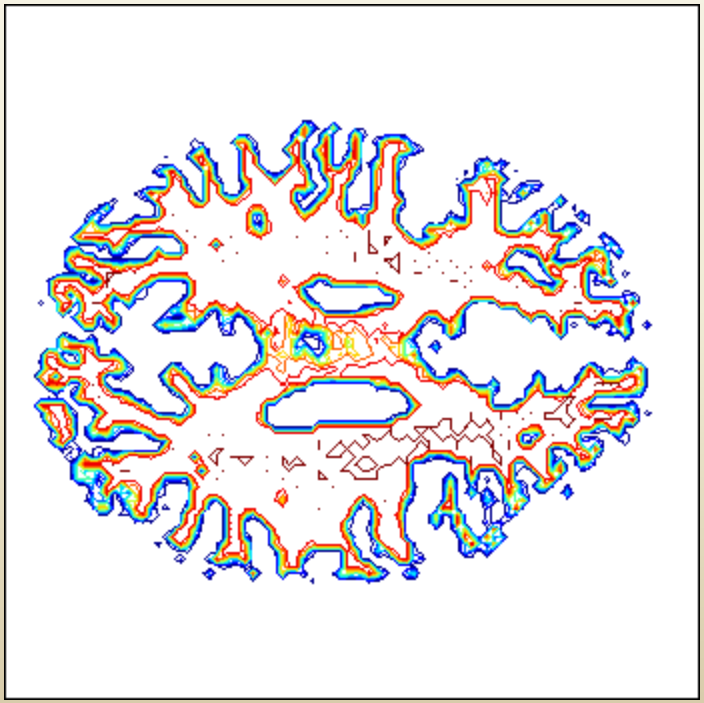
- 2D FFT > 8 MB can be 9x faster on GPU (including data transfers), but array of 1D FFTs is equally fast on 8 cores
- Limited to 256 MB due to bug in cuFFT 3.1; fixed in 3.2

