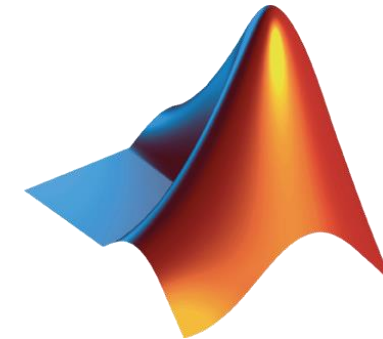


Parallel Computing Hands-On Workshop

MathWorks



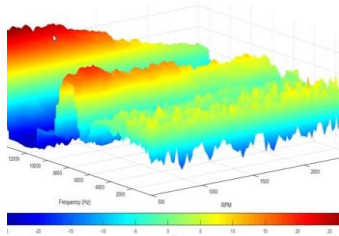
Agenda

- Part I – Parallel Computing with MATLAB on the Desktop
 - Parallel Computing Toolbox
- Part II – Scaling MATLAB to HPC
 - MATLAB Parallel Server

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

[User stories](#)

Optimizing before parallelizing

1. Find bottlenecks

2. Implement effective

3. (Ad

The screenshot shows the MATLAB Profiler window with a 'Profile Summary' for a function. The total time is 0.052 s. The code being profiled is as follows:

```

40 length(unique(data{2}(:))) == 1 ...
41 length(unique(data{3}(:))) == 1)
42 data{3} = zeros(size(data{1}));
43 dim(nl) = 2;
44 else
45 dim(nl) = 3;
46 end
47 % Do the actual computation
48 temp = diff([data{1}(:) data{2}(:) data{3}(:)
49 len(nl) == sum([sqrt(dot(temp',temp'))])
50 end
51 end
52
53 % If some indices are not lines, fill the results with
54 if any(notline(:))
55 warning('lengthofline:FillWithNaNs', ...
56 '\n%s of non-line objects are being filled with %s.' ...

```

The Profiler window highlights line 49 with a yellow warning icon. A tooltip box shows the following messages:

- ⚠ Line 49: Terminate statement with semicolon to suppress
- ⚠ Line 49: Use of brackets [] is unnecessary. Use parentheses

On the right, the MATLAB Documentation window is open to the 'mex' page. The 'Syntax' section shows the following command formats:

```

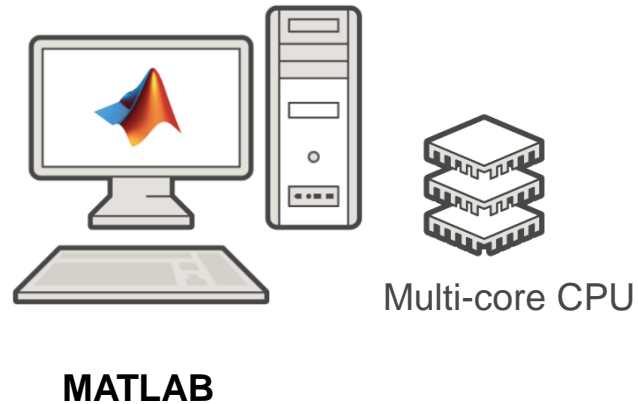
mex filenames
mex filenames api option1 ... optionN


mex -client engine filenames
mex -client engine filenames api option1 ... optionN

mex -setup [lang]
mex -setup -client engine [lang]

```

Multicore computing options for MATLAB



☰

MATLAB Multicore

🔍

Run MATLAB on multicore and multiprocessor machines

MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers. By using the full computational power of your machine, you can run your MATLAB applications faster and more efficiently.

