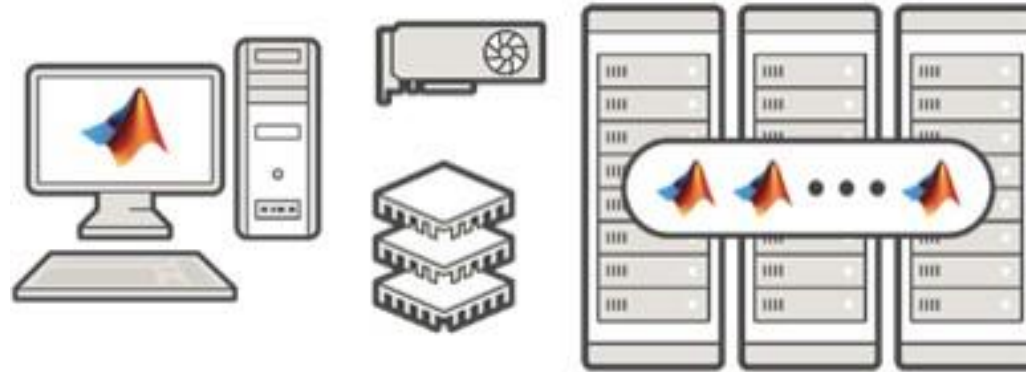


High Performance Computing with MATLAB

Yueyi Xu, Ph.D.
MathWorks China



Outline

- Before we go parallel: Code optimization for better performance
- How to write and run parallel MATLAB code on your local machine or a cluster
- How to run MATLAB code on GPUs to accelerate your computations
- Use big data processing methods to handle large datasets
- Learning Resources

Outline

- Before we go parallel: Code optimization for better performance
- How to write and run parallel MATLAB code on your local machine or a cluster
- How to run MATLAB code on GPUs to accelerate your computations
- Use big data processing methods to handle large datasets
- Learning Resources

! Before going parallel, make sure you optimize your serial code for best performance

- Use the **Code Analyzer** to automatically check your code for coding (and performance) problems.

```
1 tic
2 x = 0;
3 for k = 2:1e6
4     x(k) = x(k-1) + 1;
5 end
6 toc
```

! Line 4: Variable appears to change size on every loop iteration (within a script). Consider preallocating for speed.

Details ▼

Elapsed time is 0.075824 seconds.

```
1 tic
2 x = zeros(1,1e6);
3 for k = 2:1e6
4     x(k) = x(k-1) + 1;
5 end
6 toc
```

! Preallocating the maximum amount of space required for the array instead of letting MATLAB repeatedly reallocate memory for the growing array is more than 5 times faster!

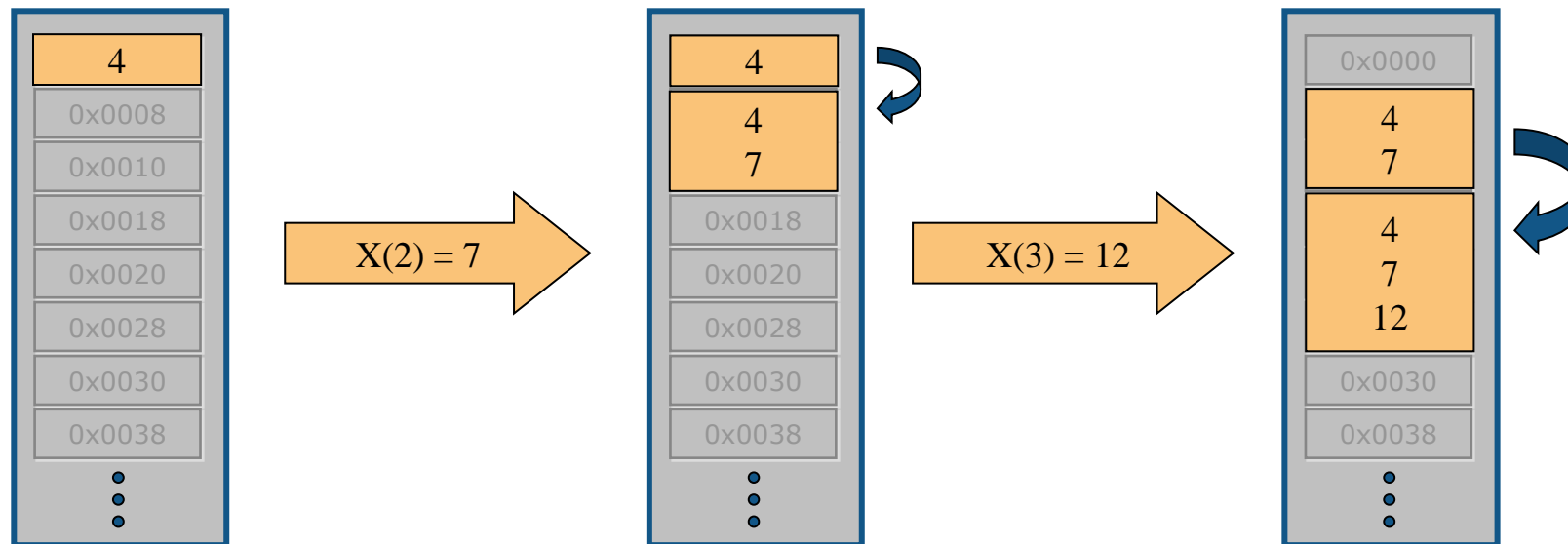
Elapsed time is 0.013109 seconds.

Effect of Not Preallocating Memory

$$x(1) = 4$$

$$x(2) = 7$$

$$x(3) = 12$$



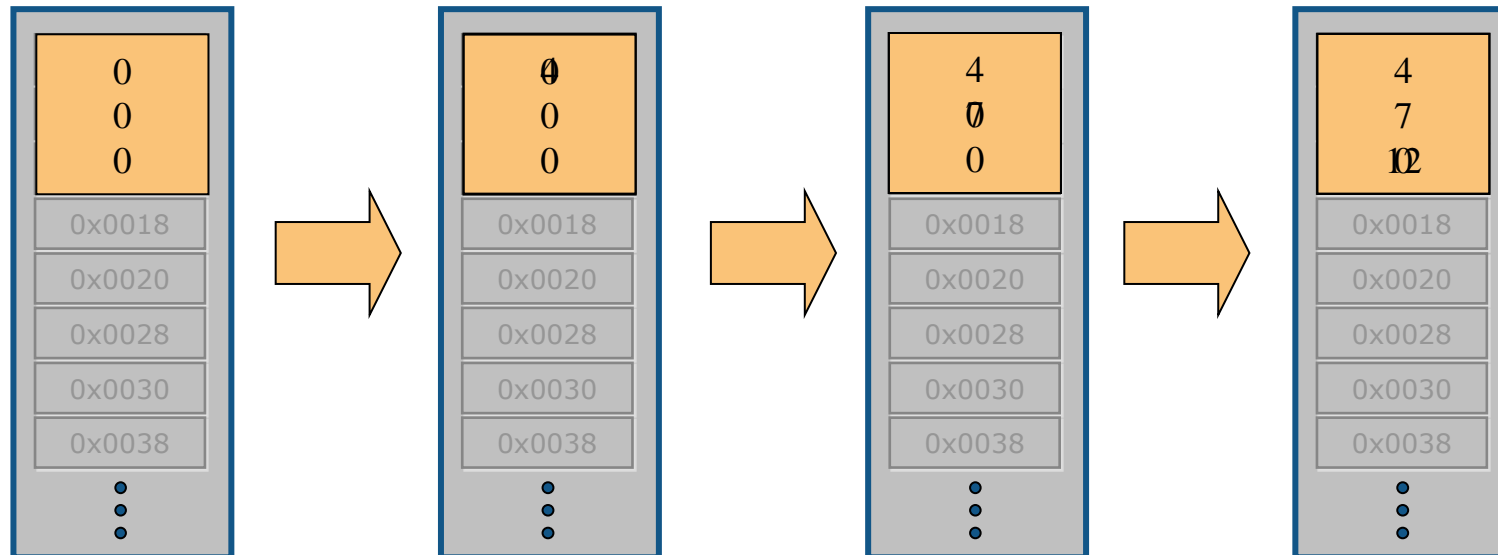
Benefit of Preallocation

```
x = zeros(3,1)
```

```
x(1) = 4
```

```
x(2) = 7
```

```
x(3) = 12
```



Demo: Preallocation

```
tic
x = 0;
for k = 2:1000000
    x(k) = x(k-1) + 5;
end
toc
```

Elapsed time is 0.301528 seconds

```
tic
x = zeros(1, 1000000);
for k = 2:1000000
    x(k) = x(k-1) + 5;
end
toc
```

Elapsed time is 0.011938 seconds.

Vectorization

Vectorizing a scalar loop

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```



```
t = 0:.01:10;
y = sin(t);
```

Array operations: leveraging operators and implicit expansion

```
mA = mean(A);
B = zeros(size(A));
for n = 1:size(A,2)
    B(:,n) = A(:,n) - mA(n);
end
```



```
devA = A - mean(A)
```

Logical array operations: logical indexing

```
for c = 1:N
    for r = 1:N
        if A(r,c) > myRef
            B(r,c) = A(r,c);
        end
    end
end
```



```
B(A > myRef) = A(A > myRef);
```

A(logical)

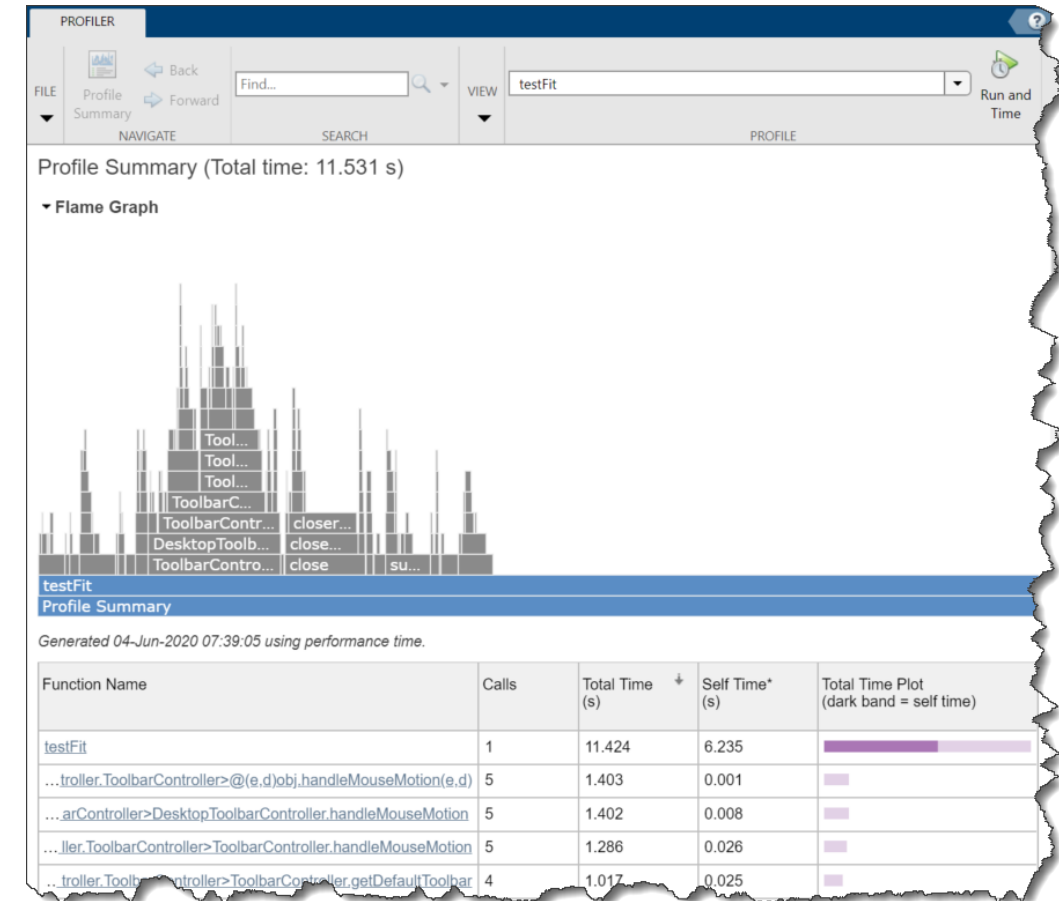
Matrix operations, sorting, etc.

