Parallel and Distributed Computing with MATLAB

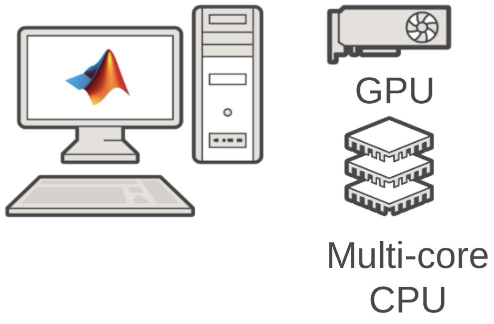
Ángel Sierra % asierra@mathworks.com
Carlos Sanchis % csanchis@mathworks.com
Lucas García % lgarcia@mathworks.com

Organized with the collaboration of the Laboratory for Advanced Computing at University of Coimbra

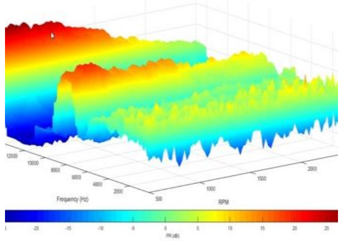
Why parallel computing?

- Save time and tackle increasingly complex problems
  - Reduce computation time by using available compute cores and GPUs

- Why parallel computing with MATLAB and Simulink?
  - Accelerate workflows with minimal to no code changes to your original code
  - Scale computations to clusters and clouds
  - Focus on your engineering and research, not the computation



Benefits of parallel computing



Automotive Test Analysis
Validation time sped up 2X
Development time reduced 4 months

Discrete-Event Model of Fleet Performance
Simulation time sped up 20X
Simulation time reduced from months to hours

Heart Transplant Study
Process time sped up 6X
4 week process reduced to 5 days

Calculating Derived Market Data
Updates sped up 8X
Updates reduced from weeks to days

Accelerating and Parallelizing MATLAB Code

- Optimize your serial code for performance
- Analyze your code for bottlenecks and address most critical items
- Include compiled languages
- Leverage parallel computing tools to take advantage of additional computing resources

https://www.mathworks.com/discovery/matlab-acceleration.html

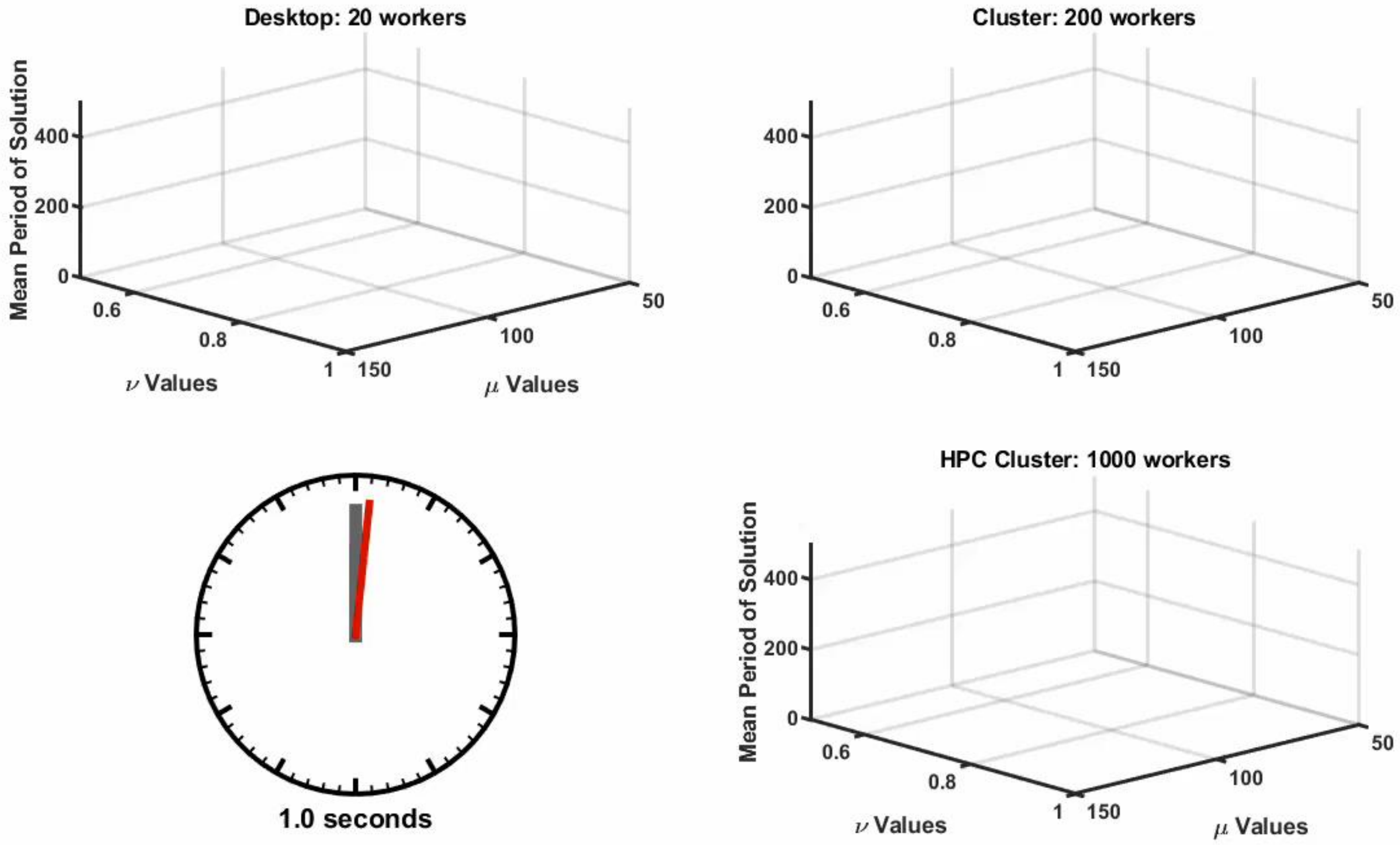
Run MATLAB on multicore machines

- Built-in multithreading (implicit)
  - Automatically enabled in MATLAB
  - Multiple threads in a single MATLAB computation engine
  - Functions such as `fft`, `eig`, `svd`, and `sort` are multithreaded in MATLAB

- Parallel computing using explicit techniques
  - Multiple computation engines (workers) controlled by a single session
  - Perform MATLAB computations on GPUs
  - High-level constructs to let you parallelize MATLAB applications
  - Scale parallel applications beyond a single machine to clusters and clouds

Compute 40,000 iterations (mean of 2.2 seconds per iteration)

van der Pol Equation study with `parfor`

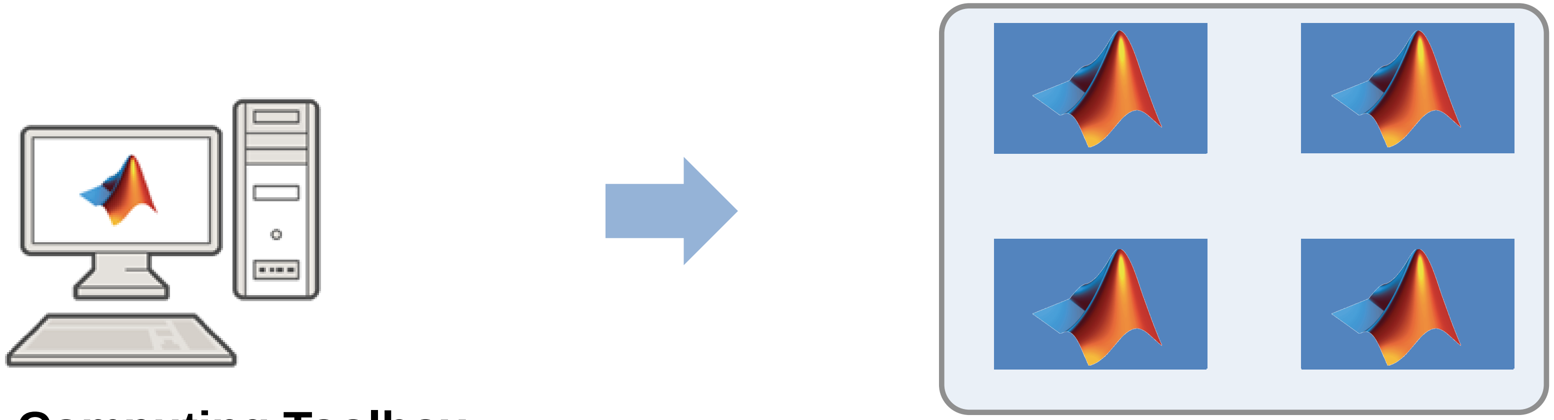


Agenda

- Utilizing multiple cores on a desktop computer
- Scaling up to cluster and cloud resources
- Tackling data-intensive problems on desktops and clusters
- Accelerating applications with NVIDIA GPUs
- Summary and resources

Parallel Computing Paradigm

Multicore Desktops

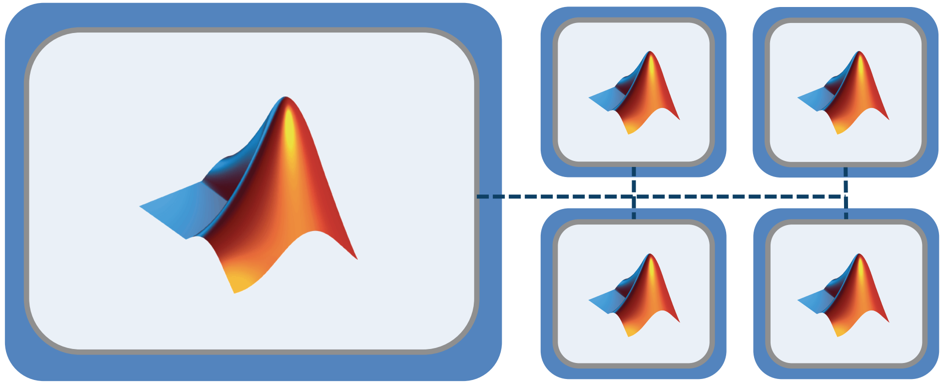


Parallel Computing Toolbox

Run multiple iterations on a process or thread level

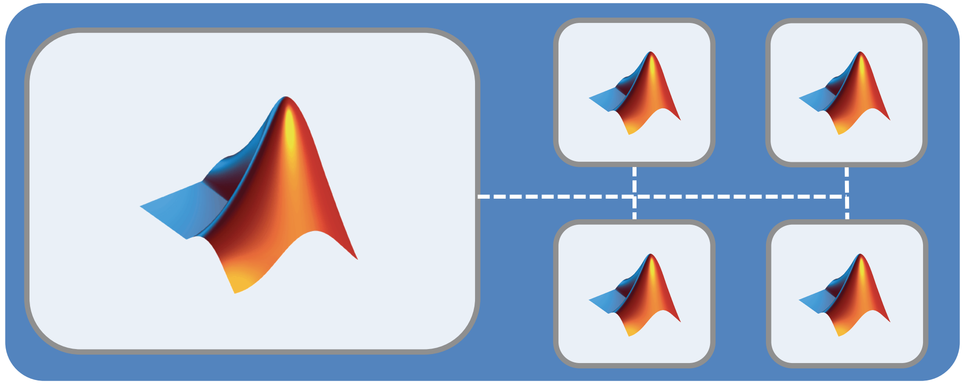
```
>> parpool("local")
```

Workers run as their own process



```
>> parpool("threads")
```

Workers run as threads in main MATLAB process and share memory



[Choose between thread-based and process-based environments]

Accelerating MATLAB and Simulink Applications

Ease of Use

Parallel-enabled toolboxes

`('UseParallel', true)`

Common programming constructs

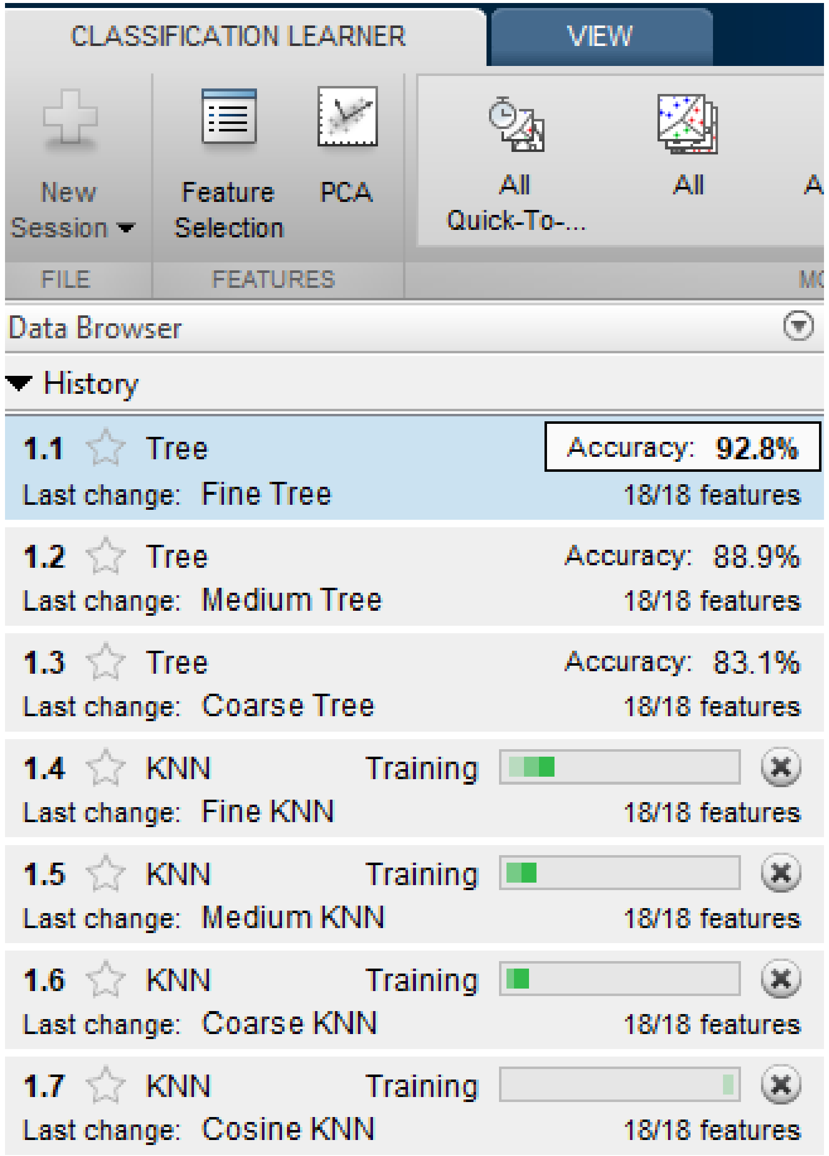
Advanced programming constructs

Greater Control

Demo: Classification Learner App

Analyze sensor data for human activity classification

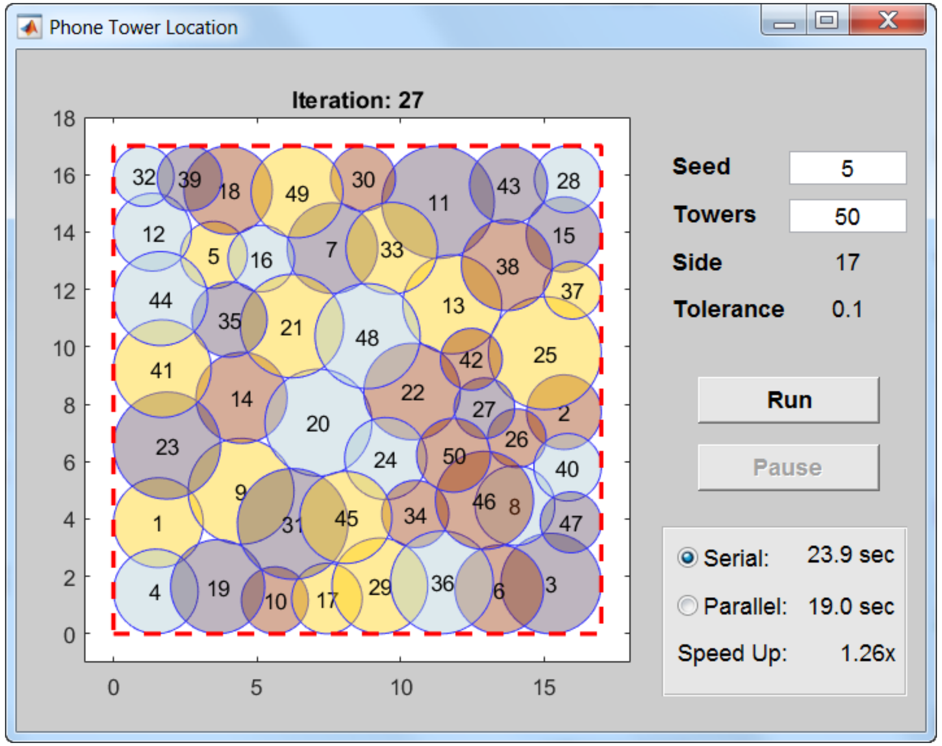
- Objective: visualize and classify cellphone sensor data of human activity
- Approach:
  - Load human activity dataset
  - Leverage multicore resources to train multiple classifiers in parallel



Demo: Cell Phone Tower Optimization

Using Parallel-Enabled Functions

- Parallel-enabled functions in Optimization Toolbox
- Set flags to run optimization in parallel
- Use pool of MATLAB workers to enable parallelism



Parallel-enabled Toolboxes (MATLAB® Product Family)

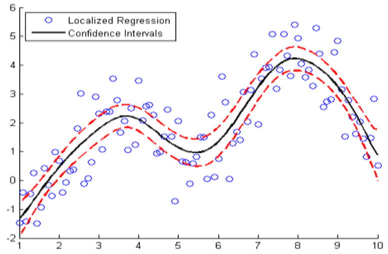
Enable parallel computing support by setting a flag or preference

Image Processing

Batch Image Processor, Block Processing, GPU-enabled functions

Statistics and Machine Learning

Resampling Methods, k-Means clustering, GPU-enabled functions

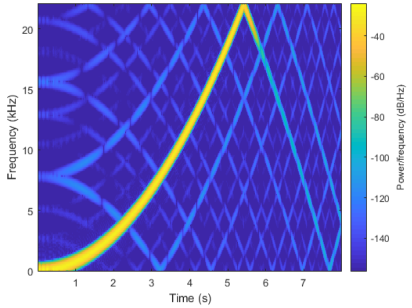


Deep Learning

Deep Learning, Neural Network training and simulation

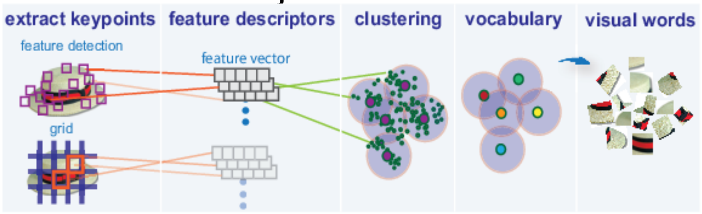
Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



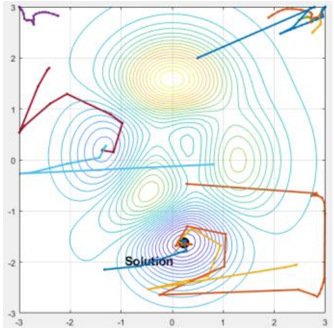
Computer Vision

Bag-of-words workflow, object detectors



Optimization & Global Optimization

Estimation of gradients, parallel search



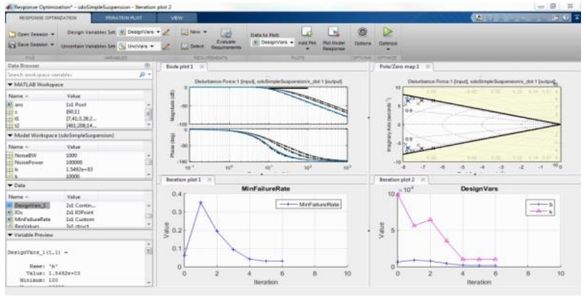
Other Parallel-enabled Toolboxes

Parallel-enabled Toolboxes (Simulink® Product Family)

Enable parallel computing support by setting a flag or preference

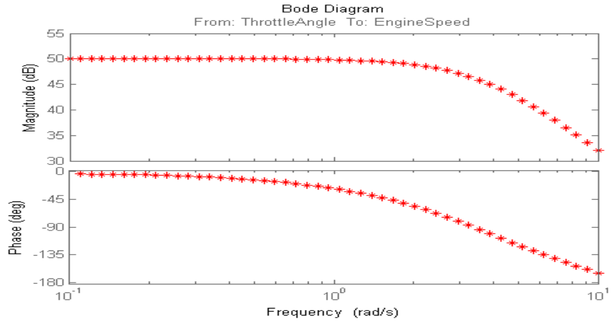
Simulink Design Optimization

Response optimization, sensitivity analysis, parameter estimation



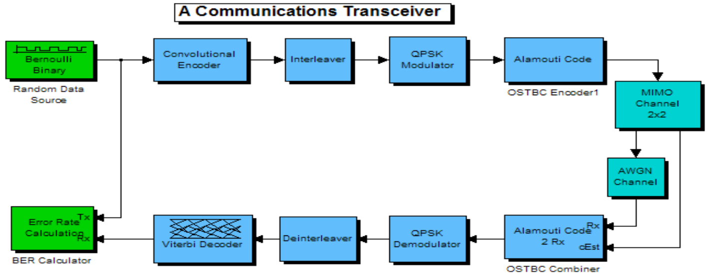
Simulink Control Design

Frequency response estimation



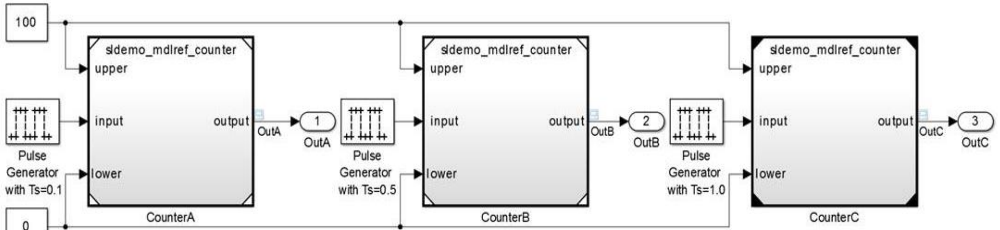
Communication Systems Toolbox

GPU-based System objects for Simulation Acceleration



Simulink/Embedded Coder

Generating and building code



[Other parallel-enabled Toolboxes]