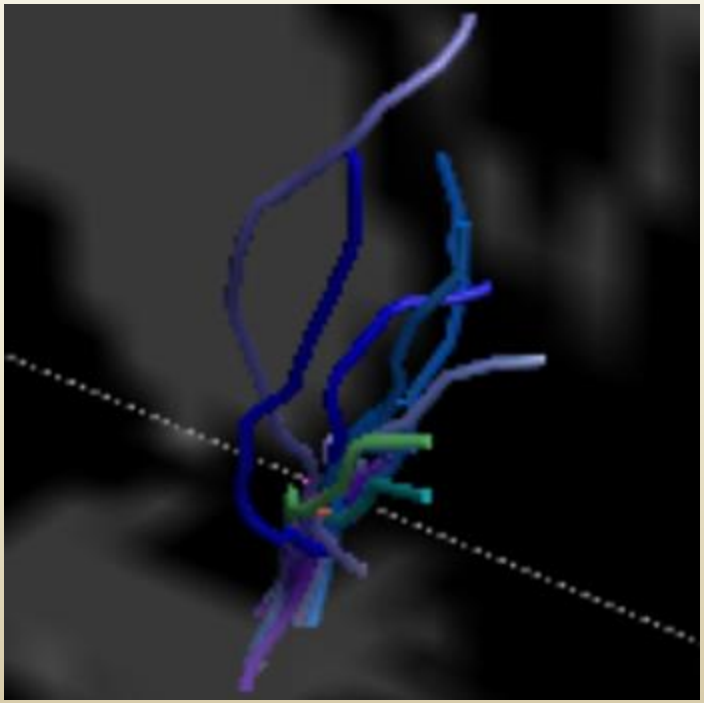
Analysis of MRI Brain Scans

- 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

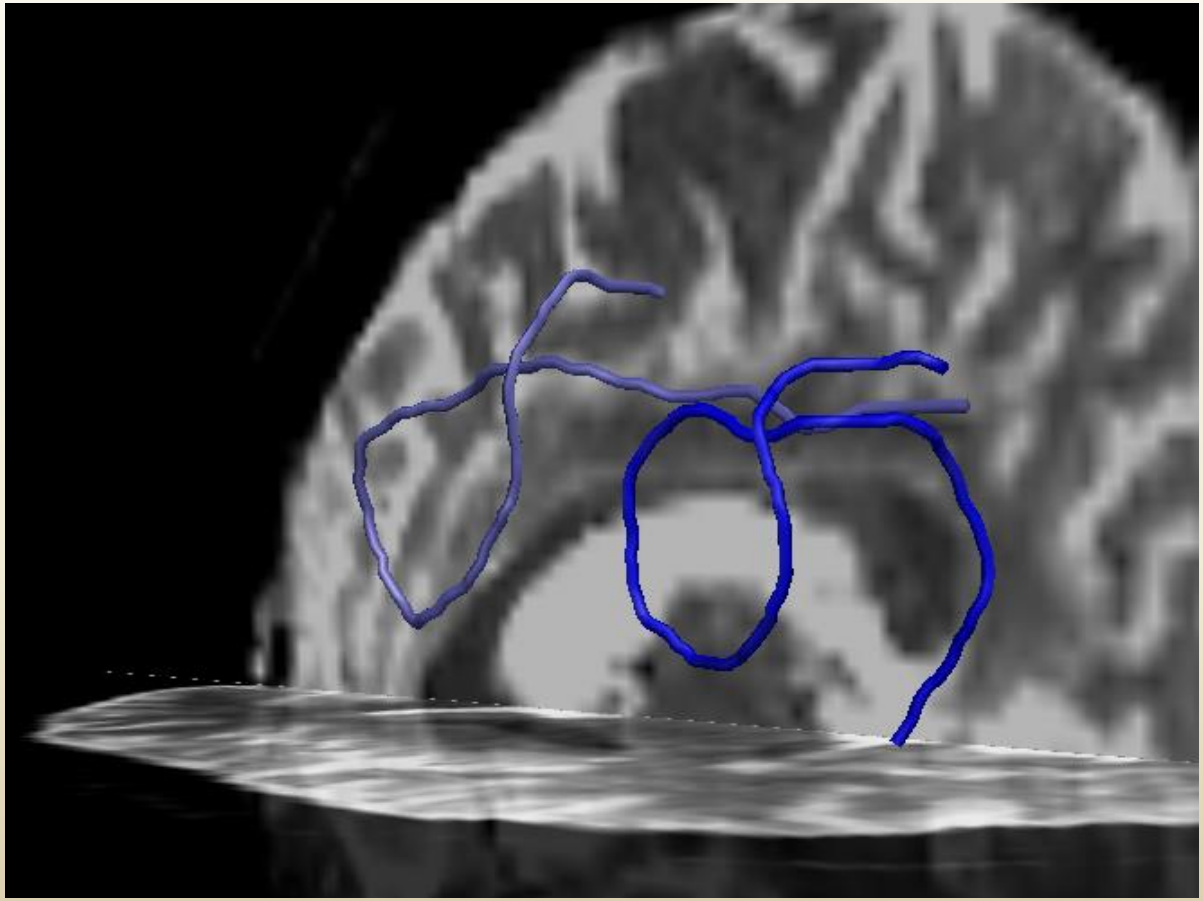
- 3D MRI scans to map the brain's white matter
- Fiber tracts to show lines of preferential diffusion