Functions Commonly Used in Vectorization

Function	Description
all	Determine if all elements are non-zero
any	Determine if any elements are non-zero
cumsum	Cumulative sum
diff	Difference between adjacent elements
find	Find indices of non-zero elements
ind2sub	Convert linear indices to subscripts
ipermute	Inverse permutation
logical	Convert numeric values to logical
meshgrid	Create mesh grid
ndgrid	Create N-D mesh grid
permute	Rearrange dimensions
prod	Product of elements
repmat	Repeat array
reshape	Reshape array
shiftdim	Shift dimensions
sort	Sort array
squeeze	Remove singleton dimensions
sub2ind	Convert subscripts to linear indices
sum	Sum of elements

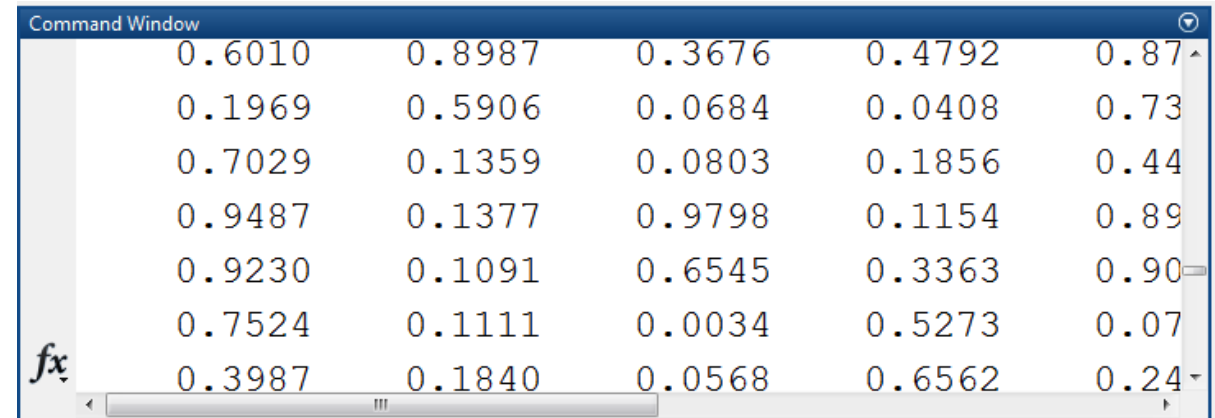
Profiler

- Total number of function calls
- Time per function call
- Self time in a function call
- Statement coverage of code



Best Practices

- Minimize file I/O
- Reuse existing graphics components
- Avoid printing to Command Window



0.6010	0.8987	0.3676	0.4792	0.87
0.1969	0.5906	0.0684	0.0408	0.73
0.7029	0.1359	0.0803	0.1856	0.44
0.9487	0.1377	0.9798	0.1154	0.89
0.9230	0.1091	0.6545	0.3363	0.90
0.7524	0.1111	0.0034	0.5273	0.07
0.3987	0.1840	0.0568	0.6562	0.24

Outline

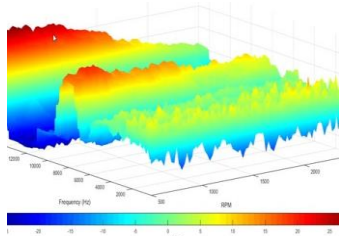
- Before we go parallel: Code optimization for better performance
- How to write and run parallel MATLAB code on your local machine or a cluster
- How to run MATLAB code on GPUs to accelerate your computations
- Use big data processing methods to handle large datasets
- Learning Resources

Practical application of parallel computing

- Why parallel computing?
 - Need faster insight to bring competitive products to market quickly
 - Computing infrastructure is broadly available (multicore desktops, GPUs, clusters)

- Why parallel computing with MATLAB?
 - Leverage computational power of more hardware
 - Accelerate workflows with minimal to no code changes to your original code
 - Focus on your engineering and research, not the computation

Results for scaling to more CPU cores and GPUs



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

FICC Department of China Galaxy Securities Valuates Financial Assets and Builds Training and Hedging Strategies

Challenge

Perform large amounts of complicated quantitative analysis under time pressure

Solution

Use MATLAB and its finance toolboxes to model and develop investment strategies

Results

- Run times reduced to a couple of minutes
- Quality of work improved
- Work efficiency increased



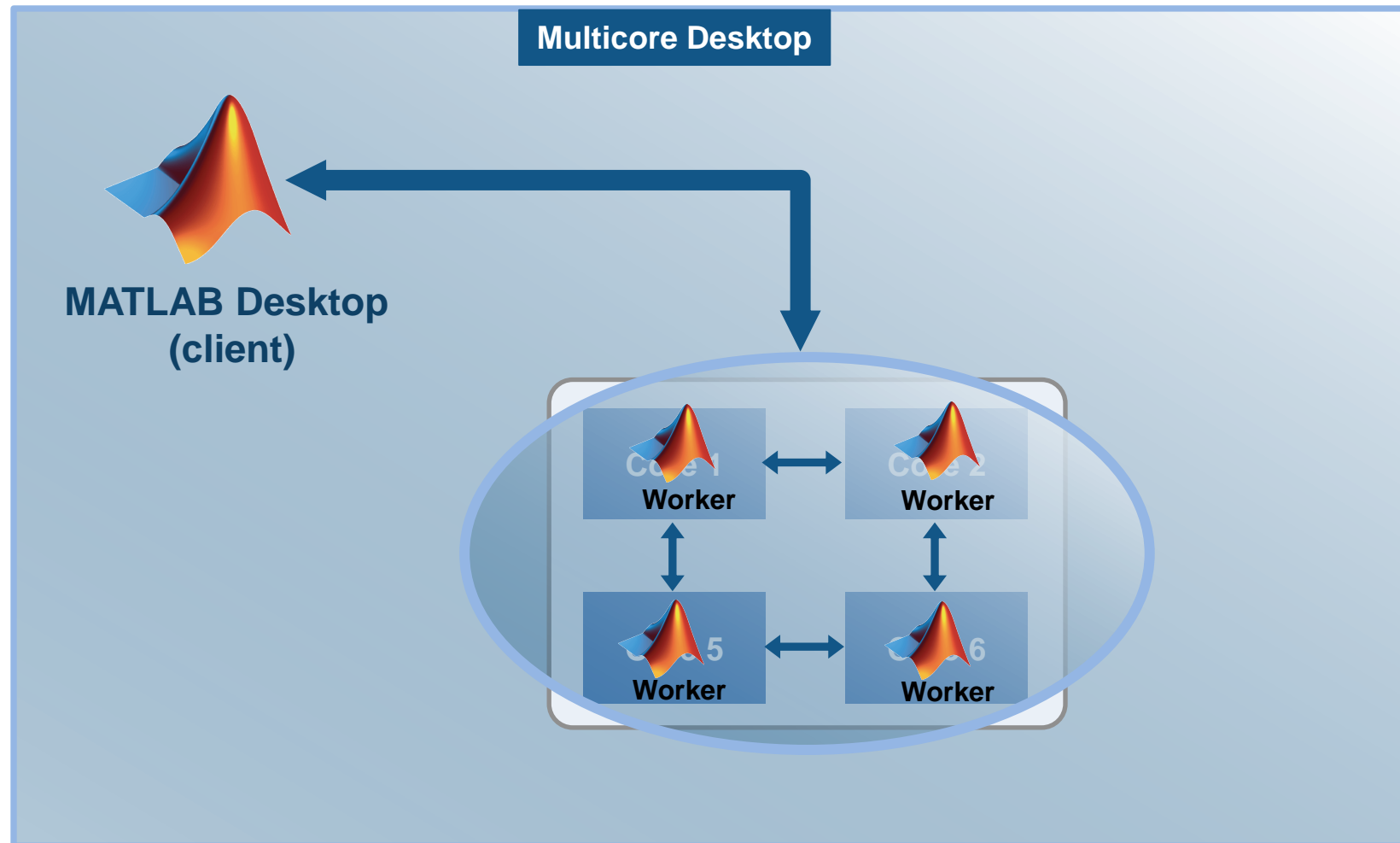
The FICC department of China Galaxy Securities applies MATLAB in their investment strategies.

"Investments in fixed income, currency, and commodities require large amounts of quantitative analysis. MATLAB supports us with highly efficient development tools for the valuation of financial instruments, statistical analysis of financial data, and the building and backtesting of quantitative strategies."

- Xian Xingchi, China Galaxy Securities

Parallel Computing Paradigm

Multicore Desktops



Accelerating MATLAB Applications



Ease of Use

Parallel-enabled toolboxes

```
('UseParallel', true)
```

Common programming constructs

Advanced programming constructs



Greater Control

Enable parallel computing support by setting a flag or preference

Automatic parallel support (*MATLAB*)

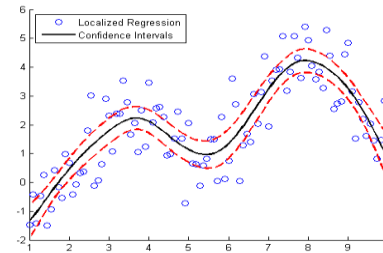
Image Processing

Batch Image Processor, Block Processing, GPU-enabled functions



Statistics and Machine Learning

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



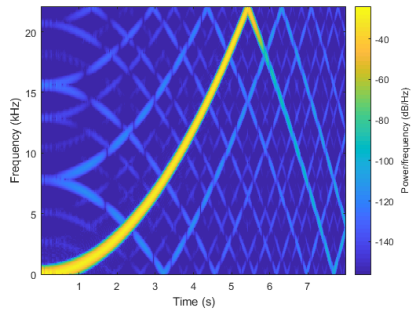
Deep Learning

Deep Learning, Neural Network training and simulation



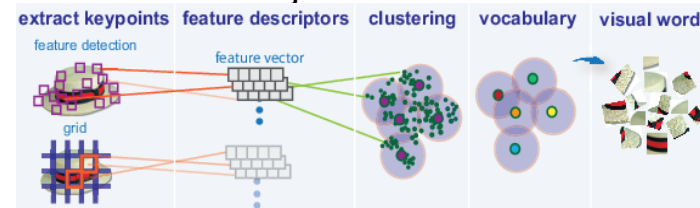
Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



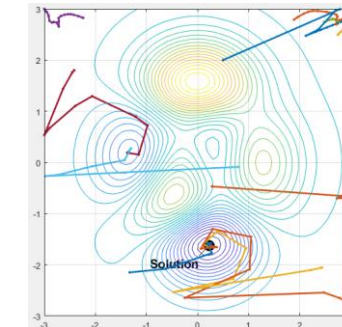
Computer Vision

Bag-of-words workflow, object detectors



Optimization and Global Optimization

Estimation of gradients, parallel search



[Additional automatic parallel support](#)

Classification Learner

The screenshot shows the Classification Learner app interface. The top toolbar includes buttons for 'New Session', 'Open', 'Save', 'Feature Selection', 'PCA', 'Costs', 'Optimizer', 'All Quick-To-Train', 'All', 'Summary', 'Duplicate', 'Delete', 'Results Table', 'Use Parallel' (highlighted with a red box), 'Train', 'Scatter', 'Confusion Matrix', 'Layout', 'Test Data', 'Test All', and 'EXPORT'. A yellow callout points to the 'Use Parallel' button with the text 'Click here to switch to parallel'. The main panel shows a list of models on the left and a summary for 'Model 2.1: SVM' on the right. A yellow callout points to the 'Linear SVM' model in the list with the text 'Use Classification Learner to train models in parallel'. The summary for 'Model 2.1: SVM' shows the status as 'Training' and lists hyperparameters: Preset: Linear SVM, Kernel function: Linear, Kernel scale: Automatic, Box constraint level: 1, Multiclass method: One-vs-One, Standardize data: Yes. It also shows feature selection status: 12/12 individual features selected, PCA: Disabled, Misclassification Costs: Default, and Optimizer: Not applicable.