Accelerating MATLAB and Simulink Applications

Ease of Use

Parallel-enabled toolboxes

Common programming constructs
(`parfor, batch`)

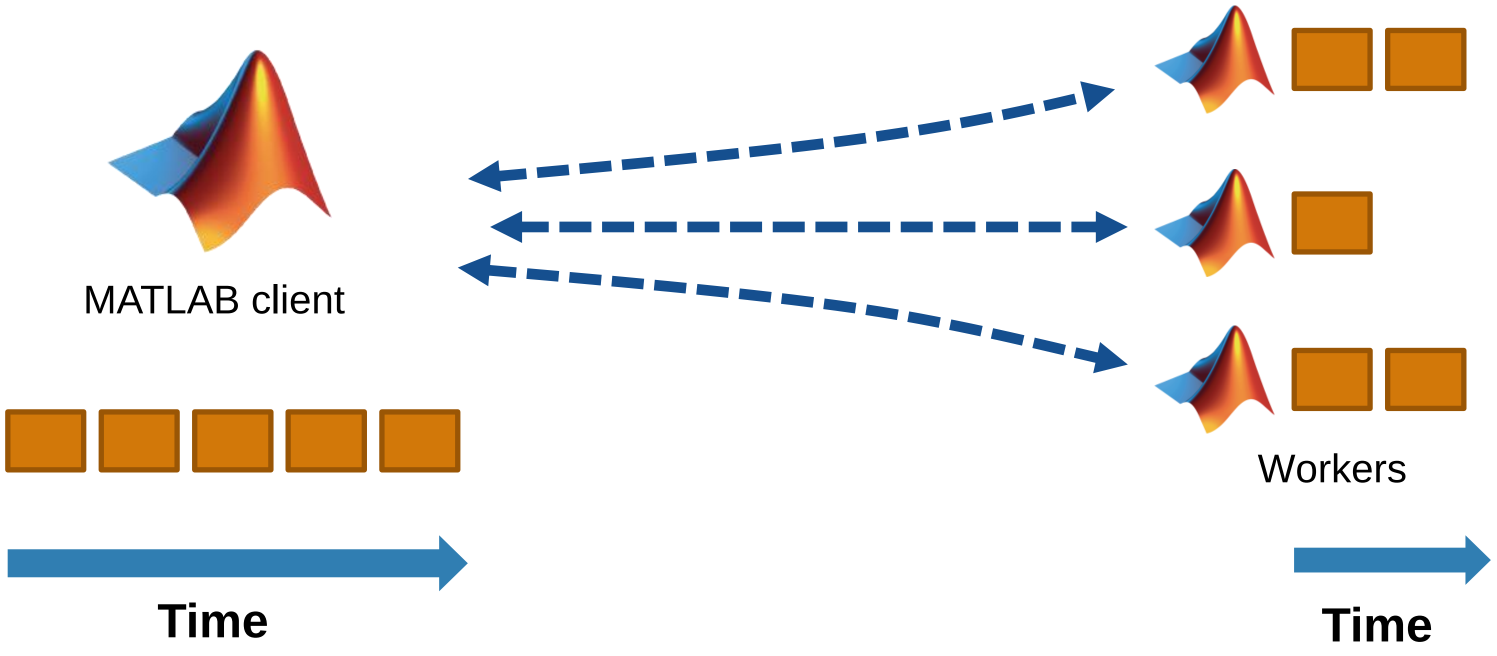
Advanced programming constructs

Greater Control

Explicit Parallelism: Independent Tasks or Iterations

Simple programming constructs: `parfor`

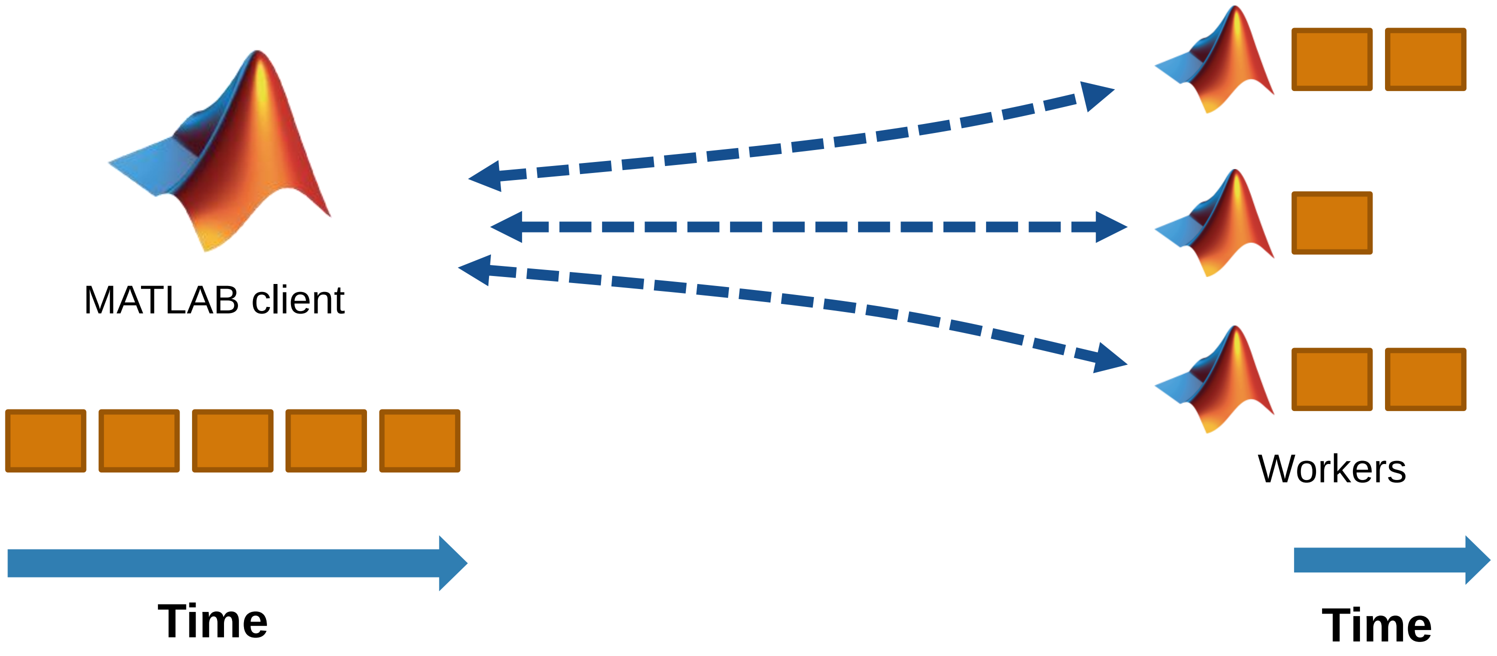
- Examples: parameter sweeps, Monte Carlo simulations
- No dependencies or communications between tasks



Explicit Parallelism: Independent Tasks or Iterations

```
for i = 1:5
      y(i) = myFunc(myVar(i));
end
```

```
parfor i = 1:5
      y(i) = myFunc(myVar(i));
end
```



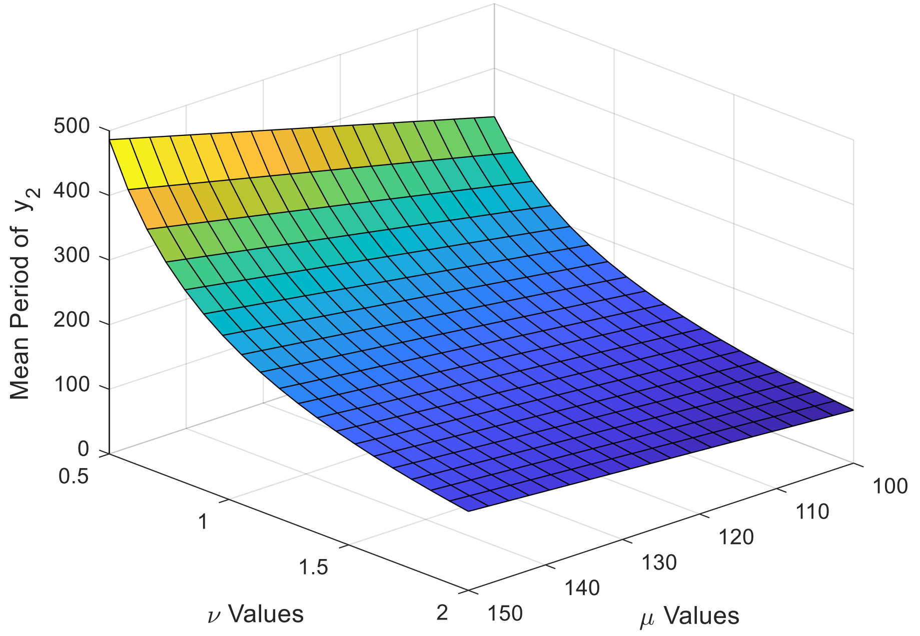
Demo: parameter sweep for van der Pol oscillator

Embarrassingly parallel tasks in MATLAB

- System of ODEs

$$\dot{y}_1 = \nu y_2$$
$$\dot{y}_2 = \mu(1 - y_1^2)y_2 - y_1$$

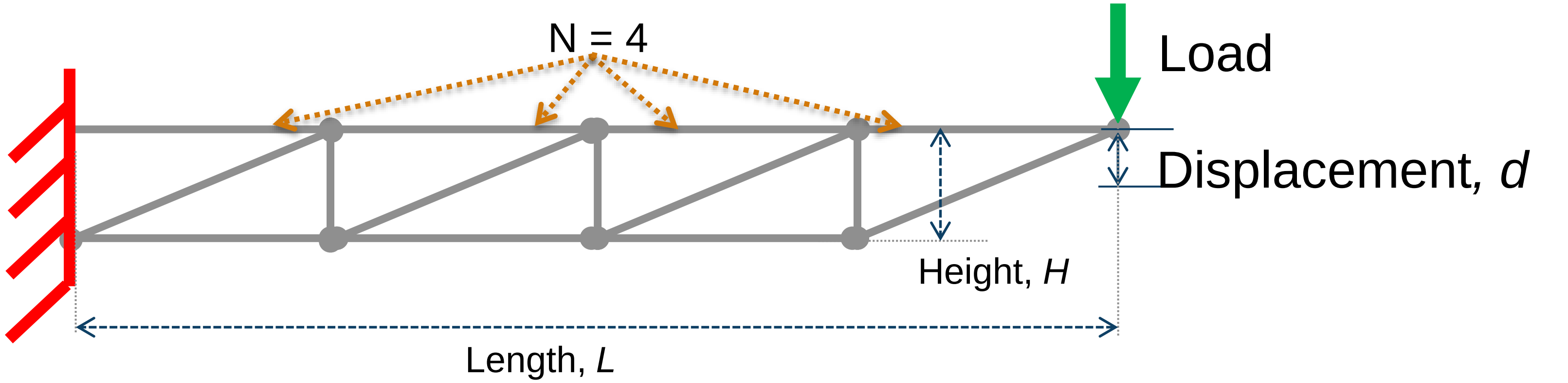
- Compute mean period of y
- Use `parfor` to study impact of ν, μ

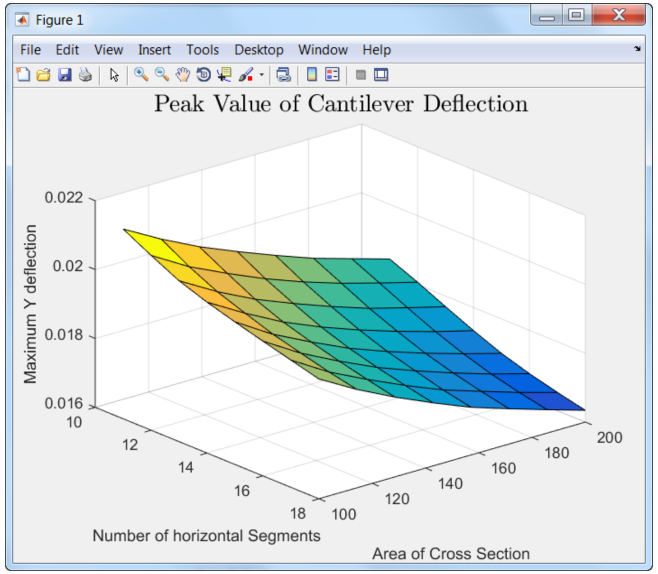


Demo: Parameter Sweep

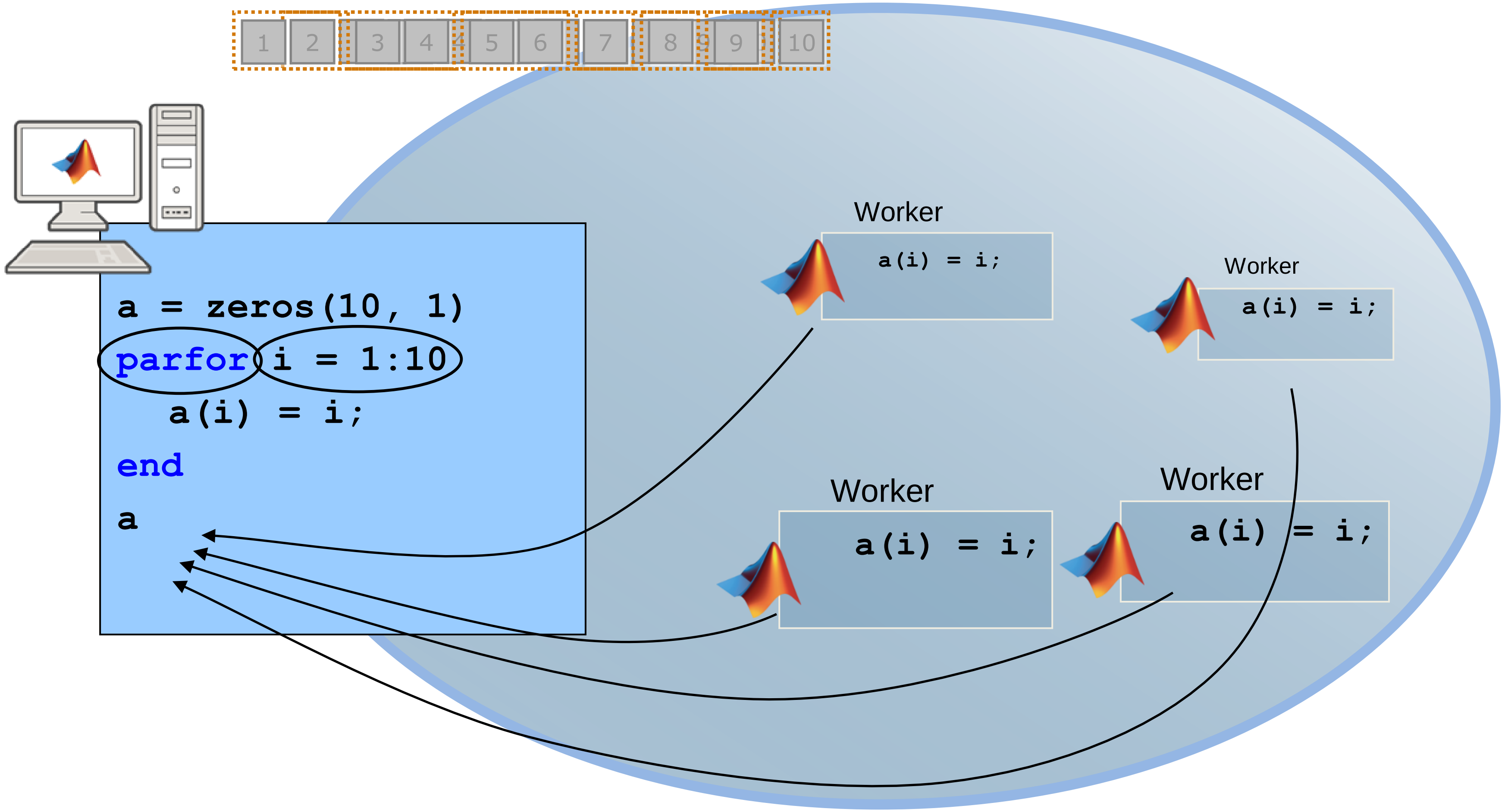
Embarrassingly parallel tasks in MATLAB

- Parameter sweep
  - Truss under a dynamic load
  - Sweeping over cross-sectional area and number of elements



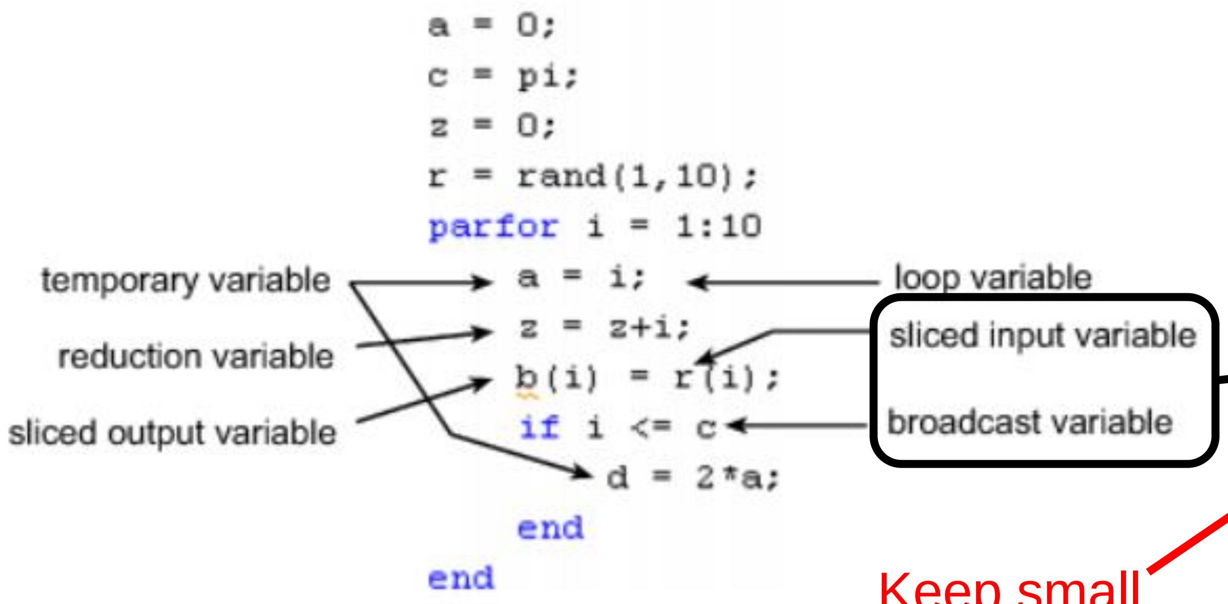


Mechanics of `parfor` Loops




```
a = zeros(10, 1)
parfor i = 1:10
  a(i) = i;
end
a
```

Optimizing `parfor`



Use more

Keep small

Type	Category
sliced input	input
broadcast	input
reduction	output
sliced output	output
loop	only exist on worker
temporary	only exist on worker

[Troubleshooting variables in parfor-loops]

Parallelism using `parfor`

```
a = zeros(5, 1);
b = pi;
parfor i = 1:5
    a(i) = i + b;
end
disp(a)
```

No warnings found.
(Using Default Settings)

```
1  a = zeros(5, 1);
2  b = pi;
3  parfor i = 2:6
4      a(i) = a(i-1) + b;
5  end
6  disp(a)
```

Line 4: In a PARFOR loop, variable 'a' is indexed in different ways, potentially causing dependencies between iterations.