Need for Computational Power

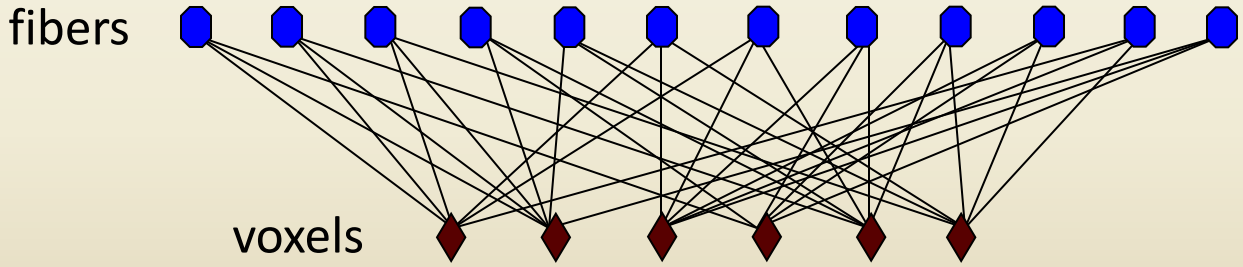
- Problem: long, spurious fibers arise in first-pass analysis
- Solution: use MATLAB to re-weight fibers according to importance in connections



Examples of improbable fibers eliminated by analysis

Connections in a Bipartite Graph

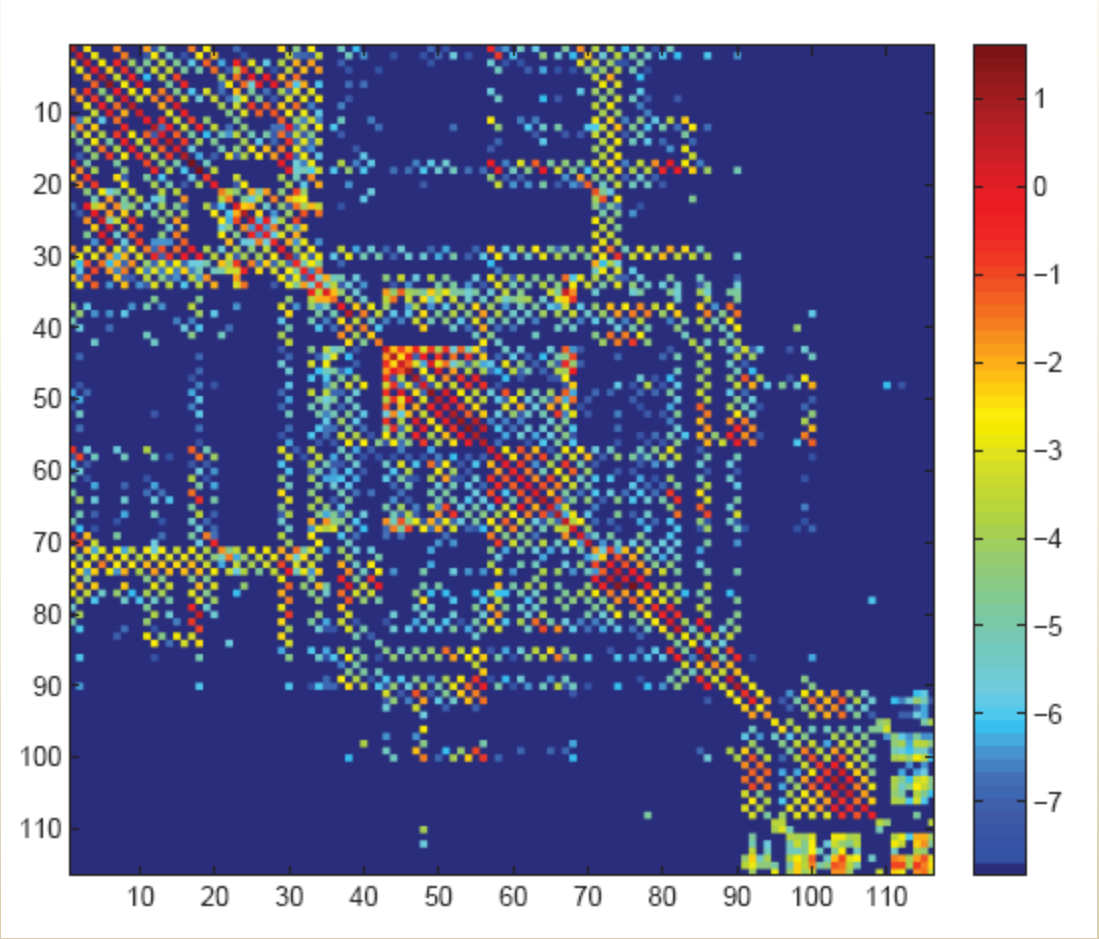
- Ivković and Raj (2010) developed a message-passing optimization procedure to solve the weighting problem
- Operates on a bipartite graph: nodes = fibers + voxels, edge weights = connection strength