Classification Learner - untitled*

CLASSIFICATION LEARNER

Options: New Session, Open, Save, Feature Selection, PCA, Costs, Optimizer

Models: All Quick-To-Train, All, Summary, Duplicate, Delete, Results Table, **Use Parallel**, Train, Scatter, Confusion Matrix, Layout, Test Data, Test All, EXPORT

Model Number: 2.1

Model 2.1: SVM
Status: Training

Model Hyperparameters:

- Preset: Linear SVM
- Kernel function: Linear
- Kernel scale: Automatic
- Box constraint level: 1
- Multiclass method: One-vs-One
- Standardize data: Yes

Feature Selection: 12/12 individual features selected

PCA: Disabled

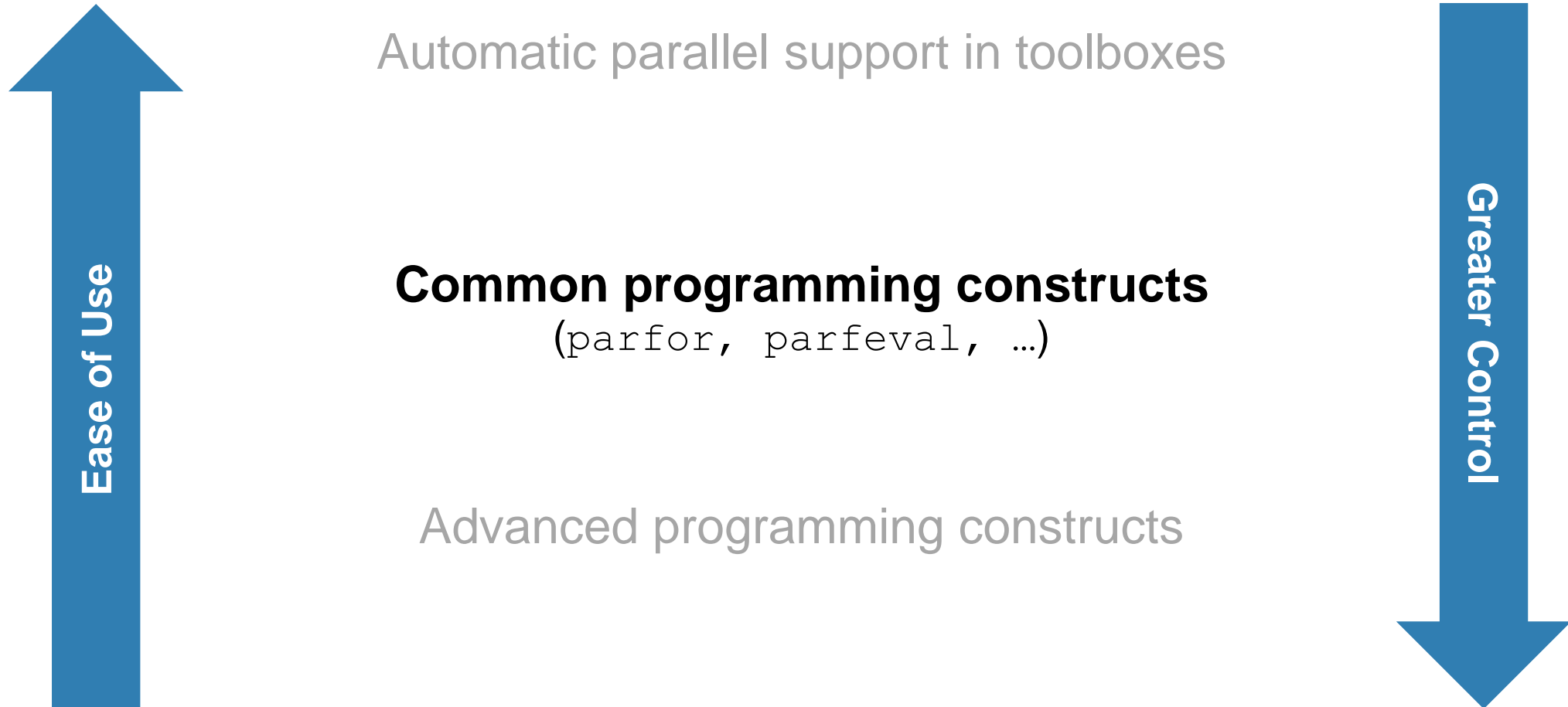
Misclassification Costs: Default

Optimizer: Not applicable

Model List:

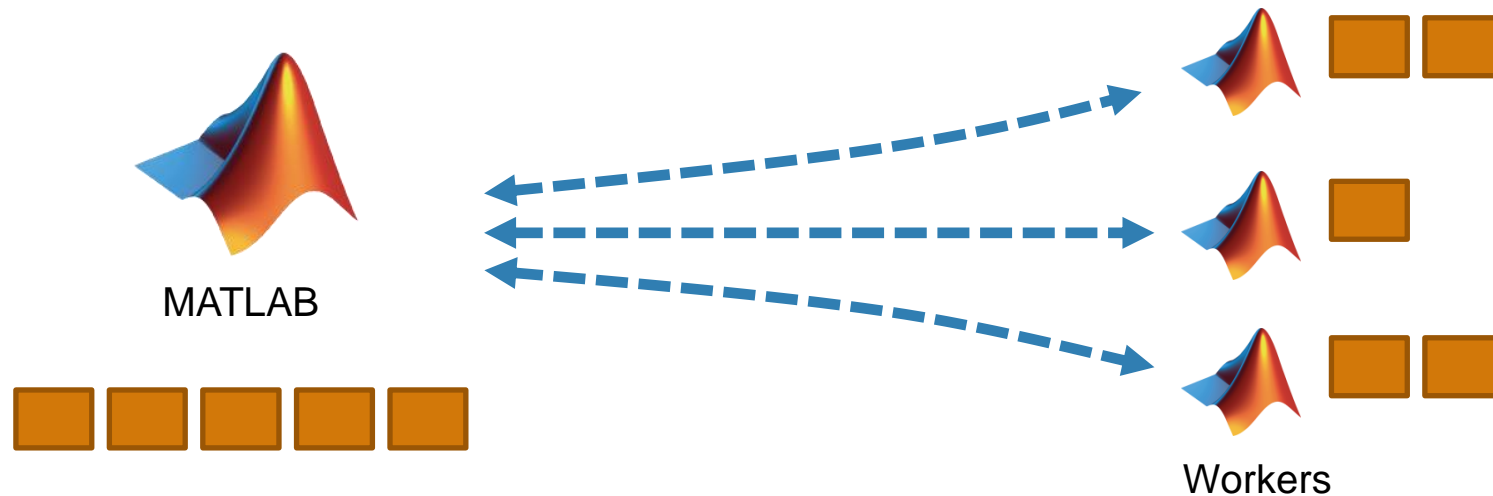
- 2.1 SVM: Training (12/12 features)
- 2.2 SVM: Training (12/12 features)
- 2.3 SVM: Training (12/12 features)
- 2.4 SVM: Training (12/12 features)
- 2.5 SVM: Queued (12/12 features)
- 2.6 SVM: Queued (12/12 features)

Scaling MATLAB applications and Simulink simulations



Run independent iterations in parallel using `parfor`

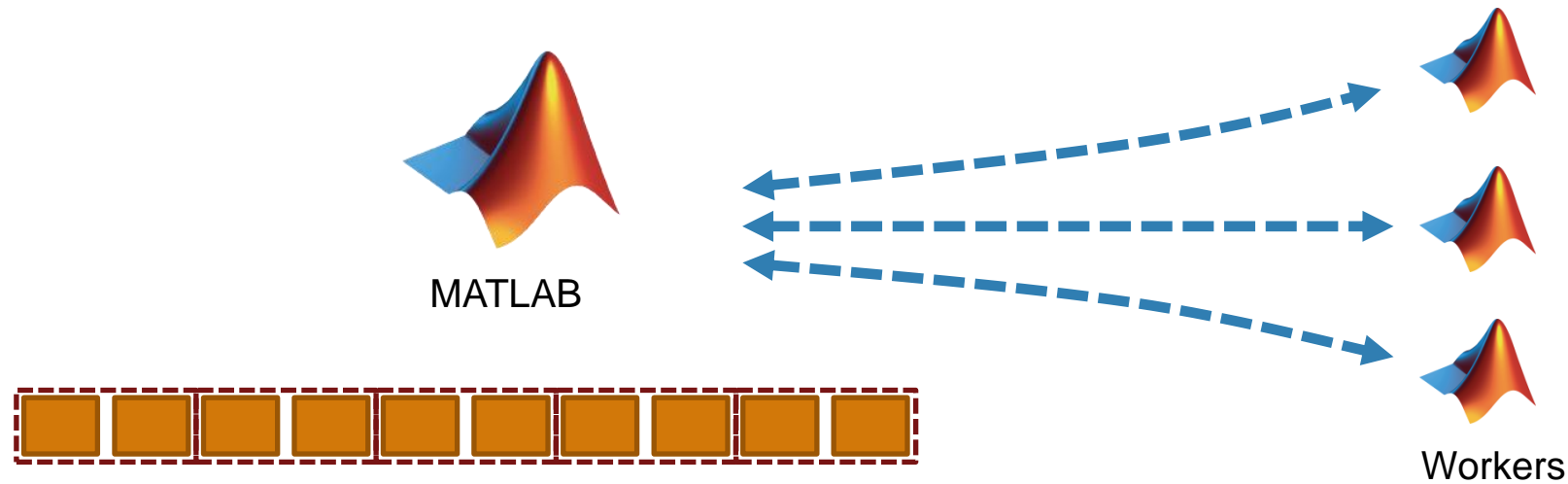
Use cases: Parameter sweeps and Monte Carlo simulations



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

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

Parallelize for loops with independent iterations



```
a = zeros(10, 1);  
b = pi;
```

```
parfor i = 1:10  
    a(i) = i + b;  
end
```

```
disp(a)
```

Demo: Compute the largest eigenvalue (serial)

```
>> edit ex_serial
```

```
function a = ex_serial(M, N)
% EX_SERIAL performs N trials of
%   computing the largest eigenvalue
%   for an M-by-M random matrix

a = zeros(N,1);
for I = 1:N
    a(I) = max(eig(rand(M)));
end
```