Tips for Leveraging `parfor`

- Consider creating smaller arrays on each worker versus one large array prior to the parfor loop
- Take advantage of parallel.pool.Constant to establish variables on pool workers prior to the loop
- Encapsulate blocks as functions when needed

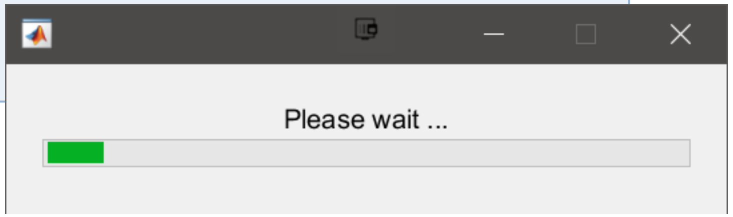
[Understanding parfor]

Execute additional code as iterations complete

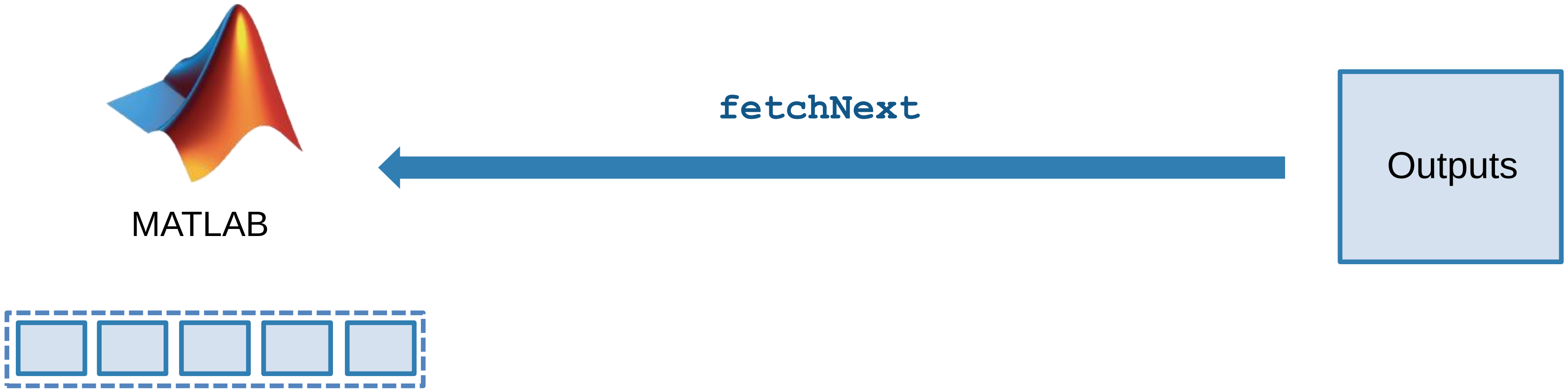
- Send data or messages from parallel workers back to the MATLAB client

- Retrieve intermediate values and track computation progress

```
function a = parforWaitbar

D = parallel.pool.DataQueue;
h = waitbar(0, 'Please wait ...');
afterEach(D, @nUpdateWaitbar)

N = 200;
p = 1;

parfor i = 1:N
    a(i) = max(abs(eig(rand(400))));
    send(D, i)
end

    function nUpdateWaitbar(~)
        waitbar(p/N, h)
        p = p + 1;
    end
end
```



Execute functions in parallel asynchronously using `parfeval`



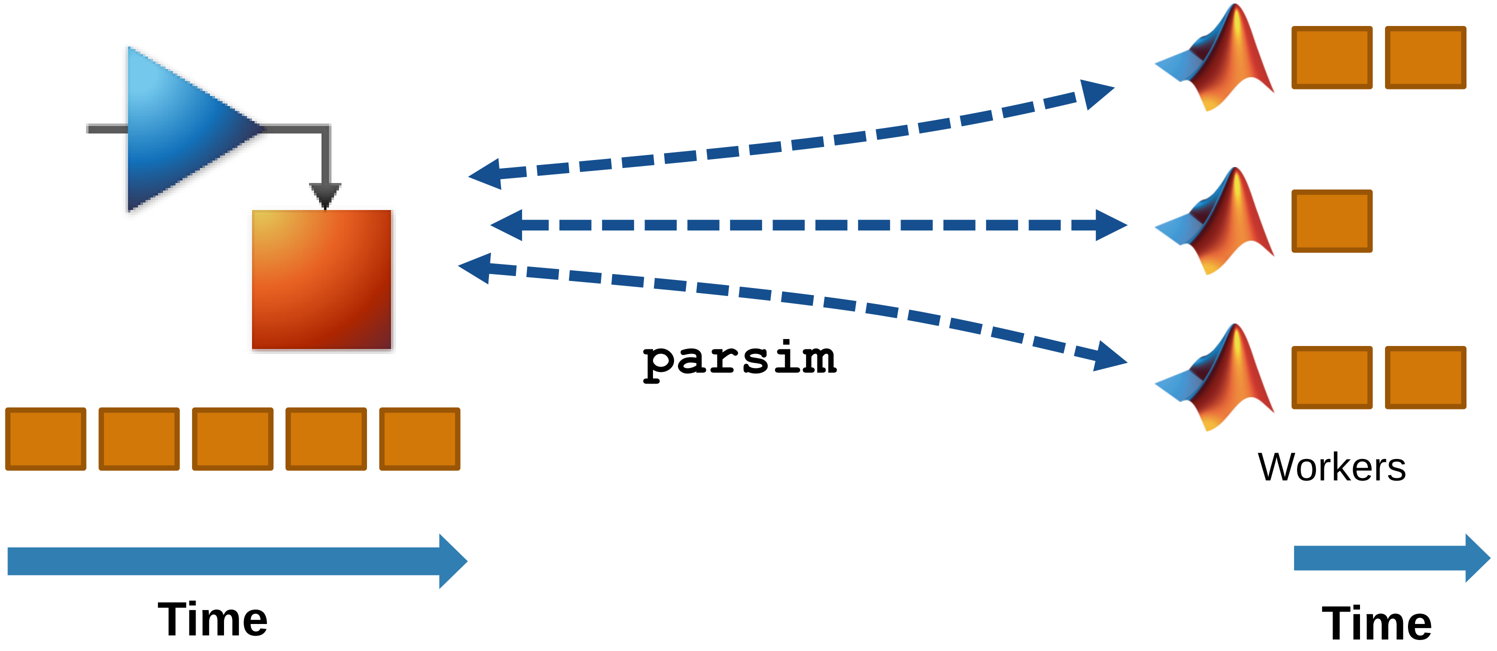
- Asynchronous execution on parallel workers
- Useful for “needle in a haystack” problems

```
for idx = 1:10
   f(idx) = parfeval(@magic,1,idx);
end

for idx = 1:10
   [completedIdx,value] = fetchNext(f);
   magicResults{completedIdx} = value;
end
```

Run Multiple Simulations in Parallel with `parsim`

- Run independent Simulink simulations in parallel using the `parsim` function

```
for i = 10000:-1:1
    in(i) = Simulink.SimulationInput(my_model);
    in(i) = in(i).setVariable(my_var, i);
end
out = parsim(in);
```



Virgin Orbit Simulates LauncherOne Stage Separation Events

Challenge

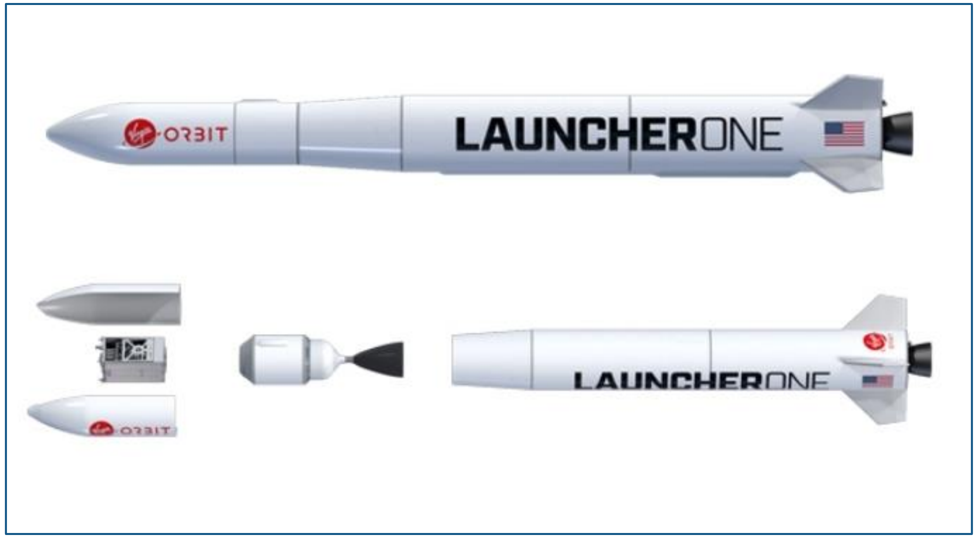
Simulate separation events for LauncherOne spacecraft

Solution

- Use MATLAB, Simulink, and Simscape Multibody to model components and automate Monte Carlo simulations
- Used Parallel Computing Toolbox to run simulations in parallel on multicore processors

Results

- Simulations completed 10 times faster
- Simulation set up times cut by up to 90%
- Hardware designs informed by simulation results



Virgin Orbit's LauncherOne vehicle assembled (top), with exploded view showing the fairing, payload, and first and second stages (bottom).

"With Simulink, we can employ simplifying assumptions and parallel processing to reduce simulation times from days to hours…Just as important, we can automate the simulations so they run in the background or overnight, and have the results waiting for us in the morning."

- Patrick Harvey, Associate Engineer at Virgin Orbit

Link to user story

`parsim` has simplified running parallel simulations

```
model = 'sldemo_suspn_3dof';
load_system(model);
parpool;

Cf                    = evalin('base', 'Cf');
Cf_sweep              = Cf*(0.05:0.1:0.95);
iterations            = length(Cf_sweep);
simout(iterations)    = Simulink.SimulationOutput;


spmd
    currDir = pwd;
    addpath(currDir);
    tmpDir = tempname;
    mkdir(tmpDir);
    cd(tmpDir);
    % Load the model on the worker
    load_system(model);
end

parfor idx=1:iterations
    set_param([model '/Road-Suspension Interaction'],'MaskValues',...
        {'Kf',num2str(Cf_sweep(idx)),'Kr','Cr'});
    simout(idx) = sim(model, 'SimulationMode', 'normal');
end

spmd
    cd(currDir);
    rmdir(tmpDir,'s');
    rmpath(currDir);
    close_system(model, 0);
end

close_system(model, 0);
delete(gcp('nocreate'));
```

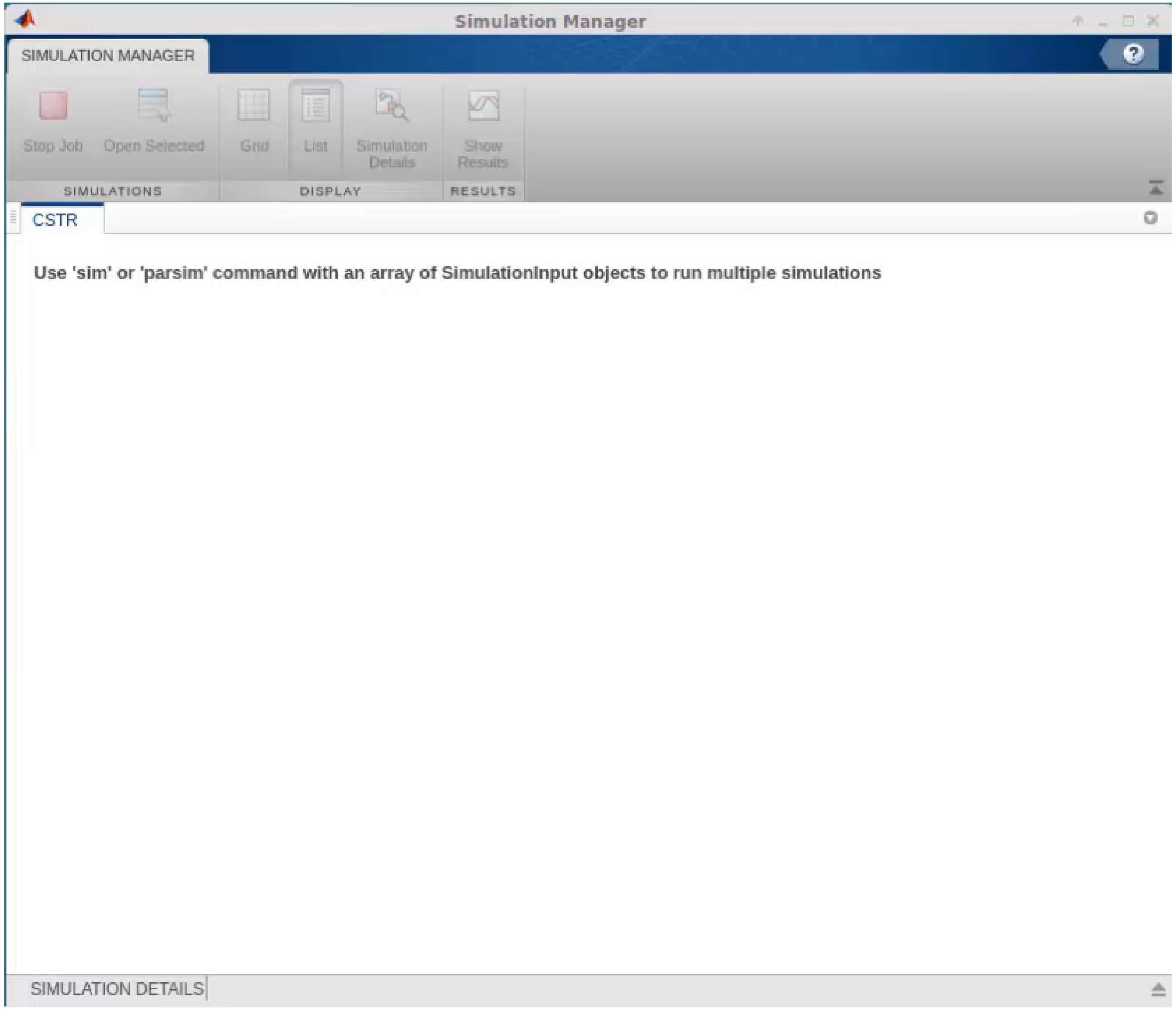
Pre-R2017a

```
mdl = 'sldemo_suspn_3dof';
load_system(mdl);

Cf_sweep = Cf*(0.05:0.1:0.95);
numSims = length(Cf_sweep);

in(1:numSims) = Simulink.SimulationInput(mdl);
for i = 1:numSims
    in(i) = setBlockParameter(in(i), [mdl '/Road-Suspension Interaction'], ...
        'Cf', num2str(Cf_sweep(i)));
end

out = parsim(in, 'ShowProgress', 'on');
```

Starting with R2017a

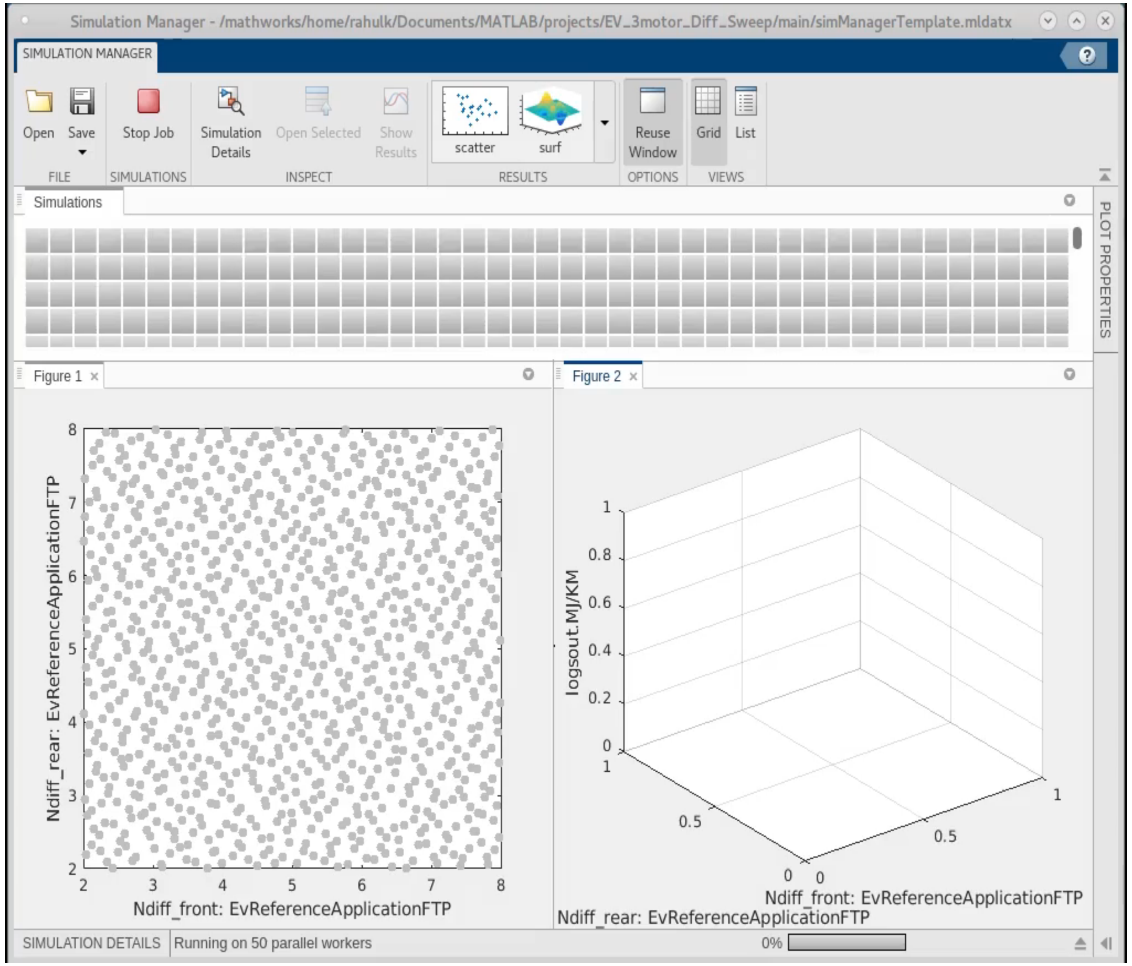
Monitor Multiple Simulations at Once with Simulation Manager

- View the progress of the simulations
- Examine simulation settings and diagnostics
- View simulation results in the **Simulation Data Inspector**.



Simulation Manager can also be used to inspect the variation in outputs with parameters

- Visualize simulation results as the simulations are running

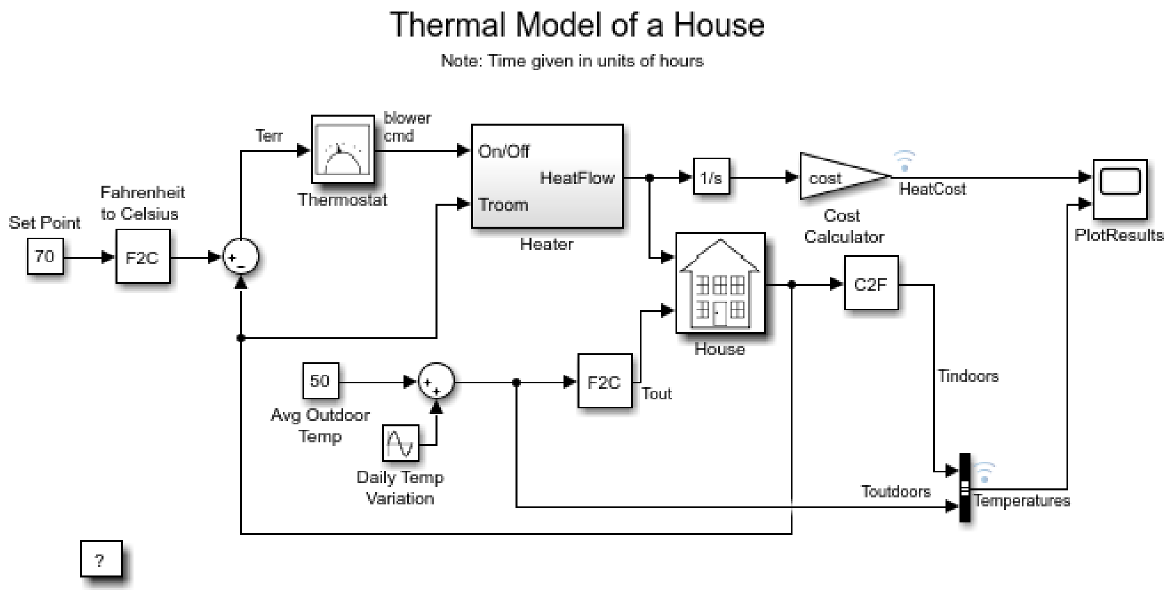


Demo: Thermal Model of House

Run parallel simulations with parsim

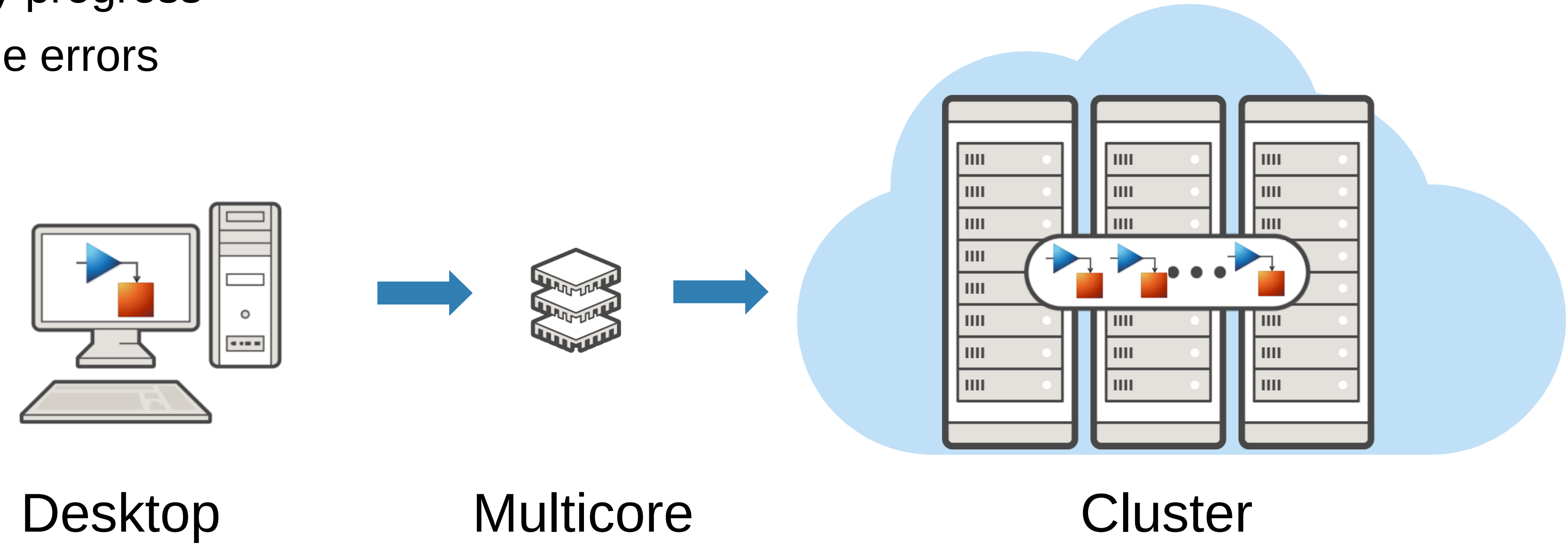
- Simulation of home heating cost by parameter sweeping (temperature set point)