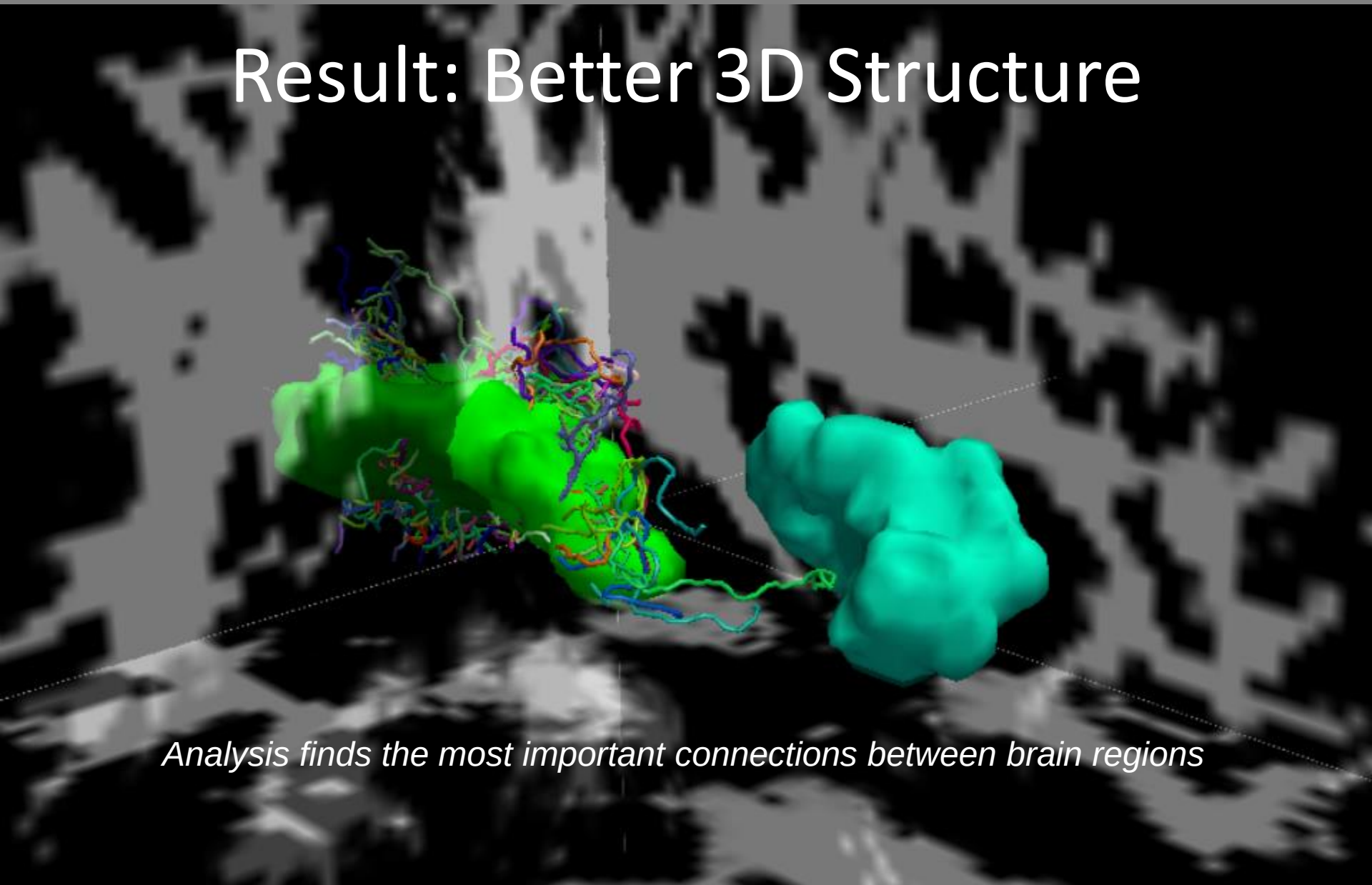
- MATLAB computations at each voxel are independent of all other voxels, likewise for fibers; inherently parallel