Demo: Compute the largest eigenvalue (parallel)

```
>> edit ex_parallel
```

```
function a = ex_parallel(M, N)
% EX_PARALLEL performs N trials of
% computing the largest eigenvalue
% for an M-by-M random matrix
```

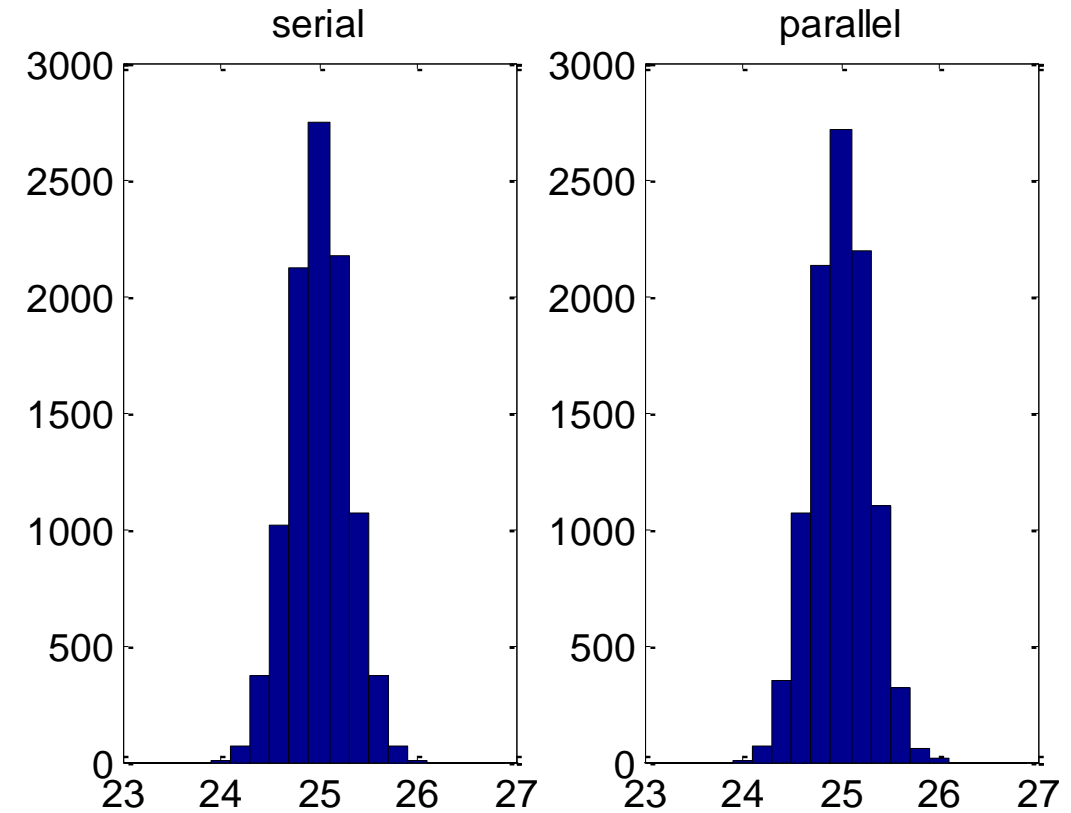
```
a = zeros(N,1);
parfor I = 1:N
    a(I) = max(eig(rand(M)));
end
```

The only modification!

Performance Comparison

```
>> edit ex_compare  
>> ex_compare
```

Serial processing time: 8.6s
Parallel processing time: 2.7s



Minor modifications resulted in a significant increase in speed.

parfor limitation

Noninteger loop variable



```
parfor x = 0:0.1:1
```

Nested parallel loops



```
parfor y = 2:10
```

```
    A(y) = A(y-1) + ...
```

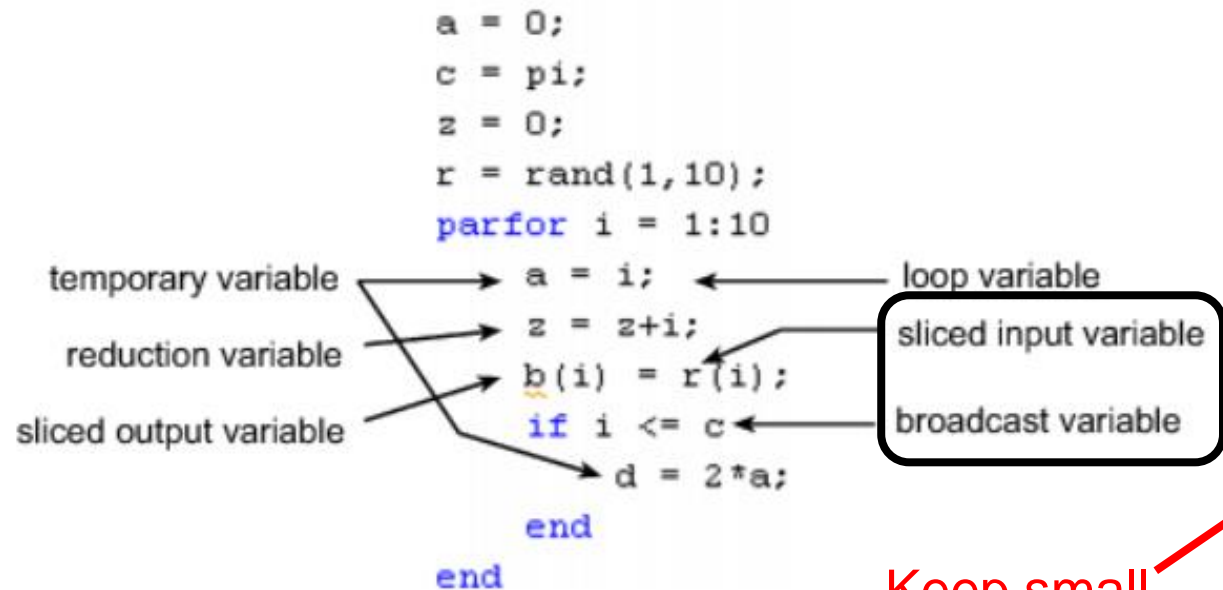


```
end
```

Dependent loop body

```
end
```

Optimizing parfor



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

Use more

Keep small

Scaling MATLAB applications and Simulink simulations



Ease of Use

Automatic parallel support in toolboxes

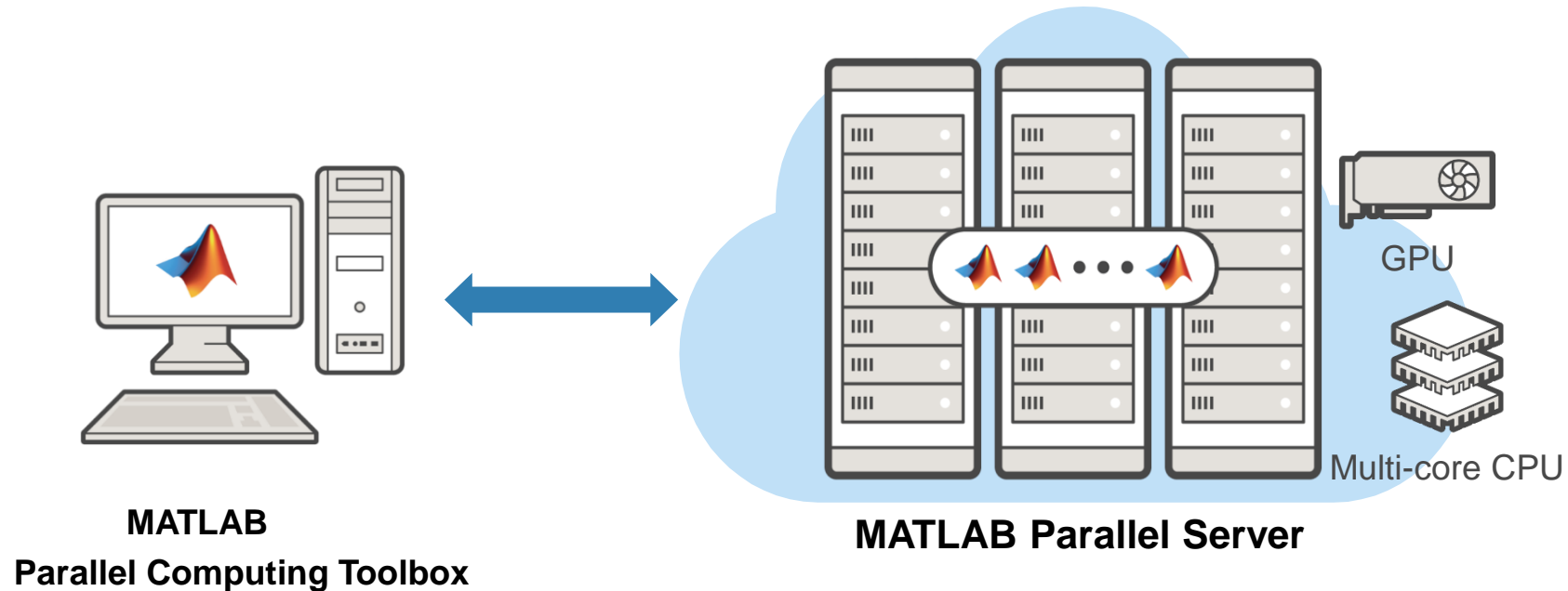
Common programming constructs

Advanced programming constructs
(`spmd`, `spmdBarrier`, ...)



Greater Control

Parallel computing on your desktop, clusters, and clouds

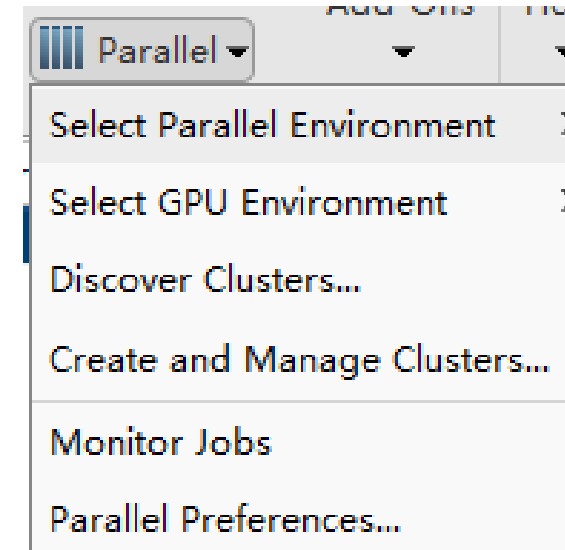


- Prototype and develop on the desktop
- Integrate with your infrastructure
- Access directly through MATLAB

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



Interactive parallel computing

Leverage cluster resources in MATLAB

```
>> parpool(myCluster, 3)
```

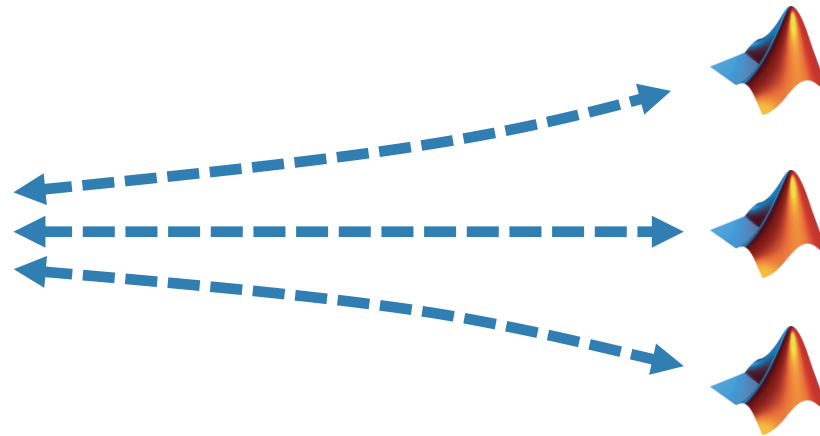
```
>> myScript
```

myScript.m:

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



MATLAB
Parallel Computing Toolbox

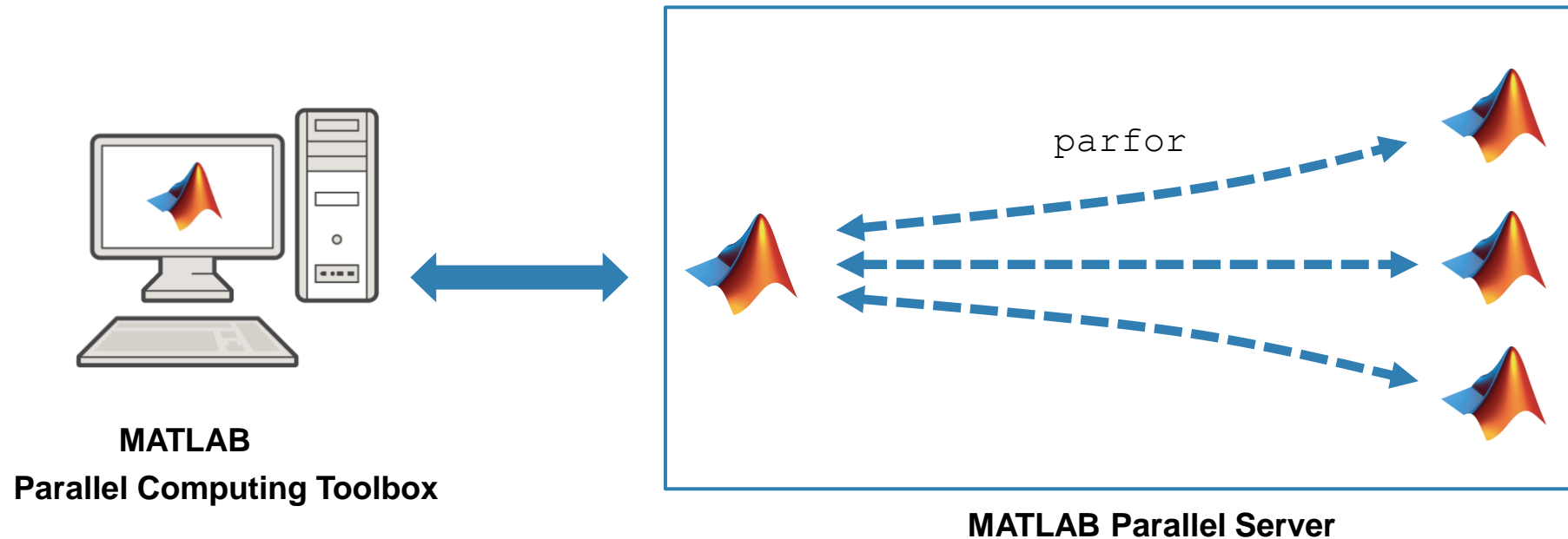


MATLAB Parallel Server

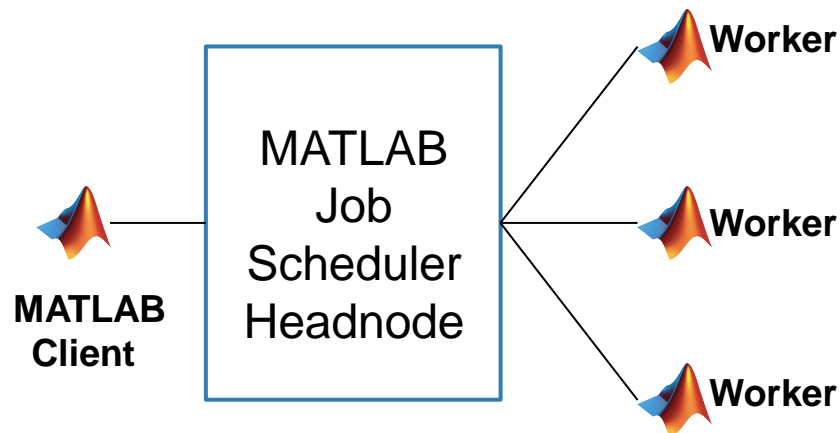
batch simplifies offloading computations

Submit MATLAB jobs to the cluster

```
>> job = batch(myCluster, 'myScript', 'Pool', 3)
```



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



Admin Center

FileHostsSchedulerWorkersHelp

Hosts

Add or Find...
Start MJS Service...
Stop MJS Service...
Test Connectivity...

Host			MJS Service		MATLAB Job Sche...	Workers
Hostname	Reachable	Cores	Status	Up Since	Name	Count
ComputeNodeHostname	yes	6	running	2019-01-03 15:55		4
HeadNodeHostname	yes	6	running	2019-01-03 15:53	myJobManager	0

MATLAB Job Scheduler

Start...
Stop...
Resume

Name	Hostname	Status	Up Since	Workers
myJobManager	HeadNodeHostname	running	2019-01-03 16:03	4

Workers

Start...
Stop...
Resume

Worker				MATLAB Job Scheduler			
Name	Hostname	Status	Up Since	Connection	Name	Hostname	
Worker1	ComputeNode...	idle	2019-01-03 16:03	connected	myJobManager	HeadNodeHos...	
Worker2	ComputeNode...	idle	2019-01-03 16:04	connected	myJobManager	HeadNodeHos...	
Worker3	ComputeNode...	idle	2019-01-03 16:04	connected	myJobManager	HeadNodeHos...	
Worker4	ComputeNode...	idle	2019-01-03 16:04	connected	myJobManager	HeadNodeHos...	

Last updated: 03/01/19 16:04

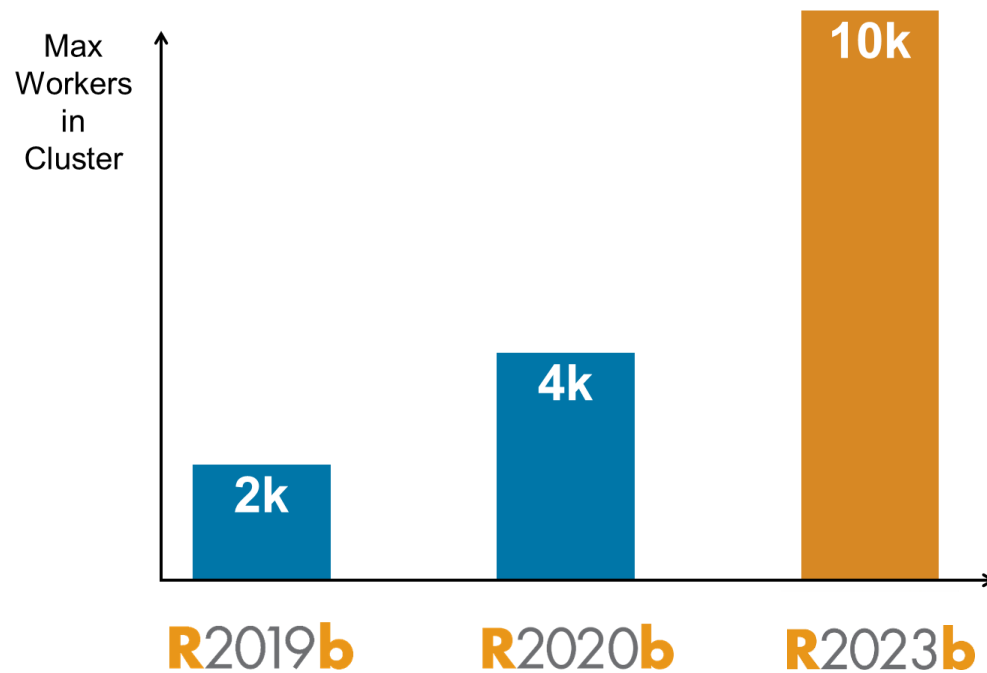
Updateevery 2 minutesUpdate Now

[Additional information](#)

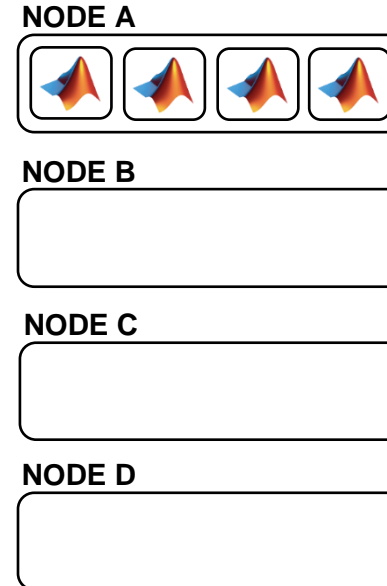
If you need to install parallel server on your cluster, please contact our dedicated installation support at info@mathworks.cn

Improvements and updates to MATLAB Job Scheduler scalability and scheduling

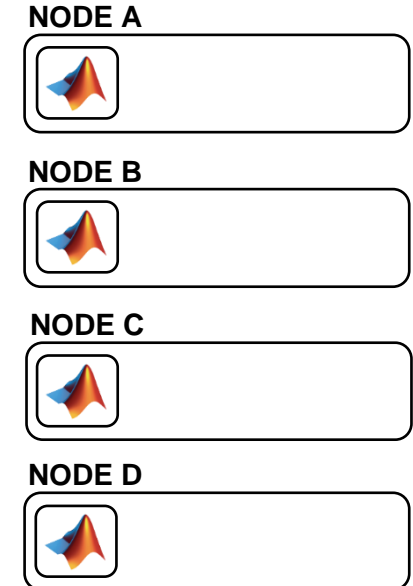
- MATLAB Job Scheduler supports clusters with up to 10K workers
- Control load-balancing scheduling algorithm on MATLAB Job Scheduler



Standard



loadBalancing



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](#)

If you need to install parallel server on your cluster, please contact our dedicated installation support at info@mathworks.cn