- Use a SimulationInput object to change block parameter and run simulations in parallel



Benefits of using `parsim`

- Run multiple simulations on your machine or clouds and clusters
- Transfer base workspace variables to workers
- Automatically transfer all files to workers
- Automatically return file logging data
- Automatically manage build folders
- Display progress
- Manage errors



Accelerating MATLAB and Simulink Applications

Ease of Use

Parallel-enabled toolboxes

Common programming constructs

Advanced programming constructs
(`spmd, createJob, labSend,..`)

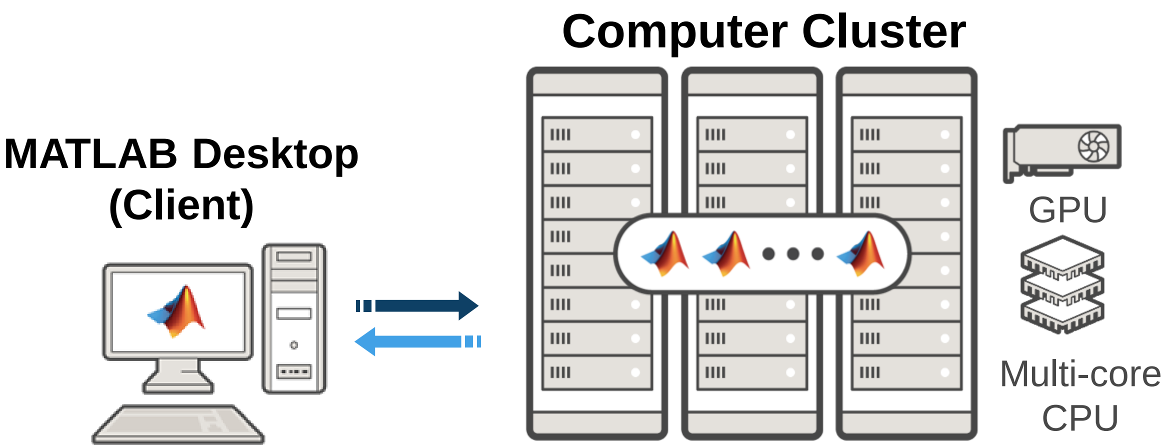
Greater Control

Agenda

- Utilizing multiple cores on a desktop computer
- Scaling up to cluster and cloud resources
- Tackling data-intensive problems on desktops and clusters
- Accelerating applications with NVIDIA GPUs
- Summary and resources

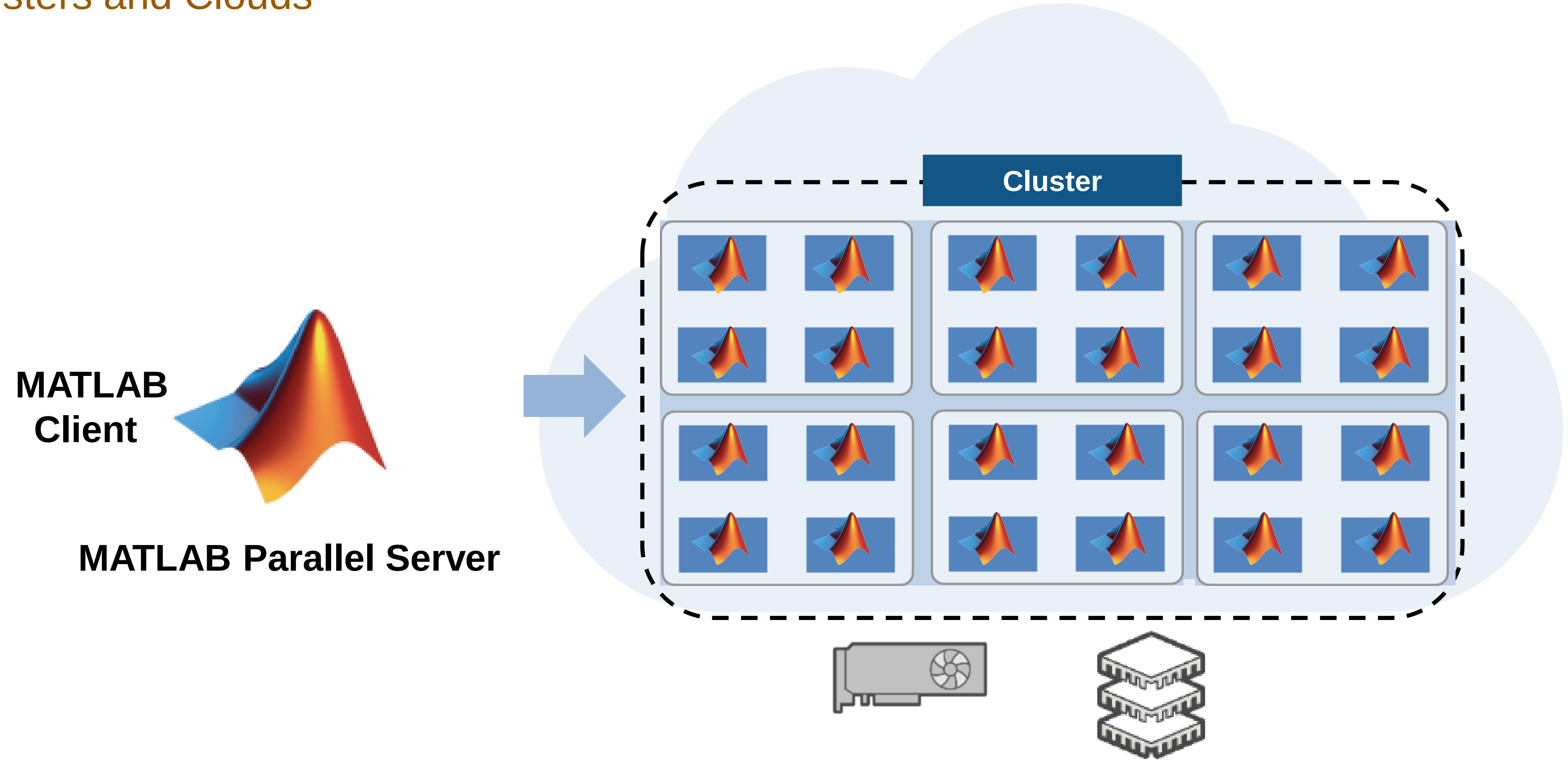
Take Advantage of Cluster Hardware

- Offload computation:
  - Free up desktop
  - Access better computers
- Scale speed-up:
  - Use more cores
  - Go from hours to minutes
- Scale memory:
  - Utilize tall arrays and distributed arrays
  - Solve larger problems without re-coding algorithms



Parallel Computing Paradigm

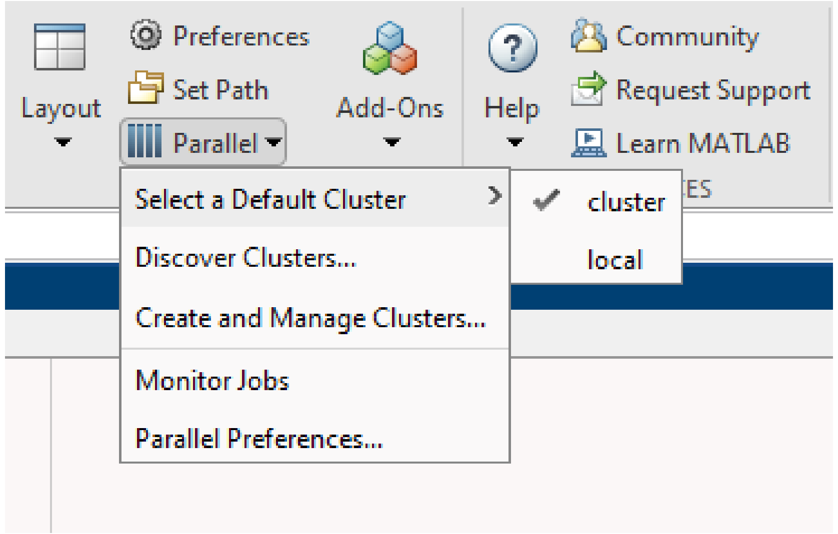
Clusters and Clouds



Scale to clusters and clouds

With MATLAB Parallel Server, you can…

- Use more hardware with minimal code change
- Submit to on-premise or cloud clusters
- Support cross-platform submission
  – Windows client to Linux cluster



Aberdeen Asset Management Implements Machine Learning–Based Portfolio Allocation Models in the Cloud

Challenge

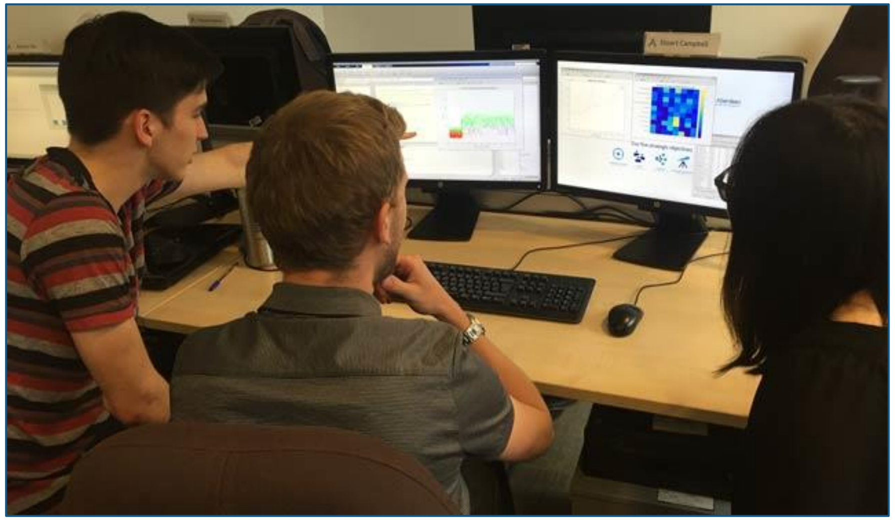
Improve asset allocation strategies by creating model portfolios with machine learning techniques

Solution

- Use MATLAB to develop classification tree, neural network, and support vector machine learning models
- Enable rapid results using MATLAB Parallel Server to run the models in the Azure cloud

Results

- Processing times cut from 24 hours to 3
- Quickly validate results by iterating multiple machine learning models



Interns using MATLAB at Aberdeen Asset Management.

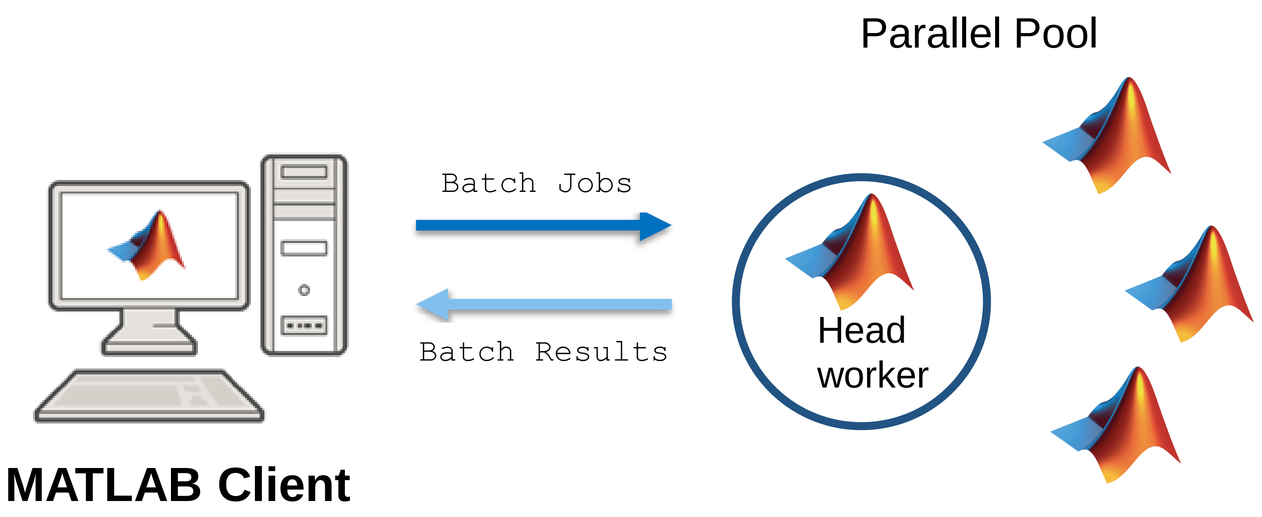
"Our processing times went from 24 hours to 3 when we started running on the Azure cloud with MATLAB Parallel Server," notes Mann. "Because the job scheduler is integrated into MATLAB, it's easy to take advantage of parallel computing just by opening a pool and using parfor loops."
- Emilio Llorente-Cano, senior investment strategist at Aberdeen Asset Management

Link to user story

`batch` Simplifies Offloading Serial Computations

Submit jobs from MATLAB and free up MATLAB for other work

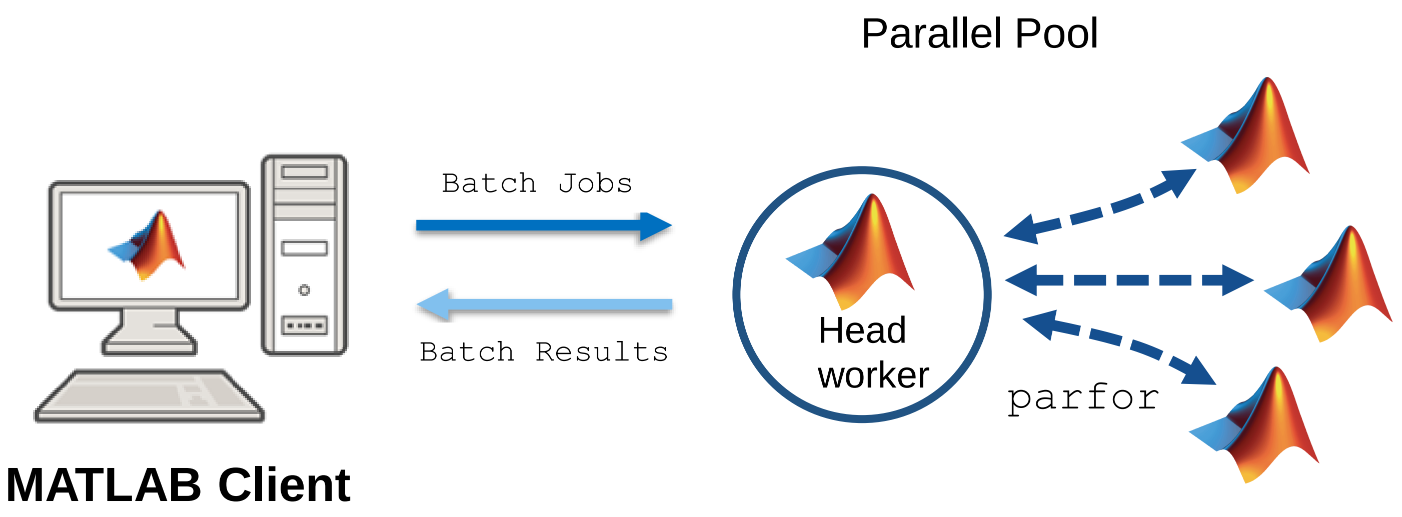
```
job = batch('myfunc');
```



`batch` Simplifies Offloading Serial Computations

Submit jobs from MATLAB, free up MATLAB for other work, access results later

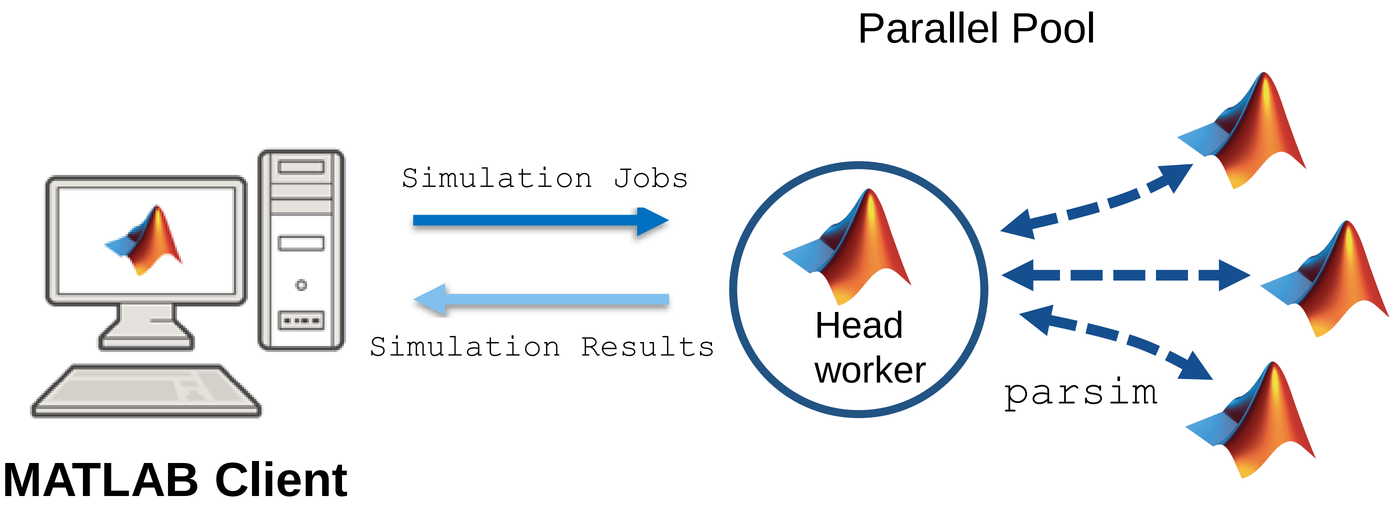
```
job = batch('myfunc','Pool',3);
```



`batchsim` Simplifies Offloading Simulations

Submit jobs from MATLAB to offload and run Simulink simulations

```
job = batchsim('mySim','Pool',3)
```



Speed up a parameter sweep using `parfor` on a cluster with MATLAB Parallel Server

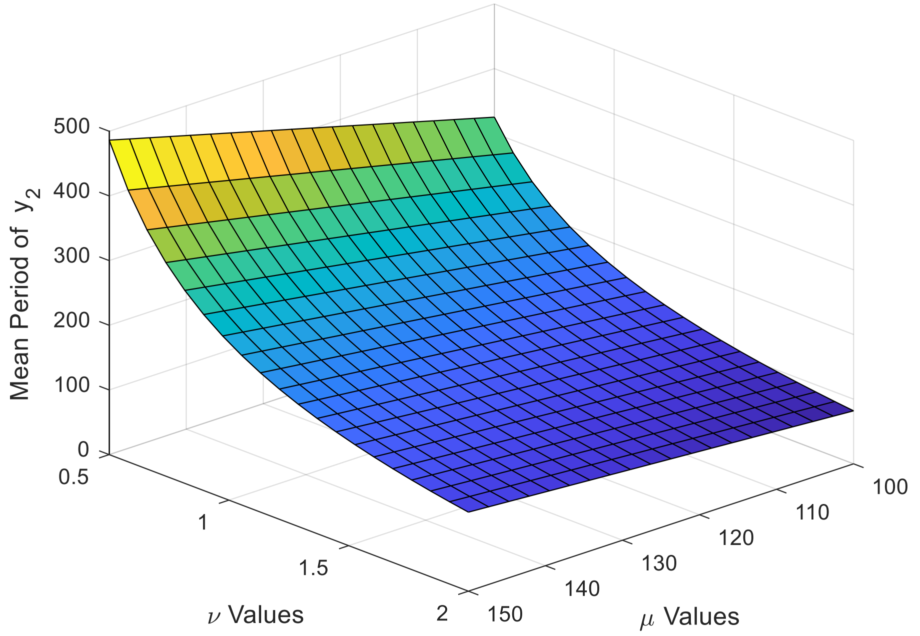
Demo: Parameter sweep for van der Pol oscillator