Built-in Multithreading

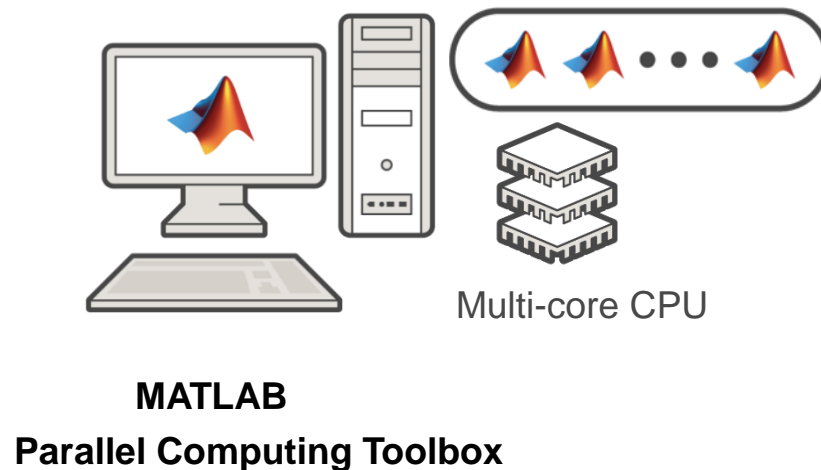
Linear algebra and numerical functions such as `fft`, `\(mldivide)`, `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session, allowing them to execute faster on multicore-enabled machines. Additionally, many functions in Image Processing Toolbox™ are multithreaded.

Parallelism Using MATLAB Workers

You can run multiple MATLAB workers (MATLAB computational engines) on a single machine to execute applications in parallel, with [Parallel Computing Toolbox™](#). This approach allows you more control over the parallelism than with built-in multithreading, and is often used for coarser grained problems such as running parameter sweeps in parallel.

[MATLAB multicore](#)

Multicore computing options for MATLAB



The screenshot shows the MATLAB Multicore page. At the top left is the MathWorks logo, and at the top right is a hamburger menu icon. Below the logo is the title 'MATLAB Multicore' in white text on a blue background, with a search icon on the right. The main content area has a sub-header 'Run MATLAB on multicore and multiprocessor machines' in orange. Below this is a paragraph: 'MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers. By using the full computational power of your machine, you can run your MATLAB applications faster and more efficiently.' This is followed by a section titled 'Built-in Multithreading' with a paragraph: 'Linear algebra and numerical functions such as `fft`, `\(mldivide)`, `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session, allowing them to execute faster on multicore-enabled machines. Additionally, many functions in Image Processing Toolbox™ are multithreaded.' A blue-bordered box contains the section 'Parallelism Using MATLAB Workers' with a paragraph: 'You can run multiple MATLAB workers (MATLAB computational engines) on a single machine to execute applications in parallel, with [Parallel Computing Toolbox™](#). This approach allows you more control over the parallelism than with built-in multithreading, and is often used for coarser grained problems such as running parameter sweeps in parallel.'

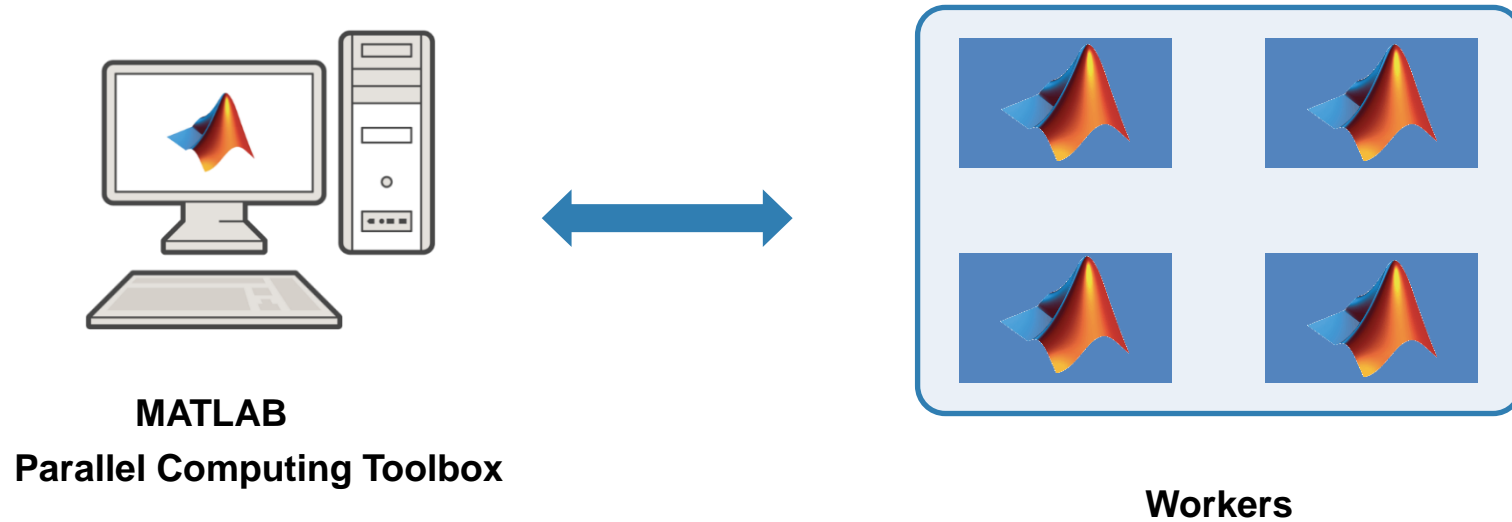
[MATLAB multicore](#)