Learn More

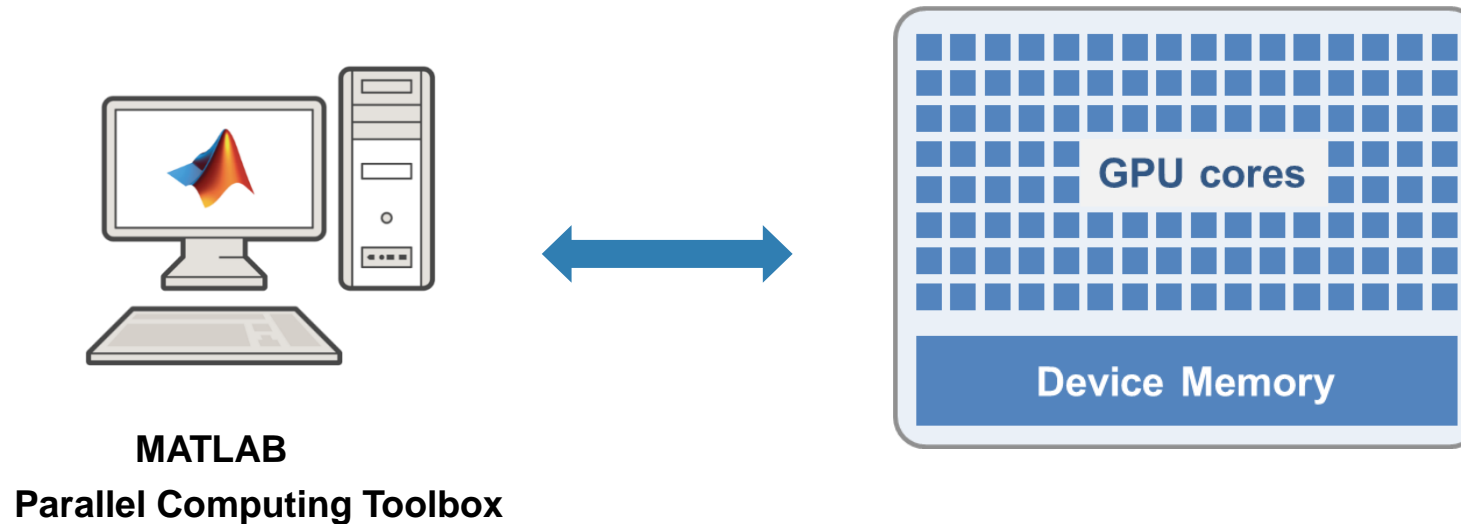
- Getting started with parallel computing
 - [Parallel Computing Toolbox](#)
 - [Performance and Memory](#)

- Scaling to the cluster and cloud
 - [MATLAB Parallel Server](#)
 - [MATLAB Parallel Server on the cloud](#)

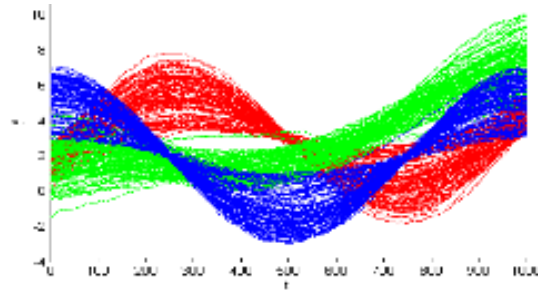
Outline

- Before we go parallel: Code optimization for better performance
- How to write and run parallel MATLAB code on your local machine or a cluster
- How to run MATLAB code on GPUs to accelerate your computations
- Use big data processing methods to handle large datasets
- Learning Resources

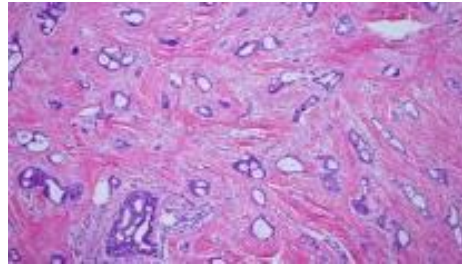
Leverage NVIDIA GPUs without learning CUDA



Results for accelerating with NVIDIA GPUs



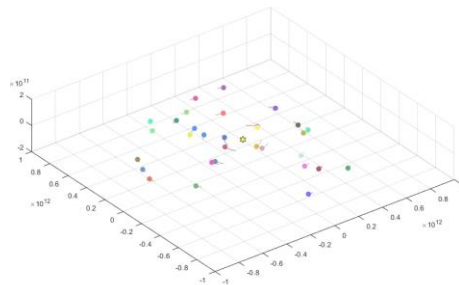
10x speedup
K-means clustering algorithm



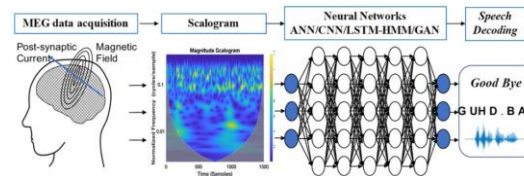
14x speedup
template matching routine



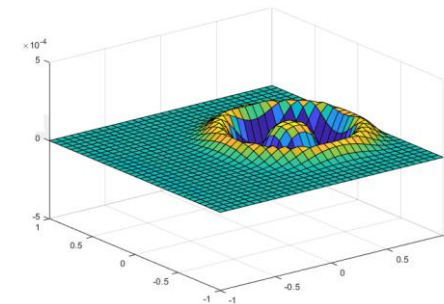
12x speedup
using Black-Scholes model



44x speedup
simulating the movement of celestial objects



10x speedup
deep learning training



77x speedup
wave equation solving

[Run MATLAB functions on a GPU](#)

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

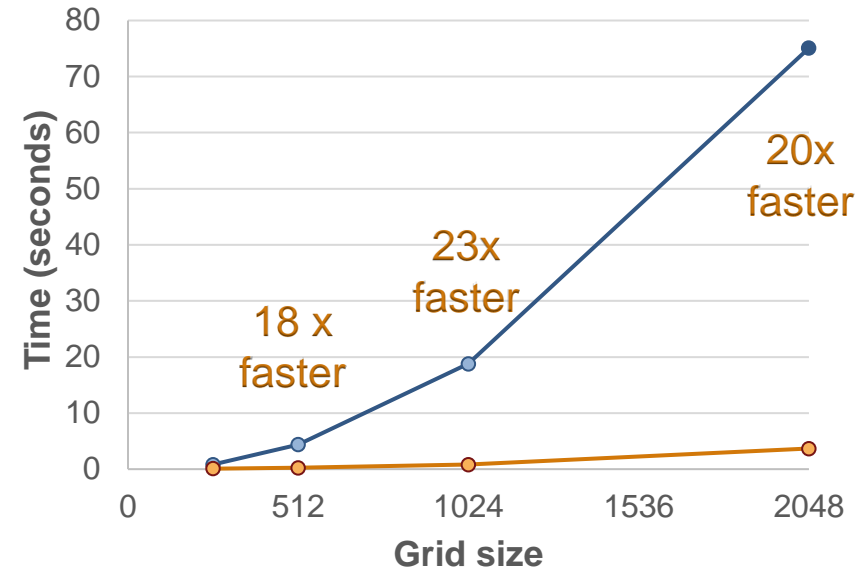
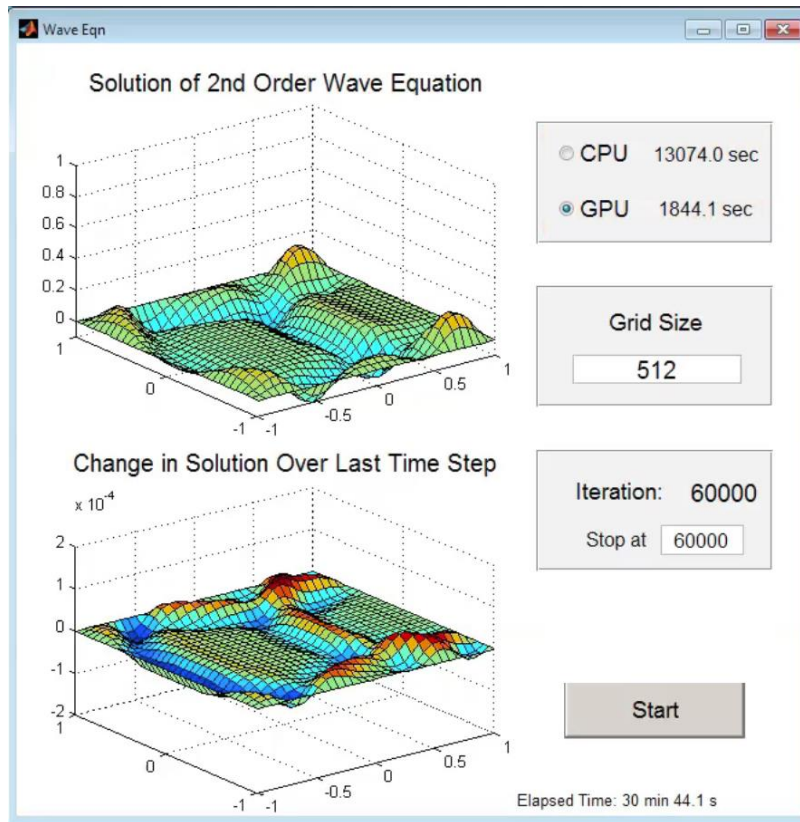


Wind tunnel test setup featuring the Hybrid Wing Body model (inverted), with 97-microphone phased array (top) and microphone tower (left).

“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

Run Same Code on CPU and GPU

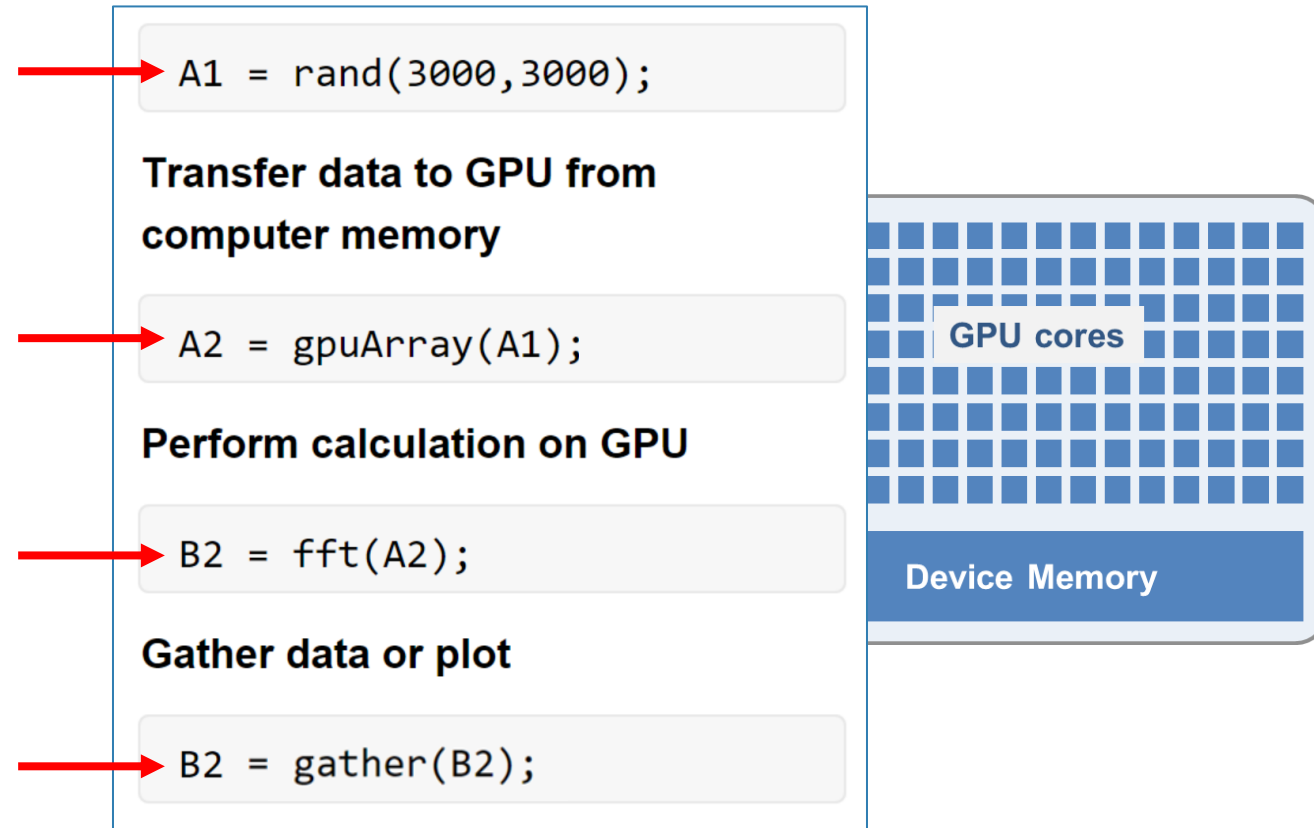
Solving 2D Wave Equation



CPU	GPU
Intel(R) Xeon(R) W3550 3.06GHz 4 cores memory bandwidth 25.6 Gb/s	NVIDIA Tesla K20c 706MHz 2496 cores memory bandwidth 208 Gb/s