- System of ODEs

$$\dot{y}_1 = \nu y_2$$
$$\dot{y}_2 = \mu(1 - y_1^2)y_2 - y_1$$

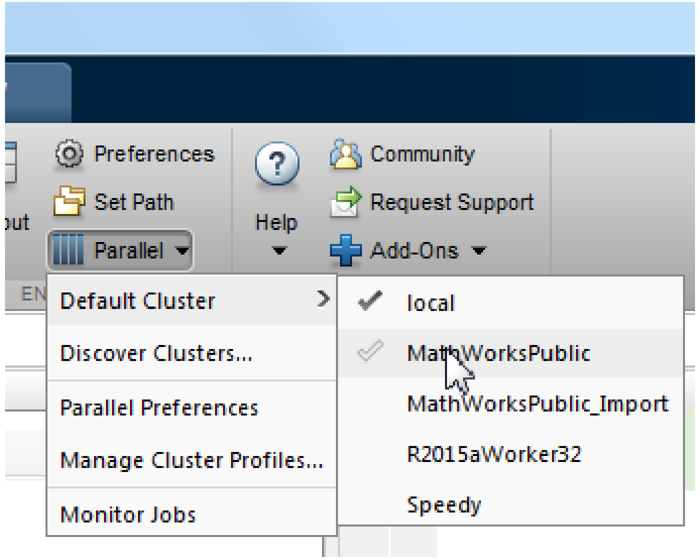
- Compute mean period of y

- Use `parfor,` study impact of ν, μ

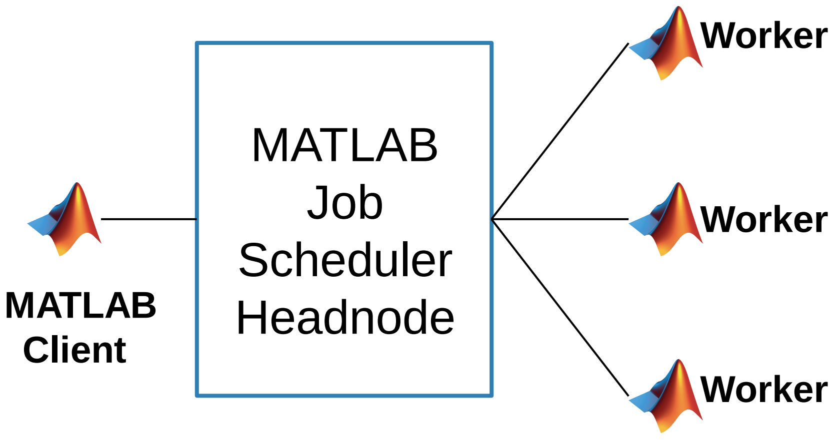


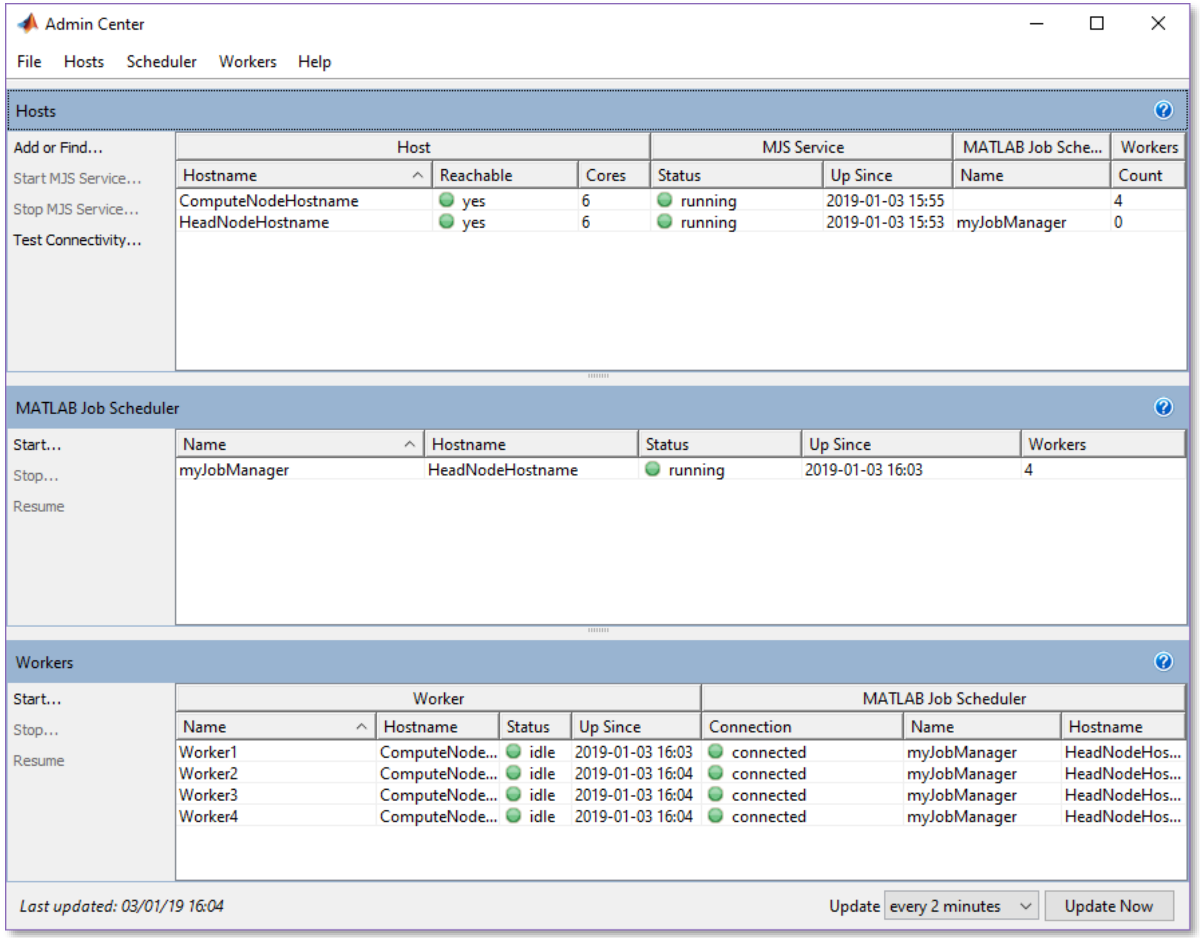
Migrate to Cluster / Cloud

- Use MATLAB Parallel Server
- Change hardware without changing algorithm



MATLAB Job Scheduler allows you to set up a MATLAB cluster from scratch





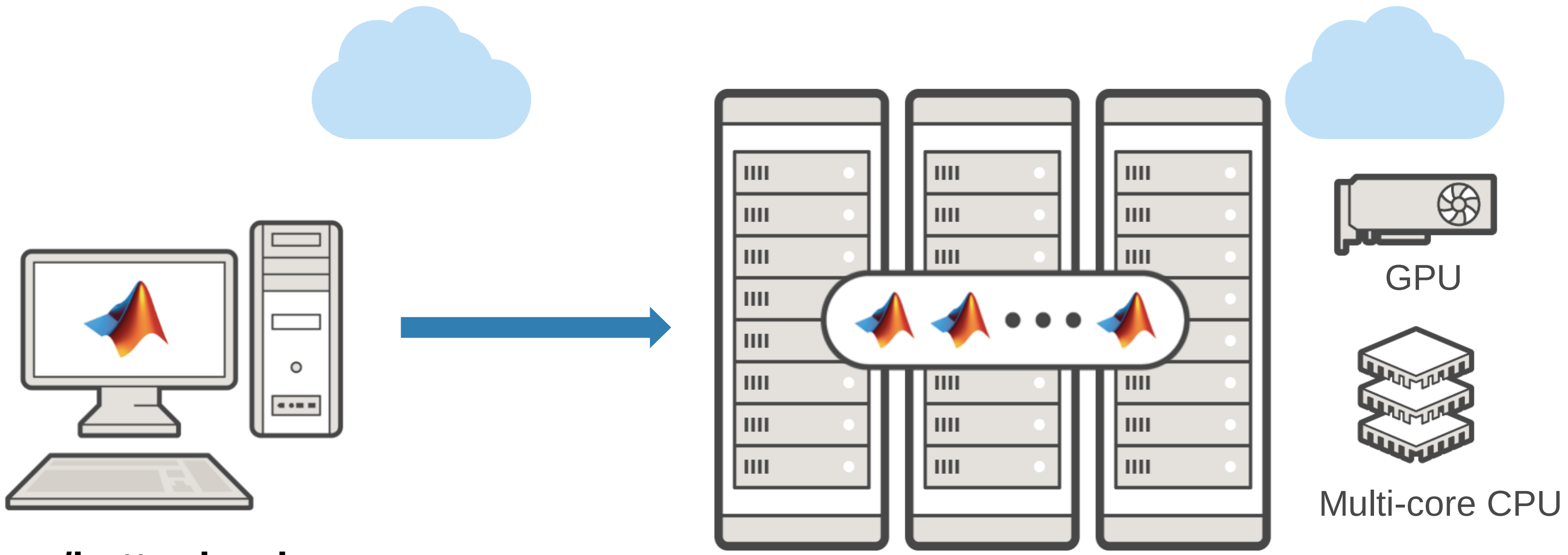
[Additional information]

Support for integration with third-party schedulers

Scheduler	Ready to use options	Customizable via generic scheduler API	Example Plugins available
Slurm	✓	✓	✓
Microsoft® Windows® HPC Server	✓		
Grid Engine family		✓	✓
IBM® Platform LSF	✓	✓	✓
PBS family	✓	✓	✓
HTCondor		✓	✓
Other schedulers		✓	

[Additional information]

Broad Range of Needs and Cloud Environments Supported



- **More /better hardware**
- **Proximity to cloud data**

Even more hardware to meet scaling needs

Access requirements	Desktop in the cloud	Cluster in the cloud (Client can be any cloud on on-premise desktop)
Any user could set up	NVIDIA GPU Cloud	MathWorks Cloud Center
Customizable template-based set up	MathWorks Cloud Reference Architecture	
Full set-up in custom environment	Custom installation - DIY	

Learn More: [Parallel Computing on the Cloud]

MathWorks Reference architectures enable advanced environments

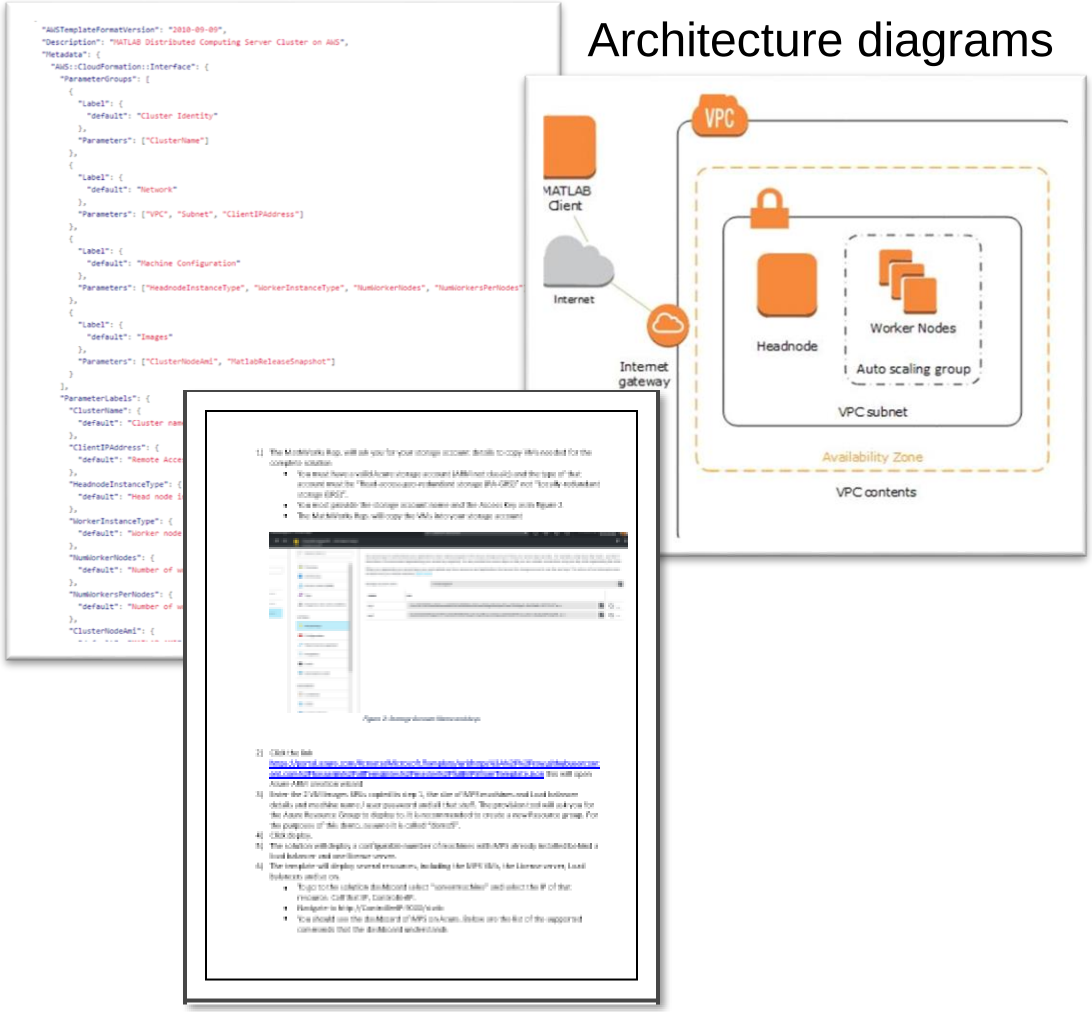
Amazon Web Services and Microsoft Azure

- Published on GitHub, by MathWorks
  - MATLAB
  - MATLAB Parallel Server
  - MATLAB Production Server

- Benefits:
  - Quick infrastructure setup in the cloud
  - MATLAB pre-installed
  - Incorporates best practices
  - You can adapt or extend for your specific needs

Cloud templates

Architecture diagrams

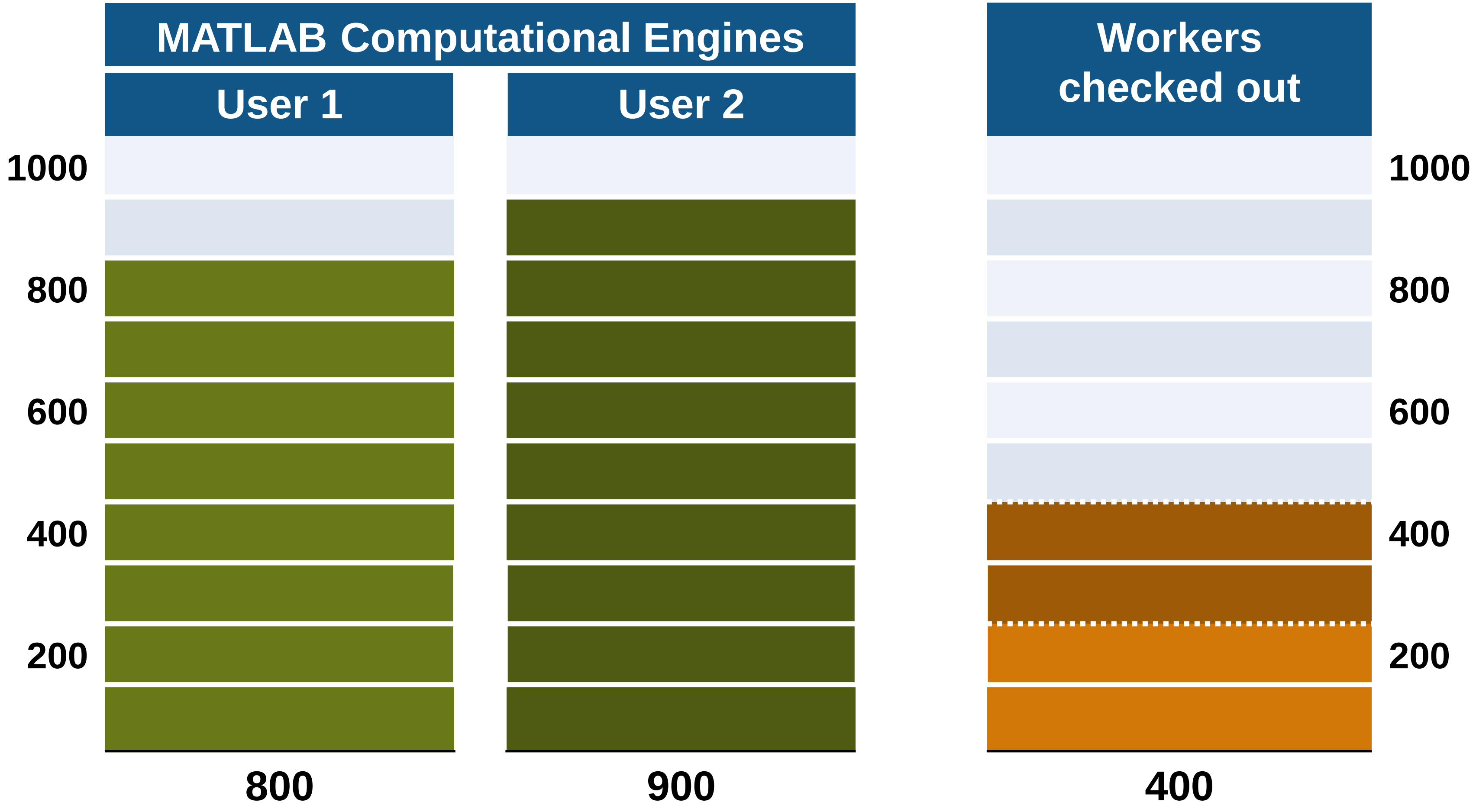


Step by step instructions

MATLAB Parallel Server – license update

Users who checkout 200 workers can scale without further check-outs

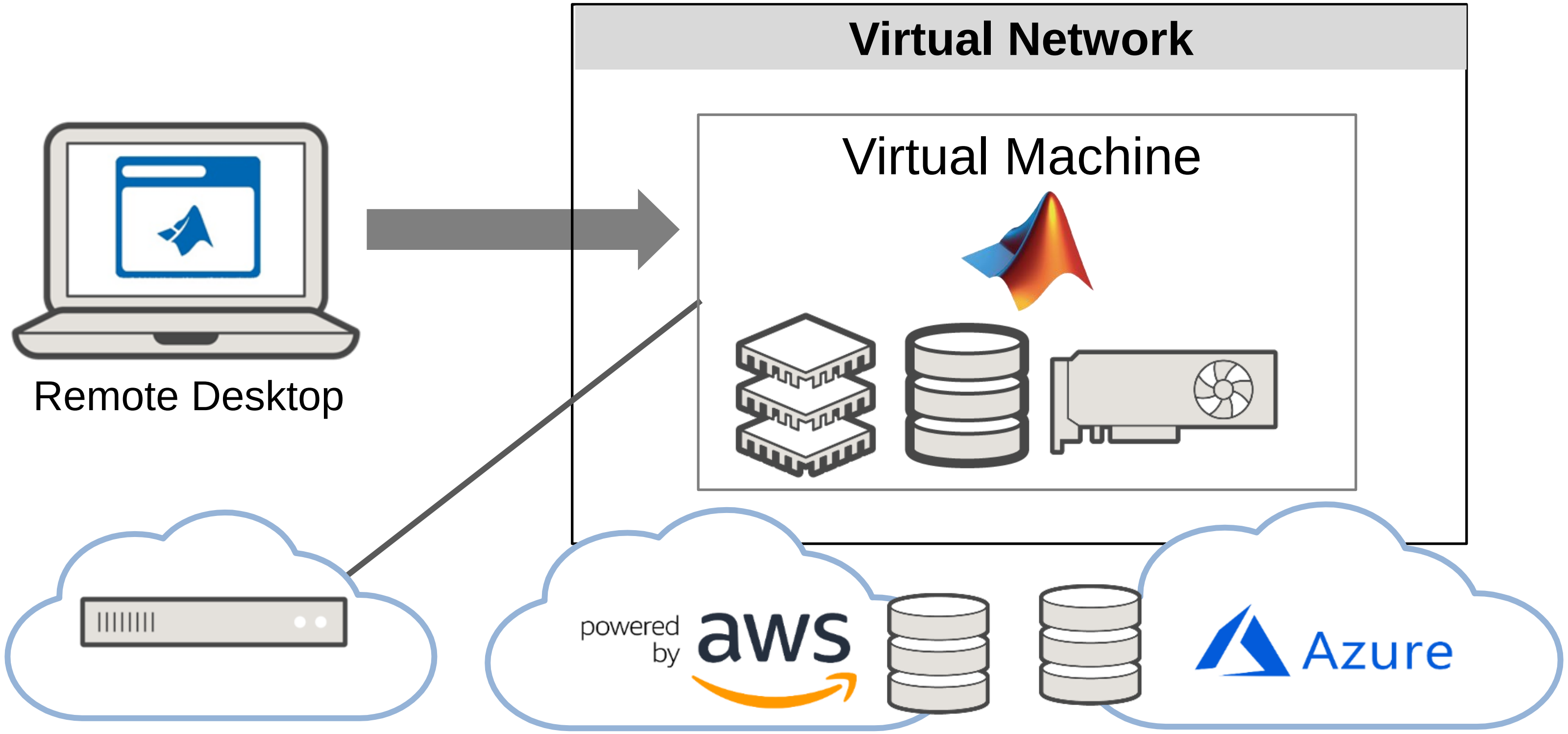
Backward compatible: Update license file *and* network license manager ***OR*** use online licensing