Compute 40,000 iterations
van der Pol Equation study with `parfor`

Agenda

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

Utilizing multiple CPU cores



```

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

```

[Show fewer details](#)

Parallel pool on local has been running for about 1 min (since 14:12) and will shut down if still idle in 29 minutes. ([Reset to 30 minutes](#)) ([Shut down now](#))
 Number of workers: 4



```

1) ;
:5
b;
end
disp(a)

```

Scaling MATLAB applications and Simulink simulations



Automatic parallel support in toolboxes

Common programming constructs

Advanced programming constructs



Automatic parallel support (MATLAB)

Enable parallel computing support by setting a flag or preference

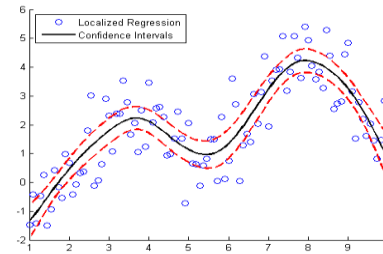
Image Processing

Batch Image Processor, Block Processing, GPU-enabled functions



Statistics and Machine Learning

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



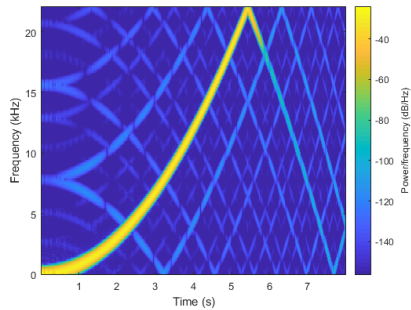
Deep Learning

Deep Learning, Neural Network training and simulation



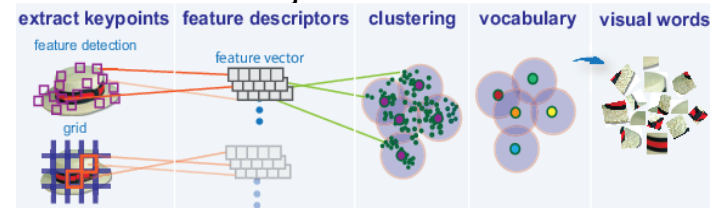
Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



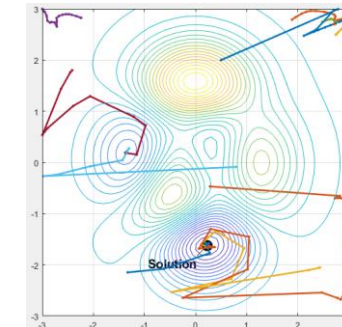
Computer Vision

Bag-of-words workflow, object detectors



Optimization and Global Optimization

Estimation of gradients, parallel search



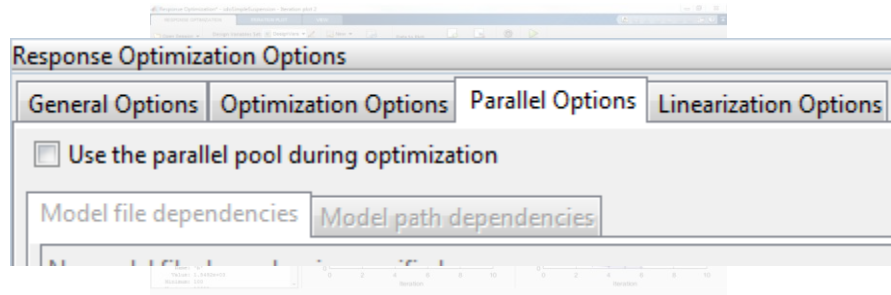
[Other automatic parallel supported toolboxes](#)