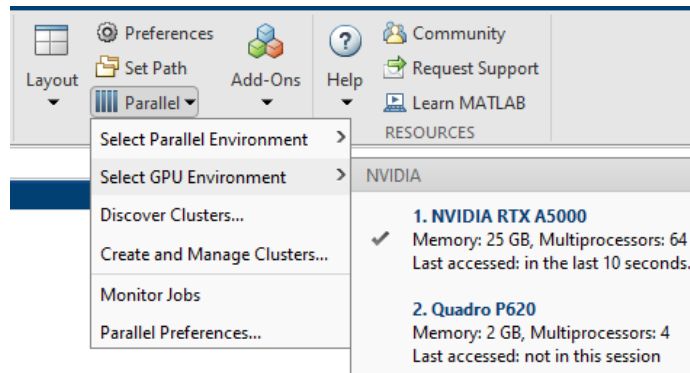
Leverage your GPU to accelerate your MATLAB code

- Ideal Problems
 - Massively parallel and/or vectorized operations
 - Computationally intensive
- 800+ GPU-supported functions
- Use `gpuArray` and `gather` to transfer data between CPU and GPU



Continued improvement to parallel control, usability, and performance

- [Select GPU](#) from Parallel menu



- [gpuArray support](#) – 15+ new features
- Performance enhancements for gpuArray



- Set [preferred workers](#) per profile
- [Thread-based parallel support](#) for 100+ additional features
- [Distributed Array support](#) – 4 new features, and new enhancements for conversion and distribution



Learn More

- [MATLAB with GPUs](#)

Outline

- Before we go parallel: Code optimization for better performance
- How to write and run parallel MATLAB code on your local machine or a cluster
- How to run MATLAB code on GPUs to accelerate your computations
- Use big data processing methods to handle large datasets
- Learning Resources

tall arrays

- New data type designed for data that doesn't fit into memory
- Lots of observations (hence “tall”)
- Looks like a normal MATLAB array
 - Supports numeric types, tables, datetimes, strings, etc.
 - Supports several hundred functions for basic math, stats, indexing, etc.
 - Statistics and Machine Learning Toolbox support (clustering, classification, etc.)

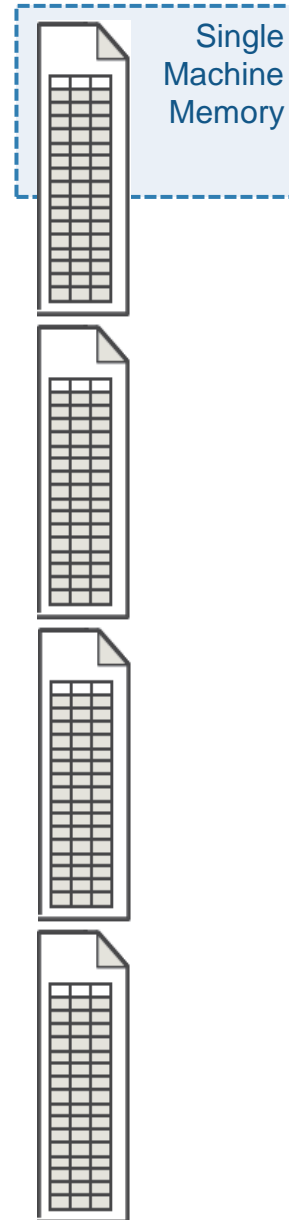


tall arrays

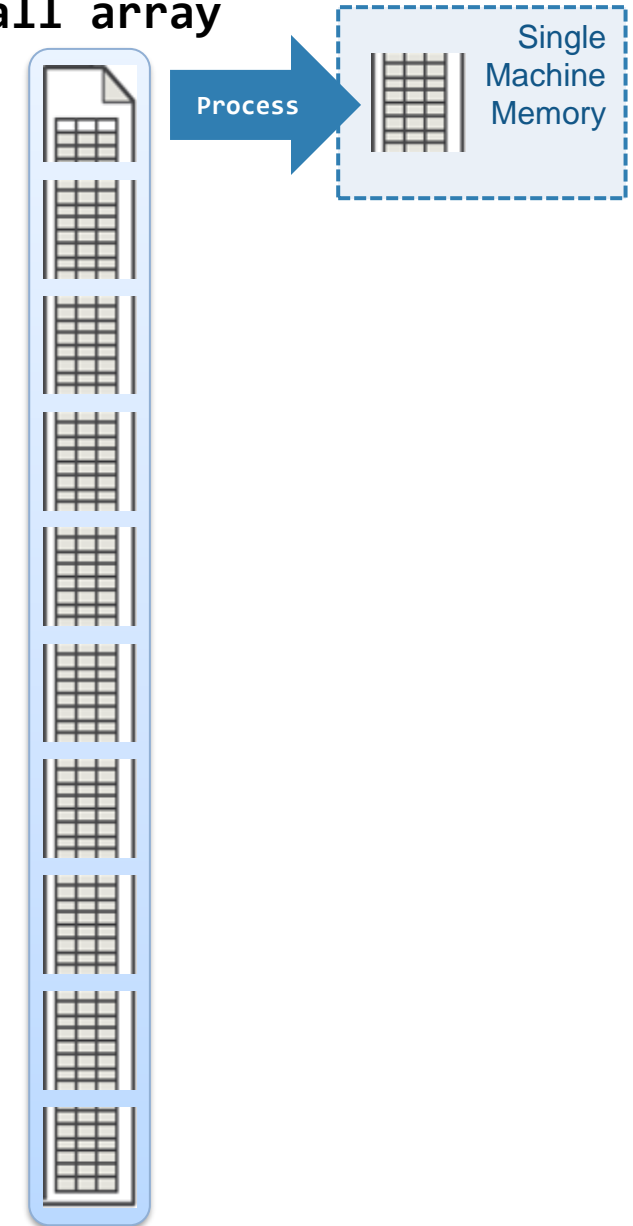
- Automatically breaks data up into small “chunks” that fit in memory
- Tall arrays scan through the dataset one “chunk” at a time

```
tt = tall(ds)
mDep = mean(tt.DepDelay, 'omitnan')
mDep = gather(mDep)
```

- Processing code for tall arrays is the same as ordinary arrays

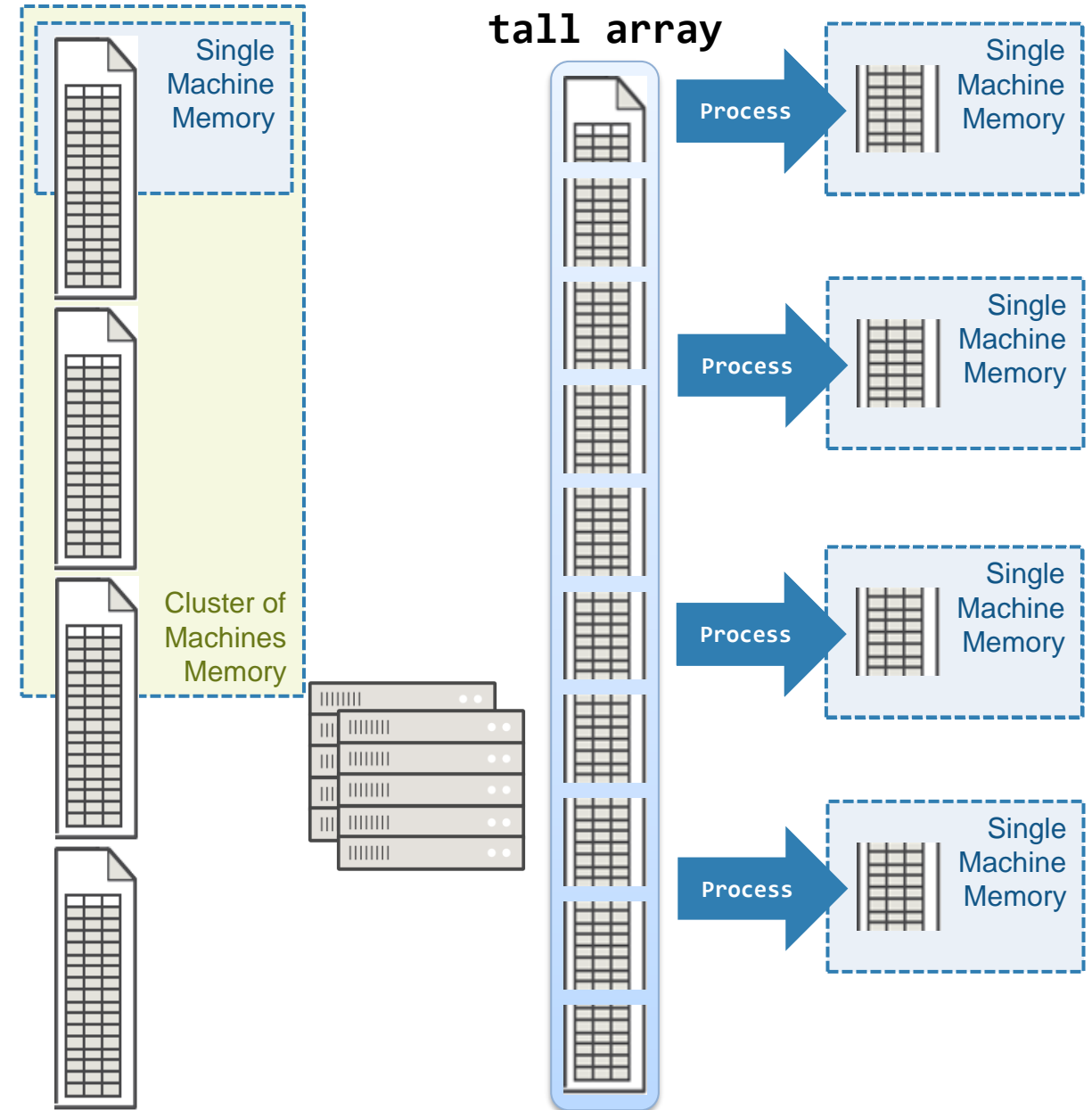


tall array



tall arrays

- With Parallel Computing Toolbox, process several “chunks” at once
- Can scale up to clusters with MATLAB Parallel Server