MATLAB Parallel Server for Campus-Wide License

- One license, no limits
  - Updated license allows unlimited scaling for all users
  - Network license manager (Flex)
    - Typically used with HPC centers, organizational clusters, or departmental clusters
    - One activation can simultaneously serve multiple clusters on the same network
  - Online licensing
    - Typically used with cloud clusters, personal clusters, or group clusters
    - Eliminates the need to set up a network license manager

Run MATLAB in the cloud



License Manager
(Hosted by your organization or by MathWorks)

- Run MATLAB in the cloud to access better hardware
- Perform data analytics on cloud-stored data
- Deploy MATLAB via Azure Marketplace or MathWorks Reference Architecture

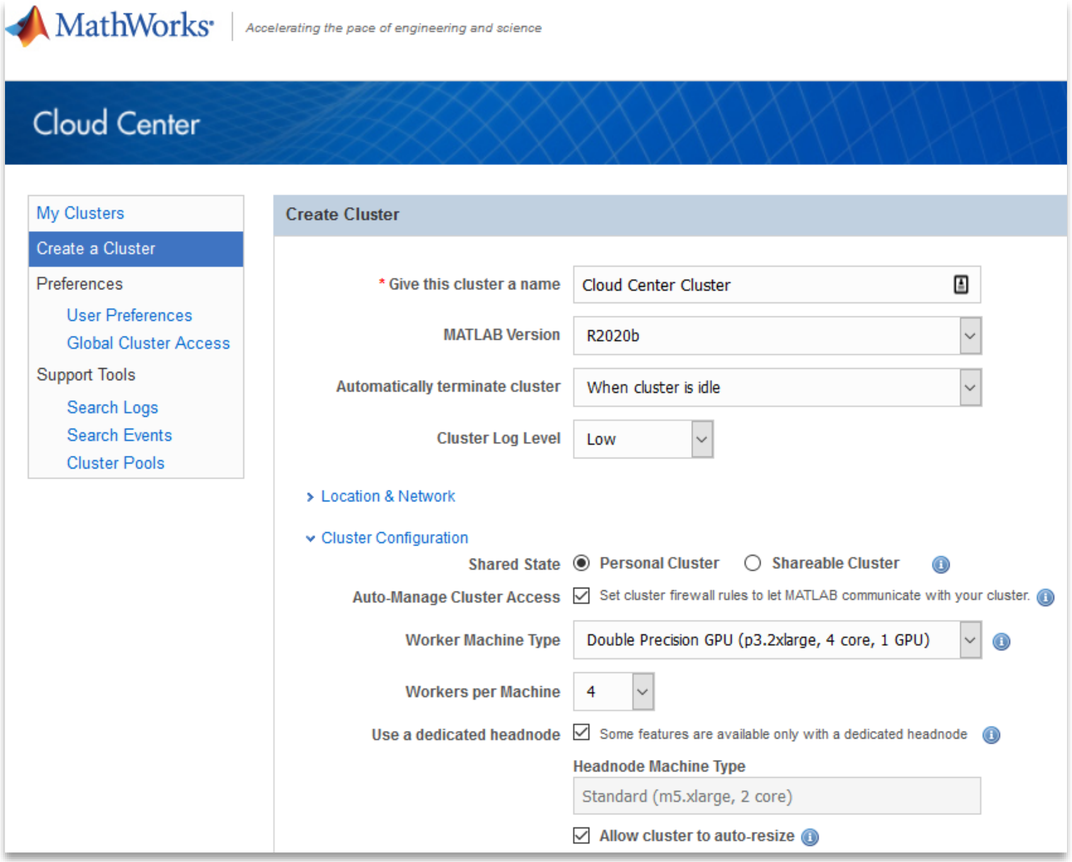
Create and manage AWS cloud clusters with Cloud Center

Use [Cloud Center] to create and access compute clusters in the Amazon cloud for parallel computing

You can access a cloud cluster from your client MATLAB session like any other cluster in your own onsite network

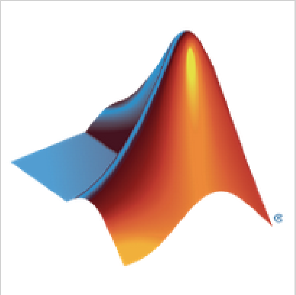
Requires:

1. MATLAB Parallel Server license configured to use [online licensing]
2. Amazon Web Services (AWS) account



Create and manage cloud clusters in the Azure Marketplace

Products > MATLAB Parallel Server (BYOL)



MATLAB Parallel Server (BYOL)

Save to my list

MathWorks

★★★★★ (0) Write a review

Overview Plans Reviews

GET IT NOW

Pricing information
Cost of deployed template components

Categories
Analytics
Compute

Support
Support
Help

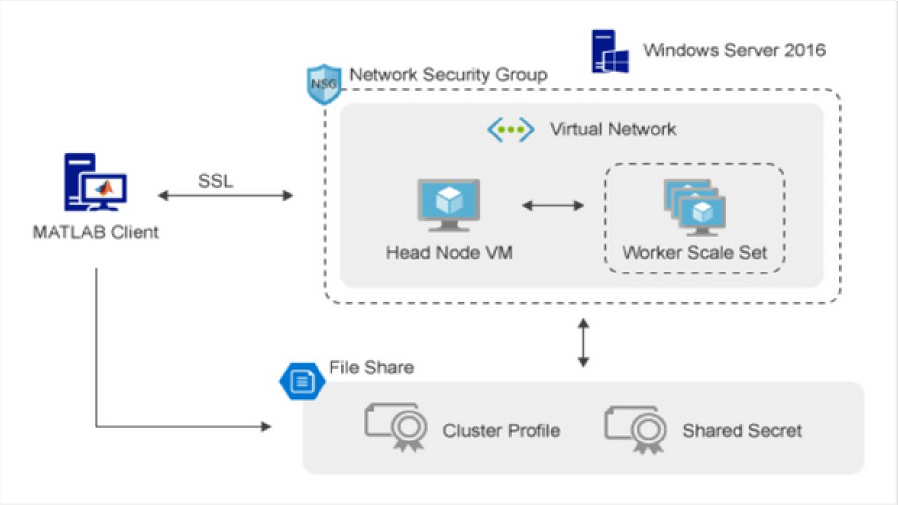
Legal
License Agreement
Privacy Policy

MATLAB Parallel Server™ lets you scale MATLAB® programs and Simulink® simulations.

Use the MATLAB Parallel Server (BYOL) for Azure Marketplace software plans to easily scale your MATLAB® programs and Simulink® simulations on Azure. The flexibility of Azure infrastructure combined with MATLAB Parallel Server enables you to scale your computation to the right size cluster at the right time.

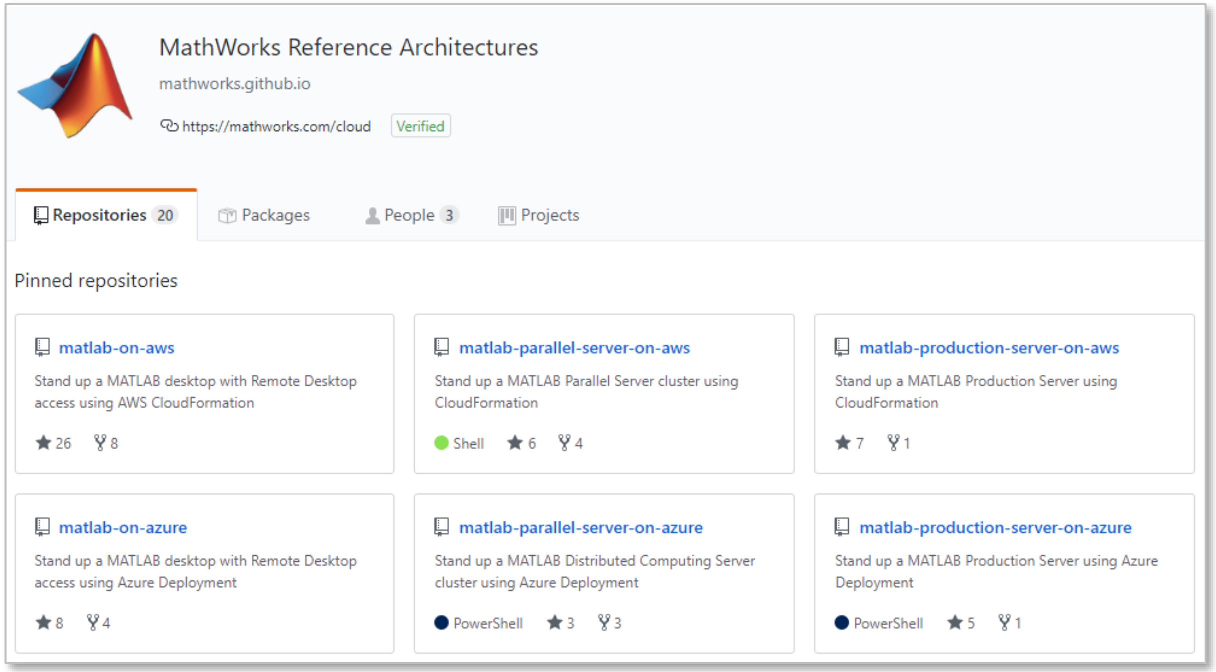
These software plans, developed by MathWorks for Azure, incorporates best practices to let you quickly create, configure, and deploy a cluster with MATLAB Parallel Server and MATLAB Job Scheduler, the MATLAB optimized scheduler provided with MATLAB Parallel Server. MathWorks provides ARM templates that use preconfigured virtual machines to create the required cluster infrastructure. This means you can configure the infrastructure and software quickly.

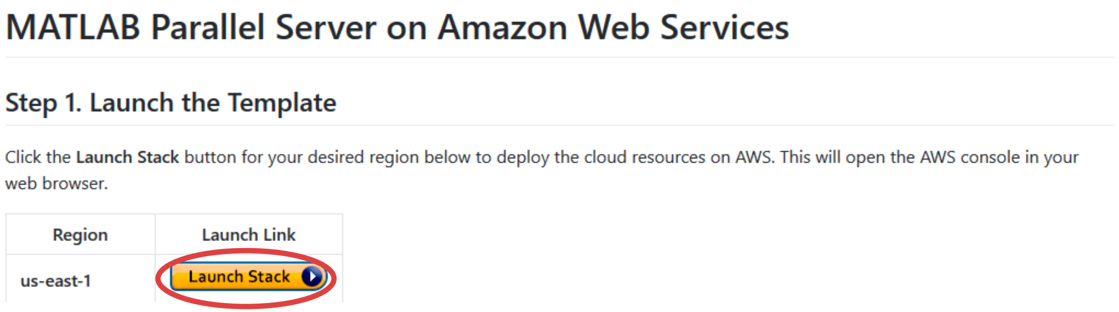
MATLAB Parallel Server™ lets you scale MATLAB® programs and Simulink® simulations to cloud clusters. You can prototype your programs and simulations on the MATLAB desktop and then run them on the cloud without recoding.



Create and manage cloud clusters with cloud reference architectures

https://github.com/mathworks-ref-arch





- What's included:
  - Virtual machine images
  - Cloud templates
  - Architecture diagrams
  - Step-by-step instructions

Resources for scaling: clouds and clusters

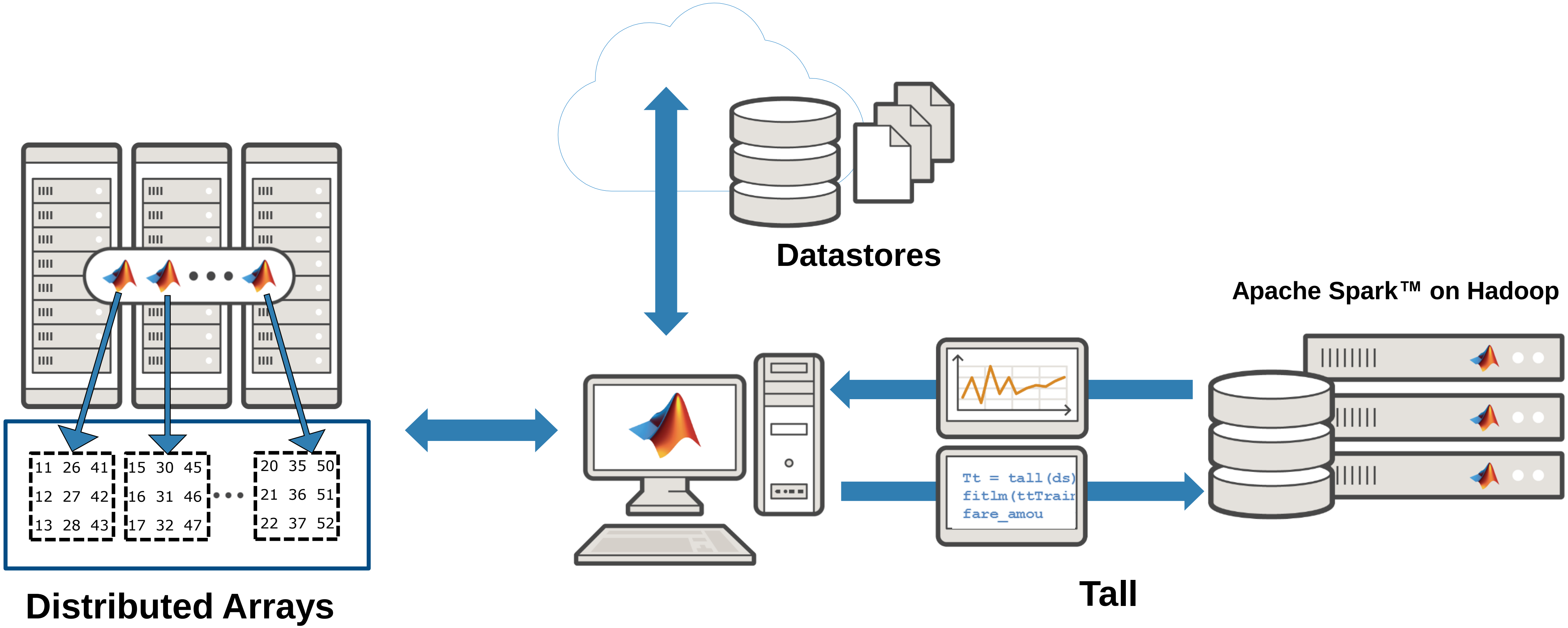
Environment	Motivation
Cloud Workstation	• AWS Reference Architecture (MATLAB) • Azure Reference Architecture (MATLAB) • Azure Marketplace – MATLAB (BYOL)
DGX, NGC (GPU Workstation)	• MATLAB Deep Learning container on NVIDIA GPU Cloud for NVIDIA DGX • Create a MATLAB Container Image
On-Premise and Cloud Clusters	• Integrate MATLAB with Third-Party Schedulers • Integrate MATLAB Job Scheduler for Online Licensing
Cloud Center	• MathWorks Cloud Center (AWS)
Marketplace	• Microsoft Azure Marketplace (MATLAB Parallel Server)
Cloud Reference Architecture	• AWS Reference Architecture (MATLAB Parallel Server) • Azure Reference Architecture (MATLAB Parallel Server)

Additional information

Agenda

- Utilizing multiple cores on a desktop computer
- Scaling up to cluster and cloud resources
- Tackling data-intensive problems on desktops and clusters
- Accelerating applications with NVIDIA GPUs
- Summary and resources

Extend Big Data Capabilities in MATLAB with Parallel Computing



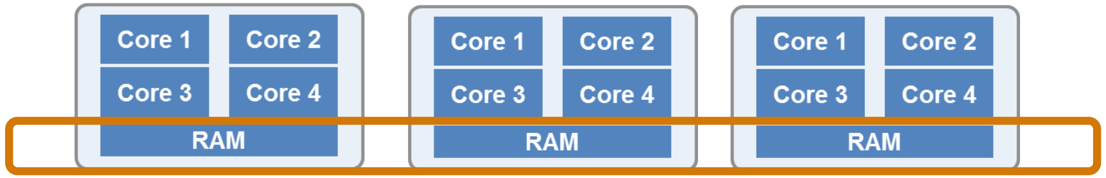
Overcoming Single Machine Memory Limitations

Distributed Arrays

- Distributed Data
  - Large matrices using the combined memory of a cluster

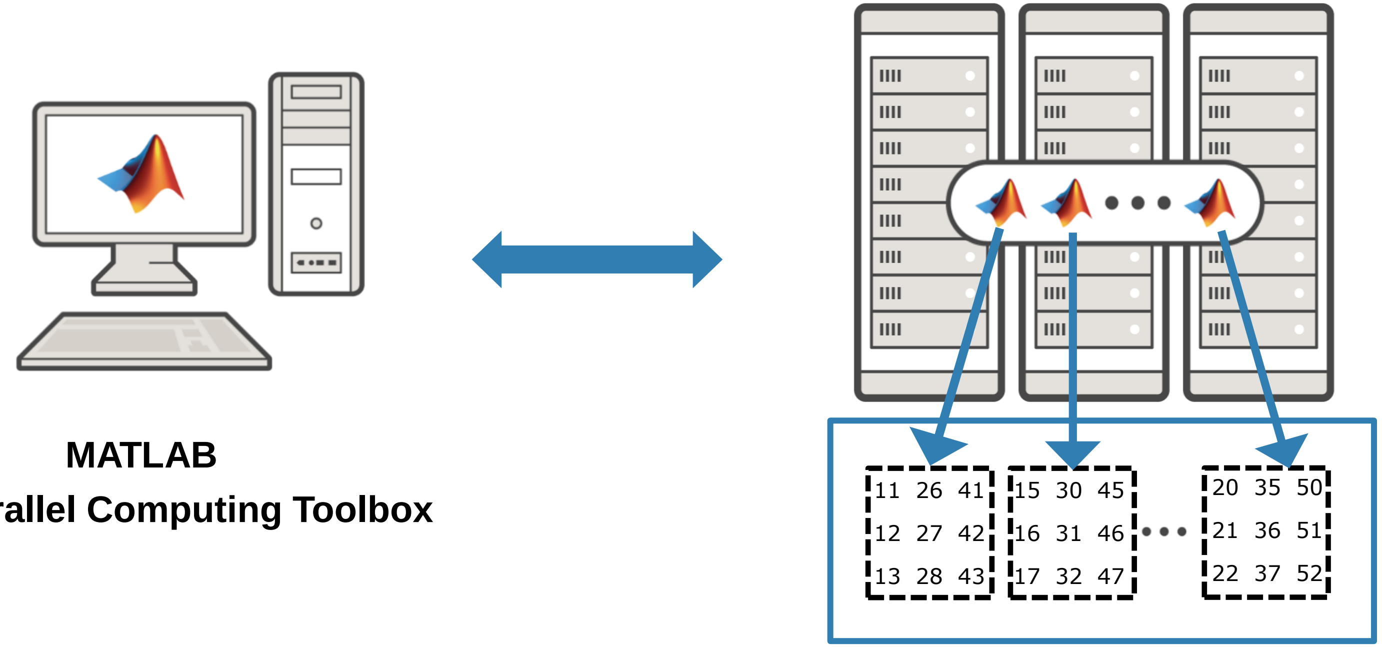
- Common Actions
  - Matrix Manipulation
  - Linear Algebra and Signal Processing

- A large number of standard MATLAB functions work with distributed arrays just as they do for normal variables



distributed arrays

- Distribute large matrices across workers running on a cluster
- Support includes matrix manipulation, linear algebra, and signal processing
- Several hundred MATLAB functions overloaded for distributed arrays



MATLAB
Parallel Computing Toolbox

MATLAB Parallel Server

distributed arrays

Develop and prototype locally and then scale to the cluster

MATLAB
Parallel Computing Toolbox

```
% prototype with a small data set
parpool('local');

% Read the data - read in part of the data
ds = datastore('colchunk_A_1.csv');

% Send data to workers
dds = distributed(ds);

% Run calculations
A = sparse(dds.i, dds.j, dds.v);
x = A \ distributed.ones(n^2, 1);

% Transfer results to local workspace
xg = gather(x);
```

MATLAB Parallel Server

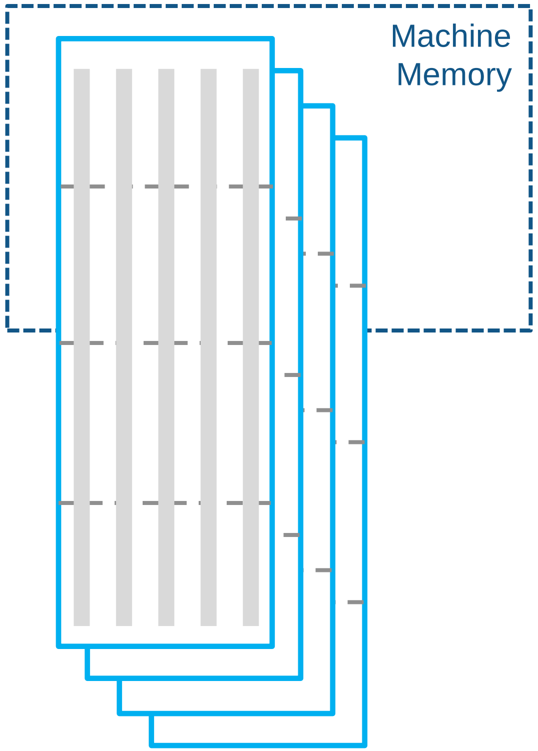
```
% prototype with a large data set
parpool('cluster');

% Read the data - read the whole dataset
ds = datastore('colchunk_A_*.csv');

% Send data to workers
dds = distributed(ds);

% Run calculations
A = sparse(dds.i, dds.j, dds.v);
x = A \ distributed.ones(n^2, 1);

% Transfer results to local workspace
xg = gather(x);
```