Automatic parallel support (Simulink)

Enable parallel computing support by setting a flag or preference

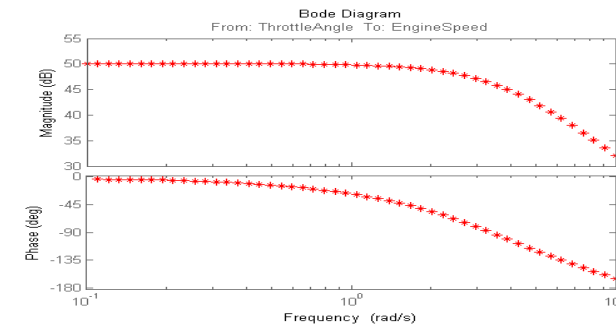
Simulink Design Optimization

Response optimization, sensitivity analysis, parameter estimation



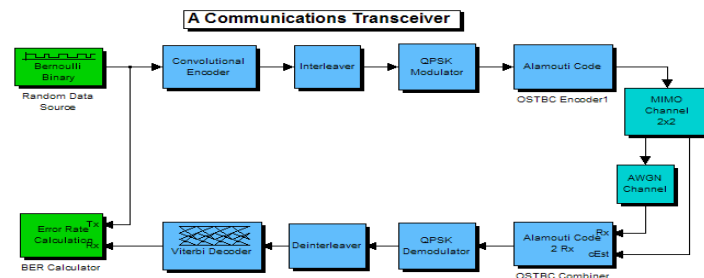
Simulink Control Design

Frequency response estimation



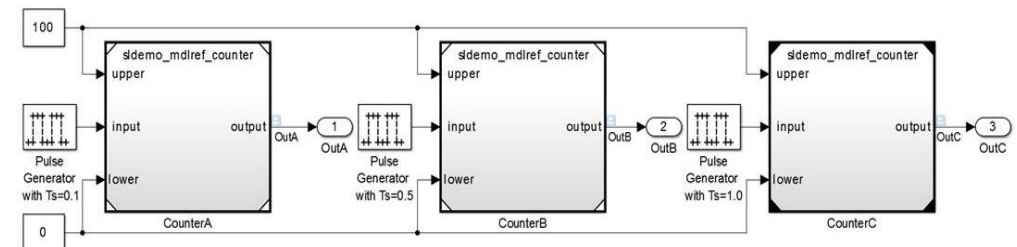
Communication Systems Toolbox

GPU-based System objects for Simulation Acceleration



Simulink/Embedded Coder

Generating and building code



[Other automatic parallel supported toolboxes](#)

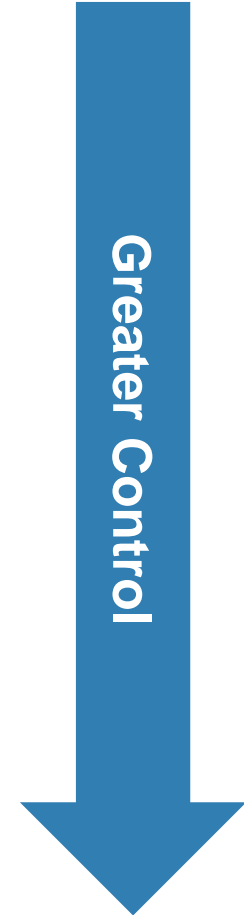
Scaling MATLAB applications and Simulink simulations