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



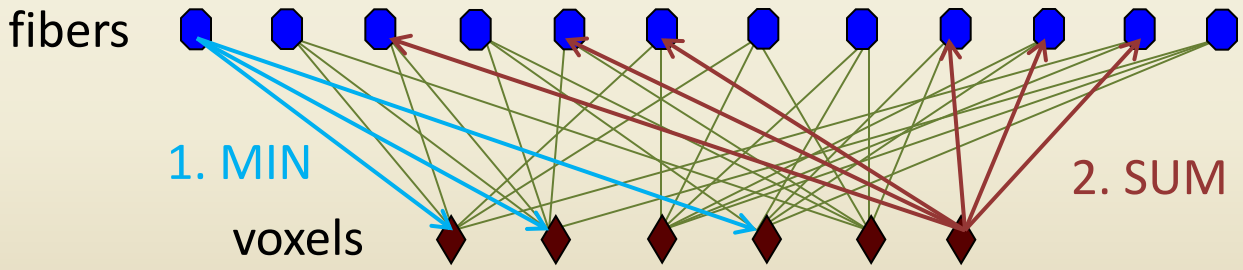
Result: Better 3D Structure



Analysis finds the most important connections between brain regions

Message-Passing Algorithm

- Iterative procedure also known as “min-sum”
- Fiber-centric step: for each fiber, find the minimum of all its edge weights; reset the edges to that value (or to the second smallest value, if already at min)



- Voxel-centric step: for each voxel, sum up its current edge weights; distribute WM value back proportionately

Round One: Parallelization

- Min/sum can be done independently for each fiber/voxel
- Loops can be converted into parfor-loops
  - On 8 cores: 375 sec/iteration shrinks to 136 sec/iteration
  - After pre-packing the WM data structure to eliminate voxels not traversed by at least one fiber: 42 sec/iteration
  - After eliminating redundant searches via improvements to indexing, *and removing parfor*: 32 sec/iteration!
- A better algorithm with good memory locality beats parallelization!

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

Round Two: Vectorization

- So, just throw everything into one giant matrix?
  - Problem #1: row-major ordering = bad stride
  - Problem #2: mixing of dissimilar data = poor data locality
  - Due to these problems, the initial matrix-based version of the serial min-sum algorithm ran slower, 53 sec/iteration
- Initial optimization steps were easy…
  - Make *columns* receive all edge weights (messages)
  - Pull out only necessary info and store in separate, condensed matrices

Round Three: CPU Optimization

- Tighten up memory utilization by grouping fibers and voxels according to numbers of coordinating edges
  - Different matrices for fibers that connect to 2, 3, 4… edges
  - Yields full columns in the matrix for all 2-edge fibers, etc.
- Resulting code is much more complex
  - New inner for-loops over fiber count, voxel count
  - Challenge to construct the necessary indexing
- Excellent performance in the end: **0.25 sec/iteration**
- Good outcome, but days and days of work

Round Four: GPU Optimization

- Since R2011b, min and sum will work on gpuArrays!
- Go back to 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

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