[Working with distributed arrays]

Tall Arrays

- Applicable when:
  - Data is **columnar** – with **many** rows
  - Overall data size is **too big to fit into memory**
  - Operations are mathematical/statistical in nature

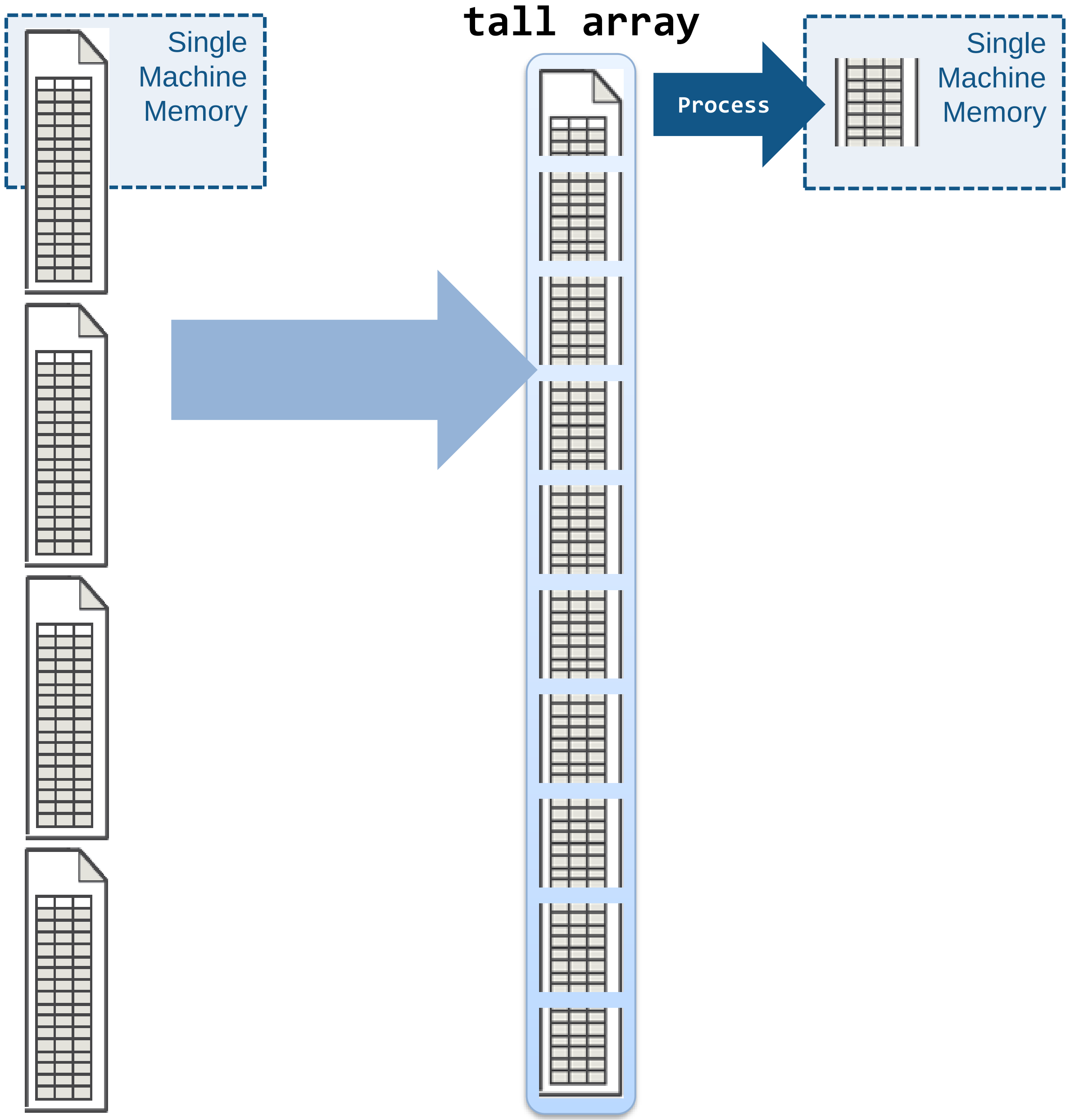
- Statistical and machine learning applications
  - Hundreds of functions supported in MATLAB and Statistics and Machine Learning Toolbox



Tall Data

Tall Arrays

- Automatically breaks data up into small “chunks” that fit in memory
- Tall arrays scan through the dataset one “chunk” at a time
- Processing code for tall arrays is the same as ordinary arrays



Demo: Predicting Cost of Taxi Ride in NYC

Working with tall arrays in MATLAB

- **Objective:** Create a model to predict the cost of a taxi ride in New York City

- **Inputs:**
  - Monthly taxi ride log files
  - The local data set contains > 2 million rows

- **Approach:**
  - Preprocess and explore data
  - Work with subset of data for prototyping
  - Fit linear model
  - Predict fare and validate model

Example: Prototyping

Preview Data

Description

- Location: New York City
- Date(s): (Partial) January 2015
- Data size: **“small data” 13,693 rows / ~2 MB**

```
>> ds = datastore('taxidataNYC_1_2015.csv');
>> preview(ds)

    VendorID    tpep_pickup_datetime    tpep_dropoff_datetime    passenger_count    trip_distance    pickup_longitude    pic
    ________    ____________________    _____________________    _______________    _____________    ________________    ___

    2           2015-01-10 02:24:04     2015-01-10 02:36:10      2                  2.19             -73.999             40.
    1           2015-01-18 21:29:35     2015-01-18 21:34:15      1                     1             -74.017             40.
    2           2015-01-23 18:23:02     2015-01-23 18:39:32      3                  2.22             -73.973             40.
    1           2015-01-01 05:29:50     2015-01-01 05:48:55      1                   3.6             -73.943             40.8
    1           2015-01-18 00:06:42     2015-01-18 00:11:43      1                   0.8             -73.983             40.
    2           2015-01-29 23:56:41     2015-01-30 00:02:49      5                  0.87             -73.982             40.
    2           2015-01-05 16:58:24     2015-01-05 17:03:33      5                  0.78             -73.992             40.
    1           2015-01-23 23:49:53     2015-01-23 23:55:42      2                   1.4             -73.956              40.
```

Example: Prototyping

Create a Tall Array

Number of rows is unknown until all the data has been read

Input data is tabular – result is a tall table

Only the first few rows are displayed

```
>> tt = tall(ds)
tt =

  M×19 tall table

    VendorID    tpep_pickup_datetime    tpep_dropoff_datetime    passenger_count    trip_distance    pickup_longitude    pic
    ________    ____________________    _____________________    _______________    _____________    ________________    ___

    2           2015-01-10 02:24:04     2015-01-10 02:36:10      2                  2.19             -73.999             40.
    1           2015-01-18 21:29:35     2015-01-18 21:34:15      1                     1             -74.017             40.
    2           2015-01-23 18:23:02     2015-01-23 18:39:32      3                  2.22             -73.973             40.
    1           2015-01-01 05:29:50     2015-01-01 05:48:55      1                   3.6             -73.943             40.
    1           2015-01-18 00:06:42     2015-01-18 00:11:43      1                   0.8             -73.983             40.
    2           2015-01-29 23           30 00:02:49              5                  0.87             -73.982             40.
    2           2015-01-05 16           05 17:03:33              5                  0.78             -73.992             40.
    1           2015-01-23 23           23 23:55:42              2                   1.4             -73.956             40.
    :           :                       :                        :                  :                :                   :
    :           :                       :                        :                  :                :                   :
```

Example: Prototyping

Calling Functions with a Tall Array

Once the tall table is created, can process much like an ordinary table

```
% Calculate average trip duration
mnTrip = mean(tt.trip_minutes,'omitnan')

mnTrip =

   tall double

    ?

Preview deferred. Learn more.

% Execute commands and gather results into workspace
mn = gather(mnTrip)

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 1: Completed in 4 sec
Evaluation completed in 4 sec

mn =

   13.2763
```

- Most results are evaluated only when explicitly requested (e.g., **`gather`**)
- MATLAB automatically optimizes queued calculations to minimize the number of passes through the data

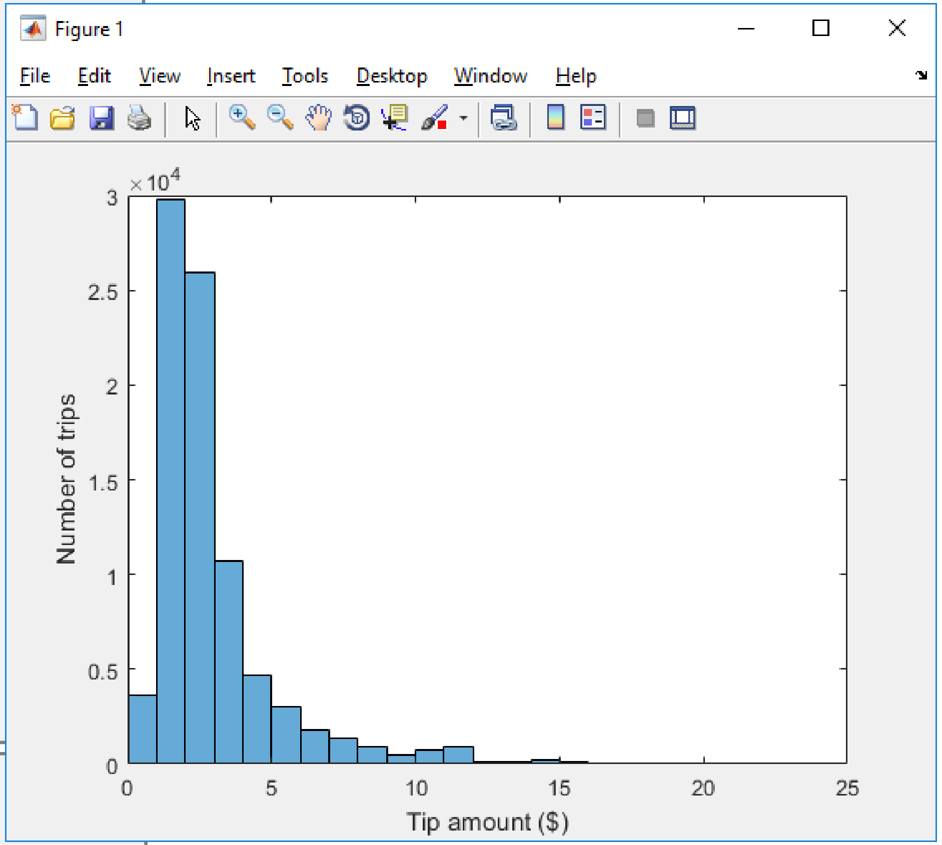
Example: Prototyping

Calling Functions with a Tall Array

```
% Remove some bad data
tt.trip_minutes = minutes(tt.tpep_dropoff_datetime -
tt.tpep_pickup_datetime);
tt.speed_mph = tt.trip_distance ./ (tt.trip_minutes ./ 60);
ignore = tt.trip_minutes <= 1 | ...     % really short time
    tt.trip_minutes >= 60 * 12 | ...    % unfeasibly long time
    tt.trip_distance <= 1 | ...         % really short distance
    tt.trip_distance >= 12 * 55 | ...   % unfeasibly far
    tt.speed_mph > 55 | ...             % unfeasibly fast
    tt.fare_amount < 0 | ...            % negative fares?!
    tt.fare_amount > 10000;             % unfeasibly large fares
tt(ignore, :) = [];

% Credit card payments have the most accurate tip data
keep = tt.payment_type == {'Credit card'};
tt = tt(keep,:);

% Show tip distributi
histogram( tt.tip_amou
```

Data only read once, despite 21 operations

```
Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 1: Completed in 5 sec
Evaluation completed in 5 sec
```



Example: Prototyping

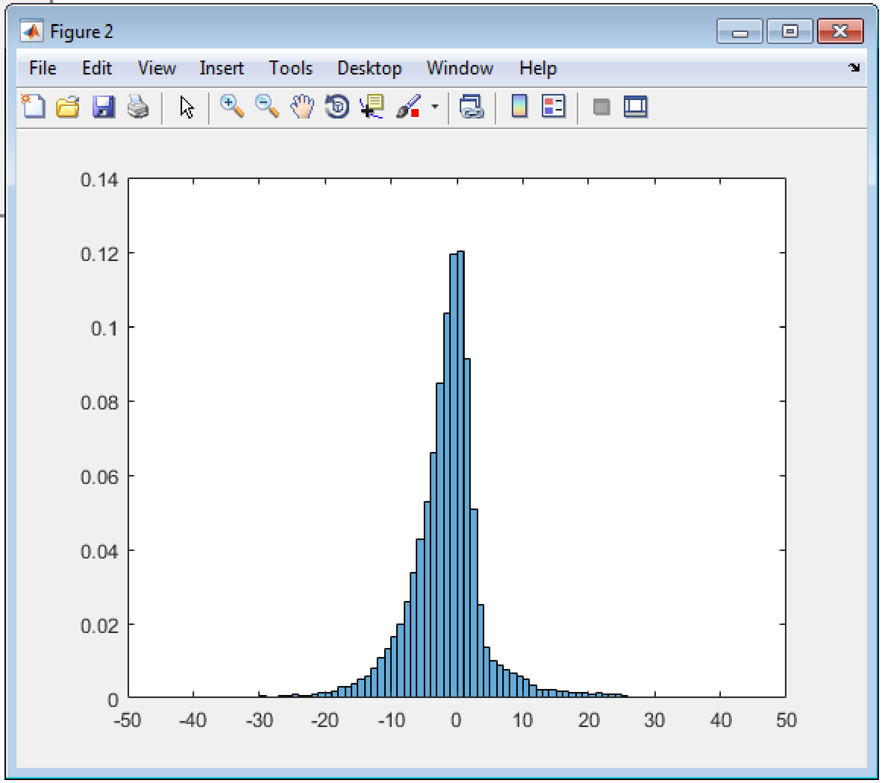
Fit predictive model

```
% Fit predictive model
model = fitlm(ttTrain,'fare_amount ~ 1 + hr_of_day + trip_distance*trip_minutes')

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 1: Completed in 5 sec
Evaluation completed in 5 sec

model =

Compact linear regression model:
    fare_amount ~ 1 + hr_of_day + trip_distance*trip_minutes

Estimated Coefficients:
                                   Estimate          SE          tStat       pValue 
                                  __________    __________    ________    _______

    (Intercept)                       2.3432      0.040181      58.318          0
    trip_distance                     2.5841     0.0063898      404.41          0
    hr_of_day                     -0.0012969     0.0018789    -0.69024    0.49005
    trip_minutes                     0.22098     0.0020412      108.26          0
    trip_distance:trip_minutes     -0.007857    0.00017539     -44.798          0

Number of observations: 42373, Error degrees of freedom: 42368
Root Mean Squared Error: 2.58
R-squared: 0.938,  Adjusted R-Squared 0.938
F-statistic vs. constant model: 1.59e+05, p-value = 0
```

Example: Prototyping

Predict and validate model

```
% Predict and validate
yPred = predict(model,ttValidation);
residuals = yPred - ttValidation.fare_amount;
figure
histogram(residuals,'Normalization','pdf','BinLimits',[-50 50])

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 2: Completed in 5 sec
- Pass 2 of 2: Completed in 4 sec
Evaluation completed in 10 sec
```



Scale to the Entire Data Set

Description

- Location: New York City
- Date(s): All of 2015
- Data size: **“Big Data” 150,000,000 rows / ~25 GB**

Example: “small data” processing vs. Big Data processing

“small data” processing

```
% Access the data
ds = datastore('taxidataNYC_1_2015.csv');
tt = tall(ds);
```

```
% Calculate average trip duration
mnTrip = mean(tt.trip_minutes,'omitnan')

% Execute commands and gather results into workspace
mn = gather(mnTrip)

% Remove some bad data
tt.trip_minutes = minutes(tt.tpep_dropoff_datetime -
tt.tpep_pickup_datetime);
tt.speed_mph = tt.trip_distance ./ (tt.trip_minutes ./ 60);
ignore = tt.trip_minutes <= 1 | ...    % really short time
    tt.trip_minutes >= 60 * 12 | ...   % unfeasibly long time
    tt.trip_distance <= 1 | ...        % really short distance
    tt.trip_distance >= 12 * 55 | ...  % unfeasibly far
    tt.speed_mph > 55 | ...            % unfeasibly fast
    tt.fare_amount < 0 | ...           % negative fares?!
    tt.fare_amount > 10000;            % unfeasibly large fares
tt(ignore, :) = [];
```

Big Data processing

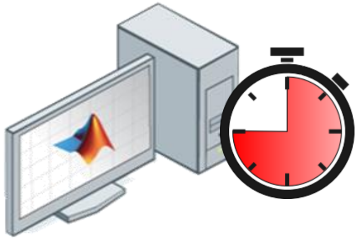
```
% Access the data
ds = datastore('taxiData\*.csv');
tt = tall(ds);
```

```
% Calculate average trip duration
mnTrip = mean(tt.trip_minutes,'omitnan')

% Execute commands and gather results into workspace
mn = gather(mnTrip)

% Remove some bad data
tt.trip_minutes = minutes(tt.tpep_dropoff_datetime -
tt.tpep_pickup_datetime);
tt.speed_mph = tt.trip_distance ./ (tt.trip_minutes ./ 60);
ignore = tt.trip_minutes <= 1 | ...    % really short time
    tt.trip_minutes >= 60 * 12 | ...   % unfeasibly long time
    tt.trip_distance <= 1 | ...        % really short distance
    tt.trip_distance >= 12 * 55 | ...  % unfeasibly far
    tt.speed_mph > 55 | ...            % unfeasibly fast
    tt.fare_amount < 0 | ...           % negative fares?!
    tt.fare_amount > 10000;            % unfeasibly large fares
tt(ignore, :) = [];
```

Scaling up

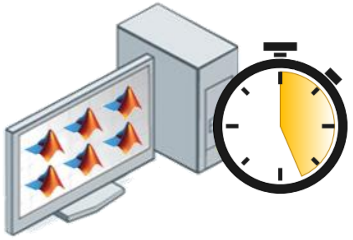
If you just have **MATLAB**:

- Run through each 'chunk' of data one by one



If you also have **Parallel Computing Toolbox**:

- Use all local cores to process several 'chunks' at once



If you also have a cluster with **MATLAB Parallel Server**:

- Use the whole cluster to process many 'chunks' at once

Scaling up

Working with clusters from MATLAB desktop:

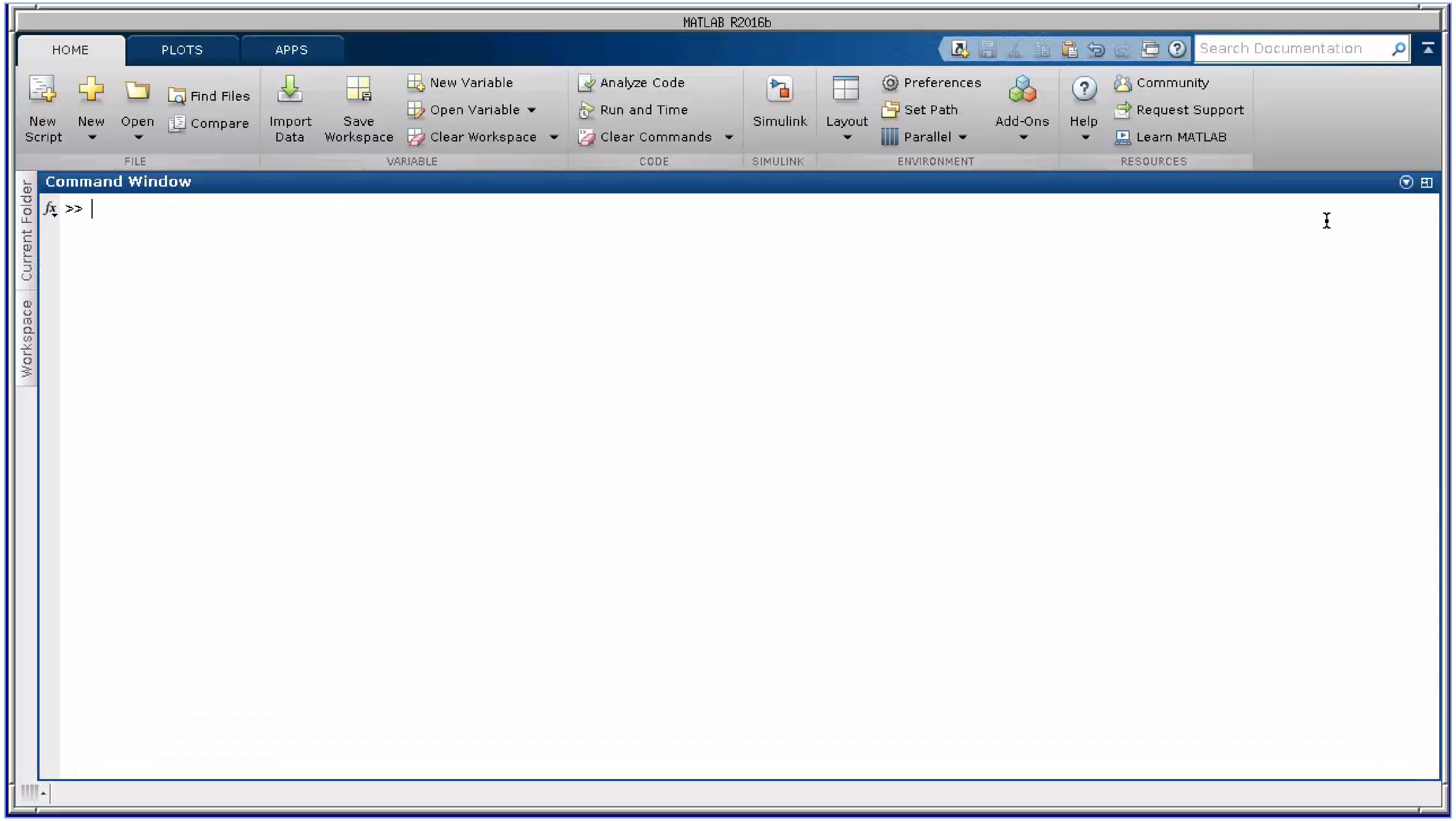
- General purpose MATLAB cluster
  - Can co-exist with other MATLAB workloads (`parfor`, `parfeval`, `spmd`, jobs and tasks, `distributed` arrays, …)
  - Uses local memory and file caches on workers for efficiency