Automatic parallel support in toolboxes

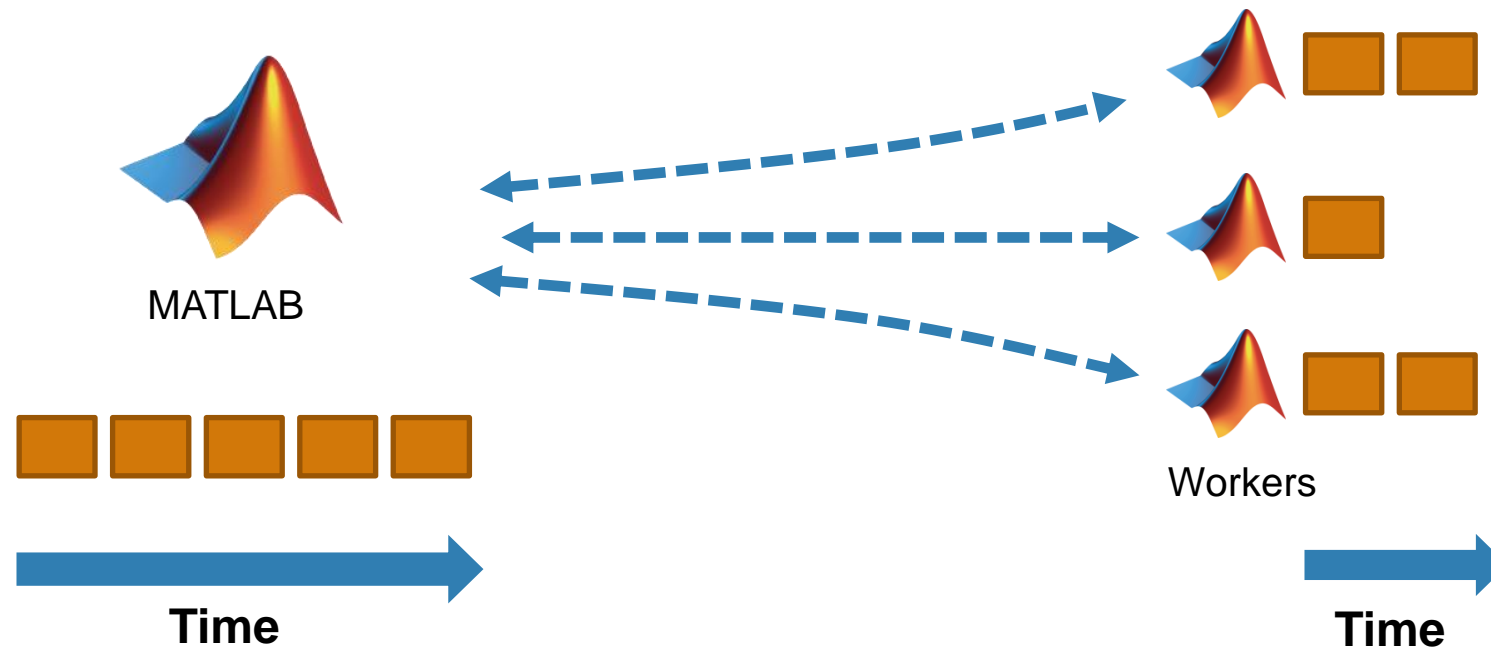
Common programming constructs
(`parfor`, `parfeval`, ...)