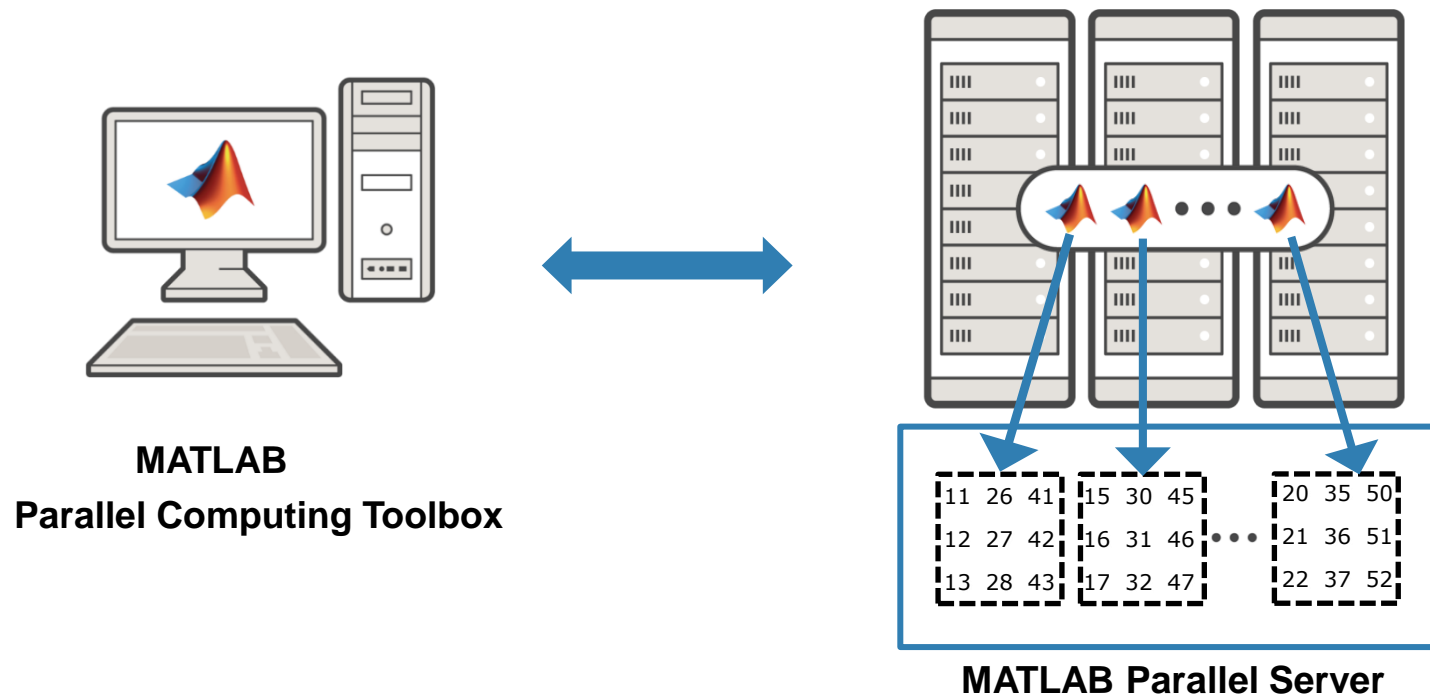


Demo: tall arrays

- https://www.mathworks.com/help/releases/R2023b/matlab/import_export/analyze-big-data-in-matlab-using-tall-arrays.html

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



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);
```

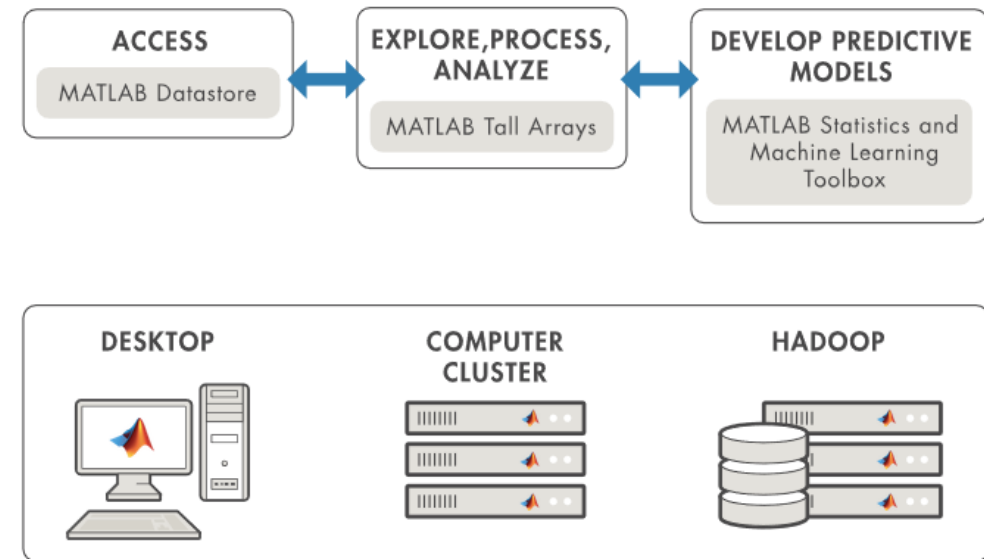
tall arrays vs. distributed arrays

- `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	✓	

Learn More: Big Data

- [Strategies for Efficient Use of Memory](#)
- [Resolving "Out of Memory" Errors](#)
- [Big Data with MATLAB](#)
- [MATLAB Tall Arrays in Action](#)



Demo files

- <https://mathworks-my.sharepoint.com/:f:/p/yueyixu/Er4mY1Ol8r5BooQT4eufh0AB-AEbn6G2dtqYqGI0M5CNZg?e=HWDbeC>

Outline

- Before we go parallel: Code optimization for better performance
- How to write and run parallel MATLAB code on your local machine or a cluster
- How to run MATLAB code on GPUs to accelerate your computations
- Use big data processing methods to handle large datasets
- Learning Resources

Documentation

Help

Documentation

Help Center

Search Help

CONTENTS

View By:

Category

Product List

Using MATLAB

MATLAB

Using Simulink

Simulink

Physical Modeling

Event-Based Modeling

Real-Time Simulation and Testing

Workflows

Parallel Computing

Reporting and Database Access

Documentation

Examples

Functions

Blocks

Apps

Documentation

R2023b

Using MATLAB

MATLAB

Using Simulink

Simulink

Physical Modeling

Applications

AI, Data Science, and Statistics

Mathematics and Optimization

Signal Processing

Image Processing and Computer Vision

Resources

Release Notes

Installation Help

Hardware Support

Community

Release Notes

The screenshot shows the MATLAB Help Center interface. At the top, there's a blue header with "Help Center" and a search bar labeled "Search Help". Below the header, a navigation bar contains links for "Documentation", "Examples", "Functions", "Blocks", and "Apps". The left sidebar shows a "CONTENTS" menu with a link to "« Documentation Home". The main content area is titled "Release Notes" in orange, with "R2023b" in the top right corner. Under "Explore Release Notes", there is a "Select a Product" dropdown. The "Online Resources" section lists links for "Installation and Licensing Changes", "System Requirements", "Bug Reports", and "Bug Fixes". The "New Products and Major Updates" section features three product cards: "Aerospace Toolbox" (Analyze and visualize aerospace vehicle motion using reference), "DO Qualification Kit" (Qualify Simulink® and Polyspace® verification tools for DO-178, DO-), and "Datafeed Toolbox" (Access financial data from data service providers). The browser's address bar at the bottom shows the file path: file:///C:/ProgramData/MATLAB/SupportPackages/R2023b/help/relnotes/index.html?s_tid=hc_resources.

Help Center

Search Help

Documentation Examples Functions Blocks Apps

« Documentation Home

Release Notes

R2023b

Explore Release Notes

Select a Product ▼

Online Resources

- Installation and Licensing Changes
- System Requirements
- Bug Reports
- Bug Fixes

New Products and Major Updates

Aerospace Toolbox

Analyze and visualize aerospace vehicle motion using reference

DO Qualification Kit

Qualify Simulink® and Polyspace® verification tools for DO-178, DO-

Datafeed Toolbox

Access financial data from data service providers

file:///C:/ProgramData/MATLAB/SupportPackages/R2023b/help/relnotes/index.html?s_tid=hc_resources

Self-Paced Online Training Courses

The screenshot shows the MATLAB Academy interface for the 'Deep Learning Onramp' course. The browser address bar shows the URL: <https://matlabacademy.mathworks.com/R2018b/portal.html?course=deeplearning#chapter=2&lesson=1§ion=1>. The course title is 'Deep Learning Onramp' with a progress indicator '(0% complete)'. The user is logged in as 'Stephen Frail'. The course is titled '2.1 Course Example - Identify Objects in Some Images'. The interface is divided into three main sections: a task pane on the left, a central workspace with a code editor and command window, and a workspace area on the right. The task pane shows 'Task 1' and 'Task 2'. Task 2 instructions state: 'You can use the `imshow` function to display an image stored in a MATLAB variable' and 'Display the imported image in the variable `img1`'. The task pane also includes a 'Submit' button and a 'Next task' button. The central workspace shows the code editor with the following code:

```
img1 = imread('file01.jpg')
```

 and

```
imshow(img1)
```

. The command window shows the output:

```
img1 = 227x227x3 uint8 array
```

 and

```
img1(:,:,1) =
```

 followed by a large matrix of numbers. The workspace area on the right shows a preview of the image loaded into `img1`, which is a landscape photo of a beach and trees.

High-quality, self-paced online training courses

Interactive learning environment provides the experience of using the product

Automated assessment and feedback

Measurable progress report and completion certificate

24/7 availability

Self-Paced Online Training Format and Learning Environment

- Learn anytime, anywhere: Access from the web browser or within the product
- Learn by doing: Complete tasks within MATLAB and Simulink. Receive immediate feedback
- Track progress and share certificate of completion

The screenshot displays the MATLAB Fundamentals online training environment. The top navigation bar shows 'MY COURSES' and 'MATLAB Fundamentals (2% complete)'. The current course is '4.3 Creating Evenly-Spaced Vectors: (4/8) Use Colon Operator and Linspace'. The interface is divided into two main sections: a task completion screen on the left and a task list on the right.

Task Completion Screen (Left):

- Task 1:** Create a variable named `x` that contains the row vector shown below.
- Row Vector:** `3 5 7 9 11`
- Buttons:** Hint | See Solution | Reset | Submit | Next task
- Test Results:** Correct!
 - ✓ Is `x` defined correctly?
 - ✓ Does script not contain square brackets?







Task List (Right):

- Task 1:** `x = 5:15`
- Task 2:** `x = linspace(5,15,13)`
- Task 3:** `x = 3:2:11`

Self-Paced Online Courses

Introductory Courses - Free to All

Get Started with MATLAB

-  MATLAB Onramp
-  Deep Learning Onramp
-  Machine Learning Onramp
-  Image Processing Onramp
-  Signal Processing Onramp
-  Statistics Onramp

-  App Building Onramp
-  Optimization Onramp
-  Computer Vision Onramp
-  Reinforcement Learning Onramp
-  Object Oriented Programming Onramp
-  Wireless Communications Onramp

Get Started with Simulink

-  Simulink Onramp
-  Simscape Onramp
-  Stateflow Onramp
-  Circuit Simulation Onramp
-  Power Systems Simulation Onramp

-  Power Electronics Simulation Onramp
-  Control Design Onramp with Simulink

Courses Included with Online Training Suite

MATLAB and Simulink

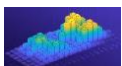


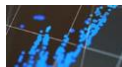




-  MATLAB Fundamentals
-  Simulink Fundamentals
-  MATLAB Programming Techniques
-  MATLAB for Data Processing and Visualization






Image and Signal Processing

-  Image Processing with MATLAB
-  Signal Processing with MATLAB

Data Science and AI

-  Deep Learning with MATLAB
-  Machine Learning with MATLAB

Computational Mathematics

-  Solving Nonlinear Equations with MATLAB
-  Solving Ordinary Differential Equations with MATLAB
-  Introduction to Symbolic Math with MATLAB
-  Introduction to Linear Algebra with MATLAB
-  Introduction to Statistical Methods with MATLAB



Learn more about MathWorks, our products, and our services at
mathworks.com and on social media:



Please fill out the survey to get access to the webinar materials

Q&A