- Spark-enabled Hadoop clusters
  - Data in HDFS
  - Calculation is scheduled to be near data
  - Uses Spark’s built-in memory and disk caching

Example: Running on Spark + Hadoop

```
% Hadoop/Spark Cluster
numWorkers = 16;

setenv('HADOOP_HOME', '/dev_env/cluster/hadoop');
setenv('SPARK_HOME', '/dev_env/cluster/spark');

cluster = parallel.cluster.Hadoop;
cluster.SparkProperties('spark.executor.instances') = num2str(numWorkers);
mr = mapreducer(cluster);


% Access the data
ds = datastore('hdfs://hadoop01:54310/datasets/taxiData/*.csv');
tt = tall(ds);
```

Example: Running on Spark + Hadoop



`tall` arrays vs. `distributed` arrays

- `tall` arrays are useful for out-of-memory datasets with a “tall” shape
  – Can be used on a desktop, cluster, or with Spark/Hadoop
  – Low-level alternatives are `mapreduce` and MATLAB API for Spark
- `distributed` arrays are useful for in-memory datasets on a cluster
  – Can be any shape (“tall”, “wide”, or both)
  – Create custom functions with `spmd` + `gop`

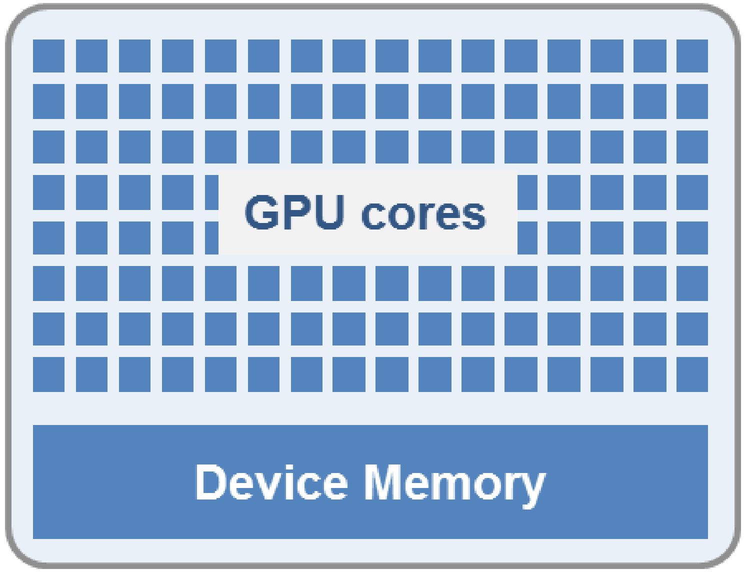
	Tall Array	Distributed Array
Support Focus	Data Analytics, Statistics and Machine Learning	Linear Algebra, Matrix Manipulations
Data Shape – Tall	✓	✓
Data Shape – Wide		✓
Prototype on Desktop	✓	✓
Helps on Desktop	✓	
Run on HPC	✓	✓
Run on Spark/Hadoop	✓	
Fault Tolerant	✓	

Agenda

- Utilizing multiple cores on a desktop computer
- Scaling up to cluster and cloud resources
- Tackling data-intensive problems on desktops and clusters
- Accelerating applications with NVIDIA GPUs
- Summary and resources

Graphics Processing Units (GPUs)

- For graphics acceleration and scientific computing
- Many parallel processors
- Dedicated high speed memory



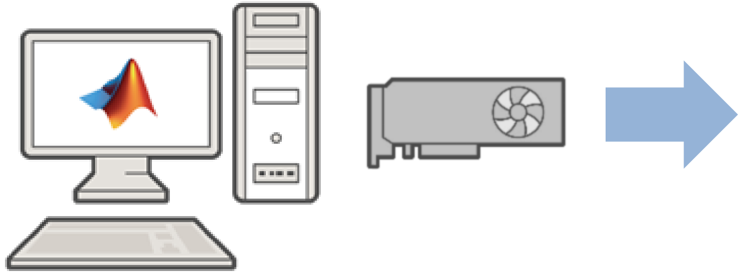
GPU Requirements

- Parallel Computing Toolbox requires NVIDIA GPUs
- www.nvidia.com/object/cuda_gpus.html
- GPU support by release

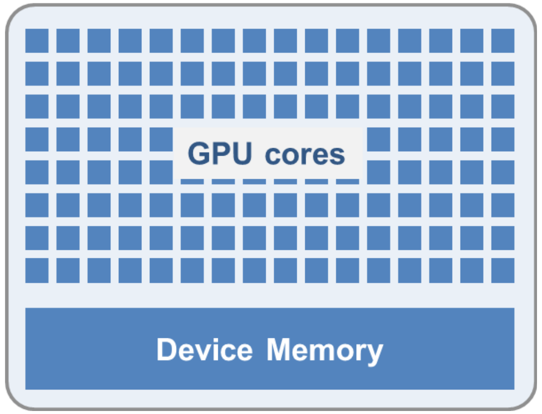
MATLAB Release	Required Compute Capability
MATLAB R2021a	3.5 or greater
MATLAB R2018a and later releases	3.0 or greater
MATLAB R2014b – MATLAB R2017b	2.0 or greater
MATLAB R2014a and earlier releases	1.3 or greater

GPU Computing Paradigm

NVIDIA CUDA-enabled GPUs



Parallel Computing Toolbox



NASA Langley Accelerates Acoustic Data Analysis with GPU Computing

Challenge

Accelerate the analysis of sound recordings from wind tunnel tests of aircraft components

Solution

- Use Parallel Computing Toolbox to process acoustic data
- Cut processing time by running computationally intensive operations on a GPU

Results

- GPU computations completed 40 times faster
- Algorithm GPU-enabled in 30 minutes
- Processing of test data accelerated

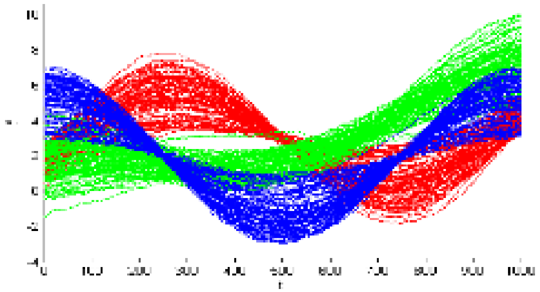
Wind tunnel test setup featuring the Hybrid Wing Body model (inverted), with 97-microphone phased array (top) and microphone tower (left).

"Our legacy code took up to 40 minutes to analyze a single wind tunnel test. The addition of GPU computing with Parallel Computing Toolbox cut it to under a minute. It took 30 minutes to get our MATLAB algorithm working on the GPU—no low-level CUDA programming was needed."

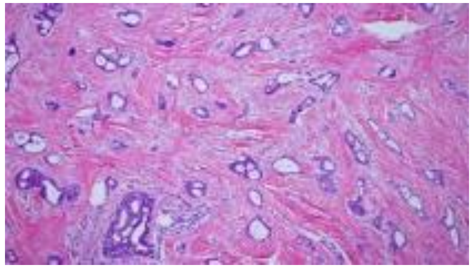
- Christopher Bahr, research aerospace engineer at NASA

Link to user story

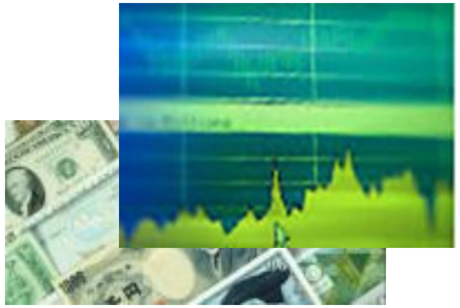
Speeding up MATLAB applications with GPUs



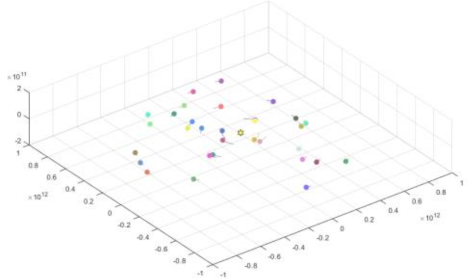
10x speedup
K-means clustering algorithm



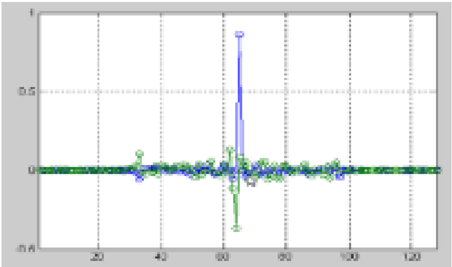
14x speedup
template matching routine



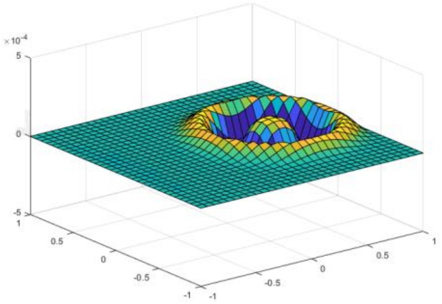
12x speedup
using Black-Scholes model



44x speedup
simulating the movement of celestial objects



4x speedup
adaptive filtering routine



77x speedup
wave equation solving

NVIDIA Titan V GPU, Intel® Core™ i7-8700T Processor (12MB Cache, 2.40GHz)

Programming with GPUs

Ease of Use

Parallel-enabled toolboxes

Common programming constructs
(`gpuArray, gather`)

Advanced programming constructs

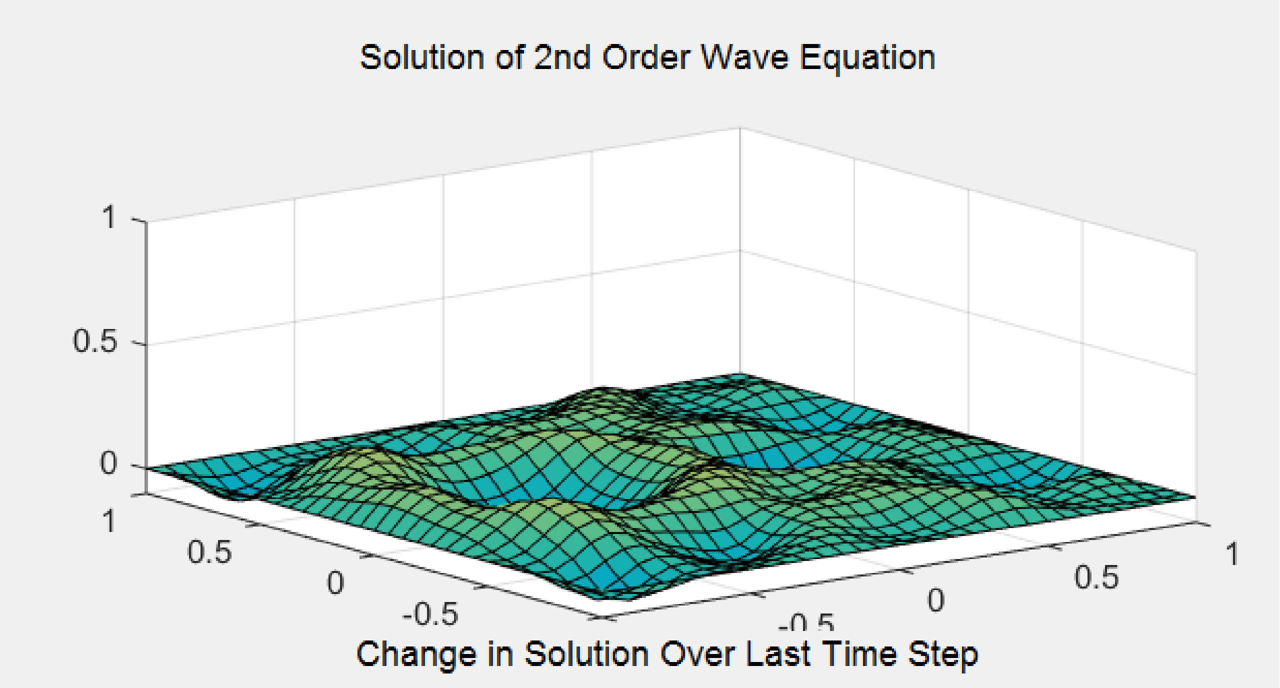
Greater Control

Demo: Wave Equation

Accelerating scientific computing in MATLAB with GPUs

- Objective: Solve 2nd order wave equation with spectral methods

- Approach:
  - Develop code for CPU
  - Modify the code to use GPU computing using gpuArray
  - Compare performance of the code using CPU and GPU



Speed-up using NVIDIA GPUs

- Ideal Problems
  - Massively Parallel and/or Vectorized operations
  - Computationally Intensive

- 500+ GPU-enabled MATLAB functions

- Simple programming constructs
  - `gpuArray, gather`

```
A1 = rand(3000,3000);
```

Transfer data to GPU from computer memory

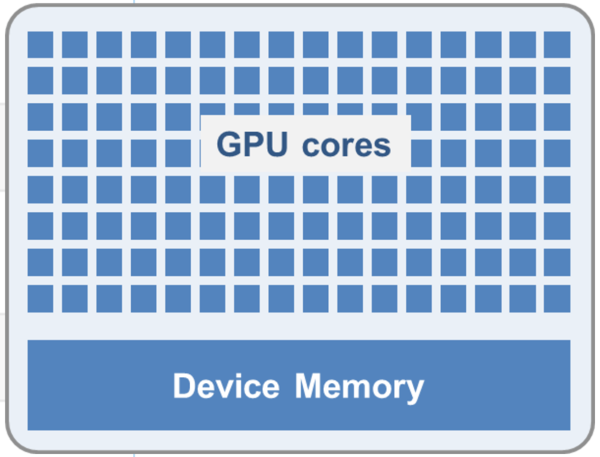
```
A2 = gpuArray(A1);
```

Perform calculation on GPU

```
B2 = fft(A2);
```

Gather data or plot

```
B2=gather(B2);
```



Programming with GPUs

Ease of Use

Parallel-enabled toolboxes

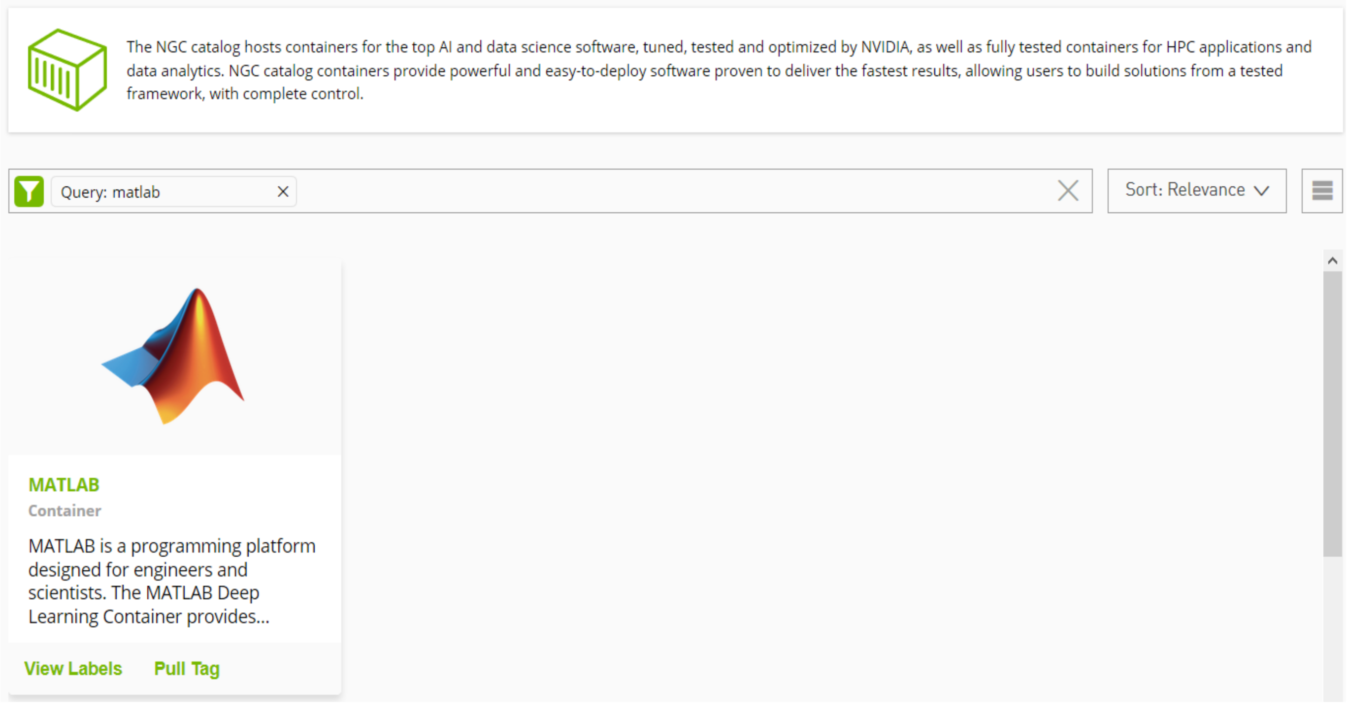
Common programming constructs

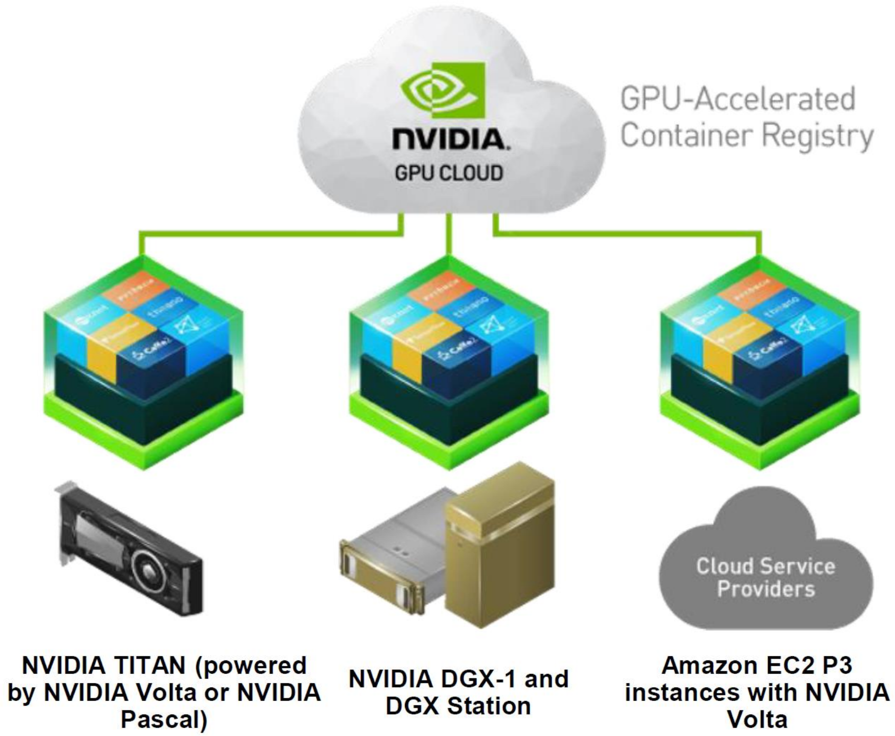
Advanced programming constructs
(`spmd`, `arrayfun`, `CUDAKernel`, `mex`)

Greater Control

Deep learning with MATLAB and NVIDIA GPU cloud containers

Use [MATLAB NGC containers]() in the cloud, on DGX, or on your machine





Agenda

- Utilizing multiple cores on a desktop computer
- Scaling up to cluster and cloud resources
- Tackling data-intensive problems on desktops and clusters
- Accelerating applications with NVIDIA GPUs
- Summary and resources

Feedback

http://bitly.com/matlab-lca-21

Summary

- Easily develop parallel MATLAB applications without being a parallel programming expert
- Run many Simulink simulations at the same time on multiple CPU cores.
- Speed up the execution of your applications using additional hardware including GPUs, clusters and clouds
- Develop parallel applications on your desktop and easily scale to a cluster when needed

Some Other Valuable Resources

- MATLAB Documentation
  – MATLAB → Advanced Software Development → Performance and Memory
  – Parallel Computing Toolbox

- Parallel and GPU Computing Tutorials
  – https://www.mathworks.com/videos/series/parallel-and-gpu-computing-tutorials-97719.html

- Parallel Computing on the Cloud with MATLAB
  – http://www.mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/

MathWorks