Advanced programming constructs



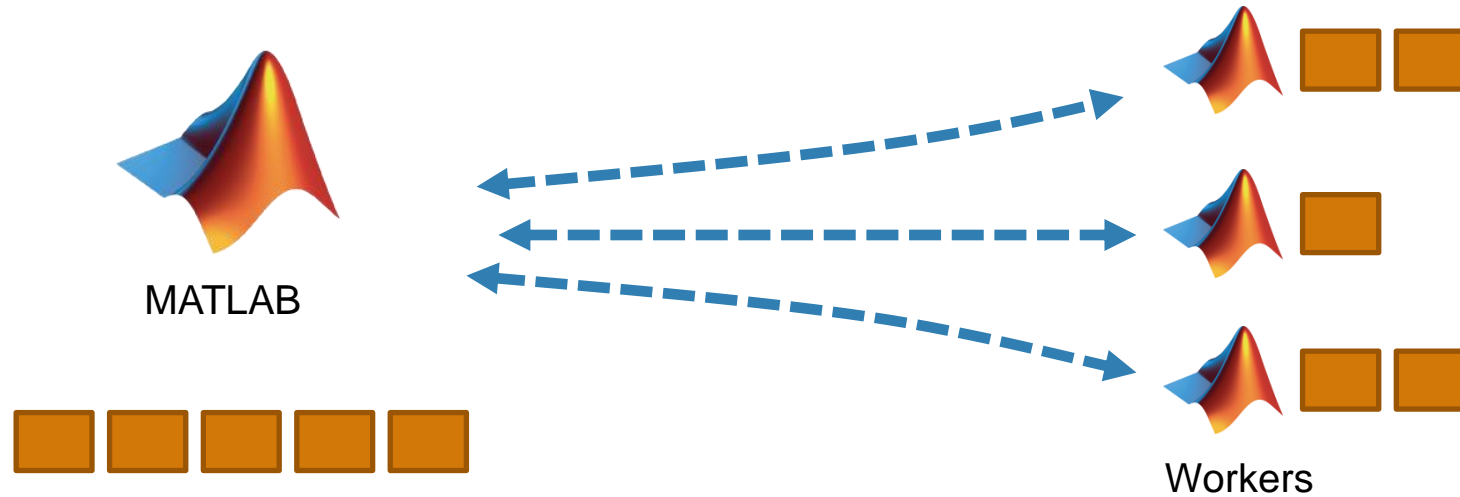
Parallelism using `parfor`

- Run iterations in parallel
- Examples: parameter sweeps, Monte Carlo simulations



[Learn more about parfor](#)

Parallelism using `parfor`



```
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)
```


Parallelism using `parfor`

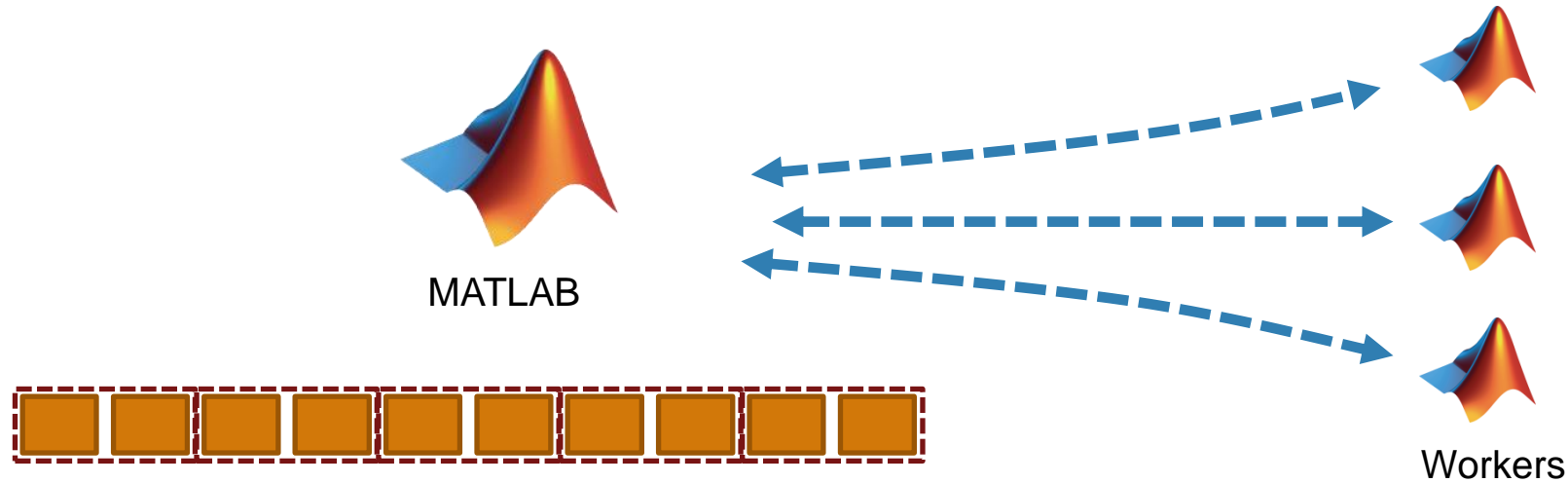
```
1 a = zeros(5, 1);
2 b = pi;
3 parfor i = 1:5
4     a(i) = i + b;
5 end
6 disp(a)
```

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.

Parallelism using `parfor`

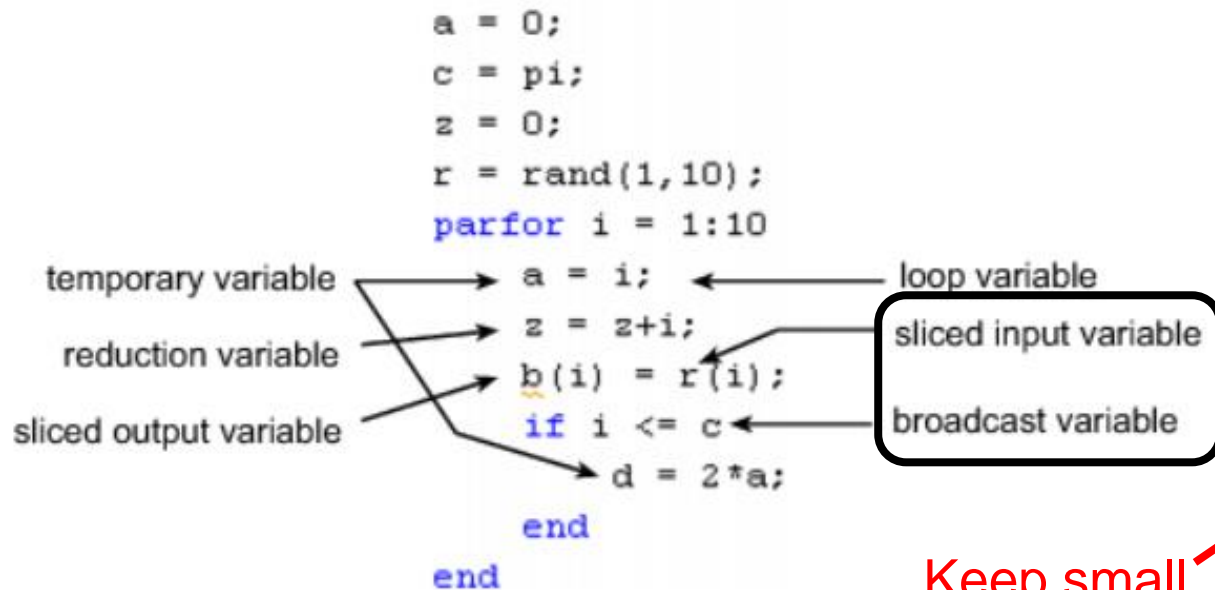


```
a = zeros(10, 1);  
b = pi;
```

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

```
disp(a)
```

Optimizing parfor

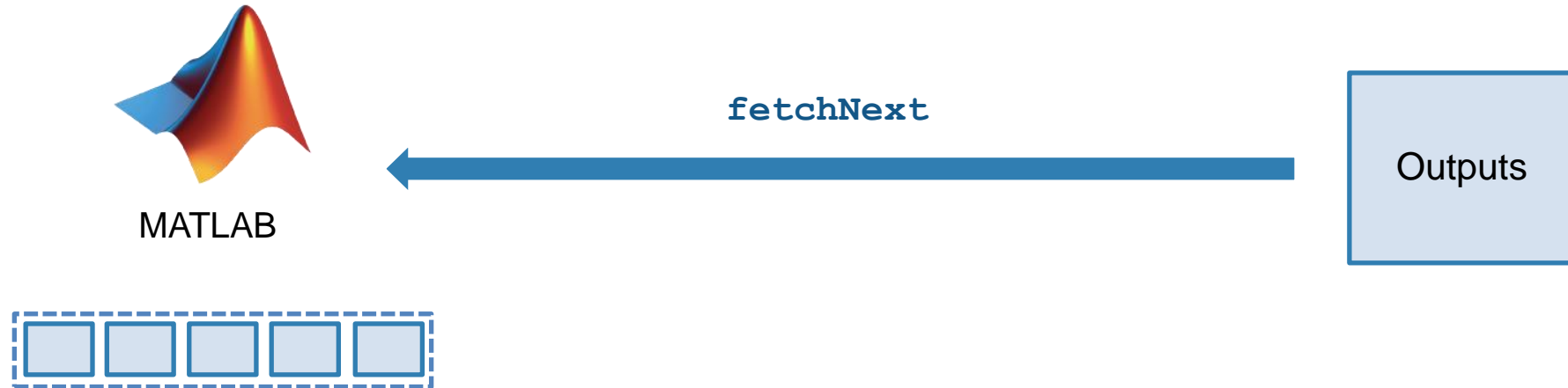


Type	Category
sliced input	input
broadcast	input
reduction	output
sliced output	output
loop	only exist on worker
temporary	only exist on worker

Use more

Keep small

Parallelism 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

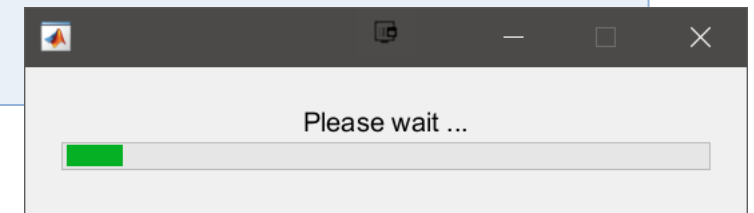
for idx = 1:10
    [completedIdx,value] = fetchNext(f);
    magicResults{completedIdx} = value;
end
```

DataQueue

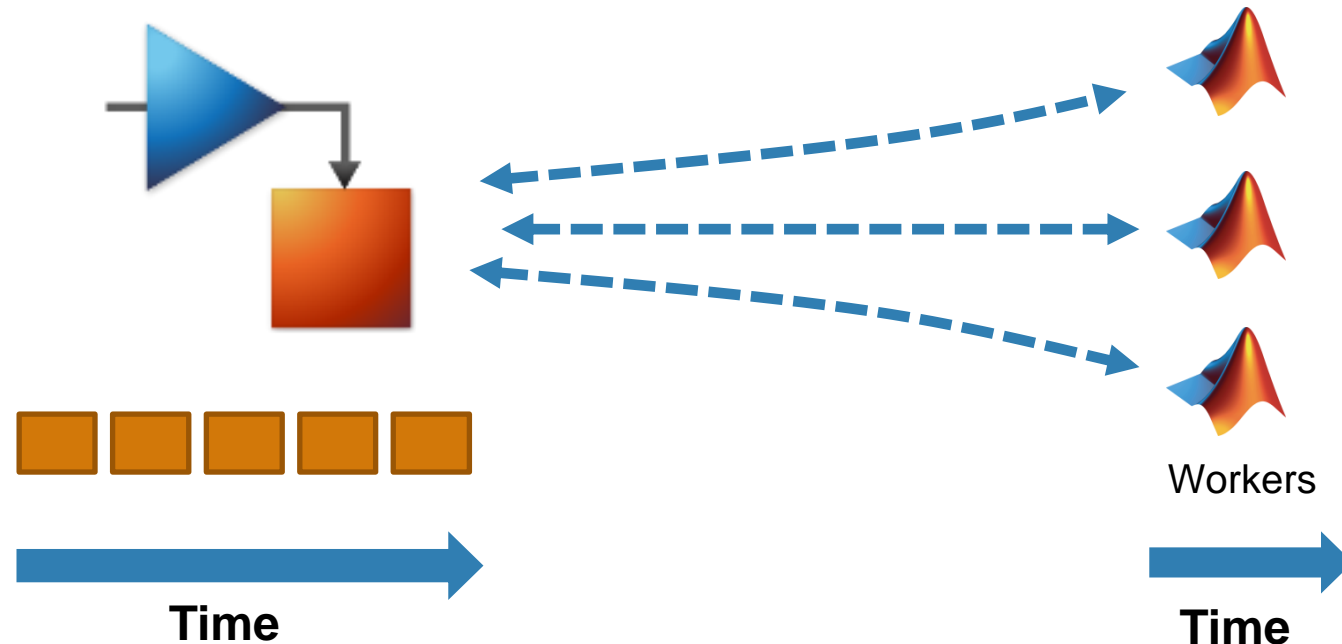
Send data or messages from parallel workers back to the MATLAB client

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
```



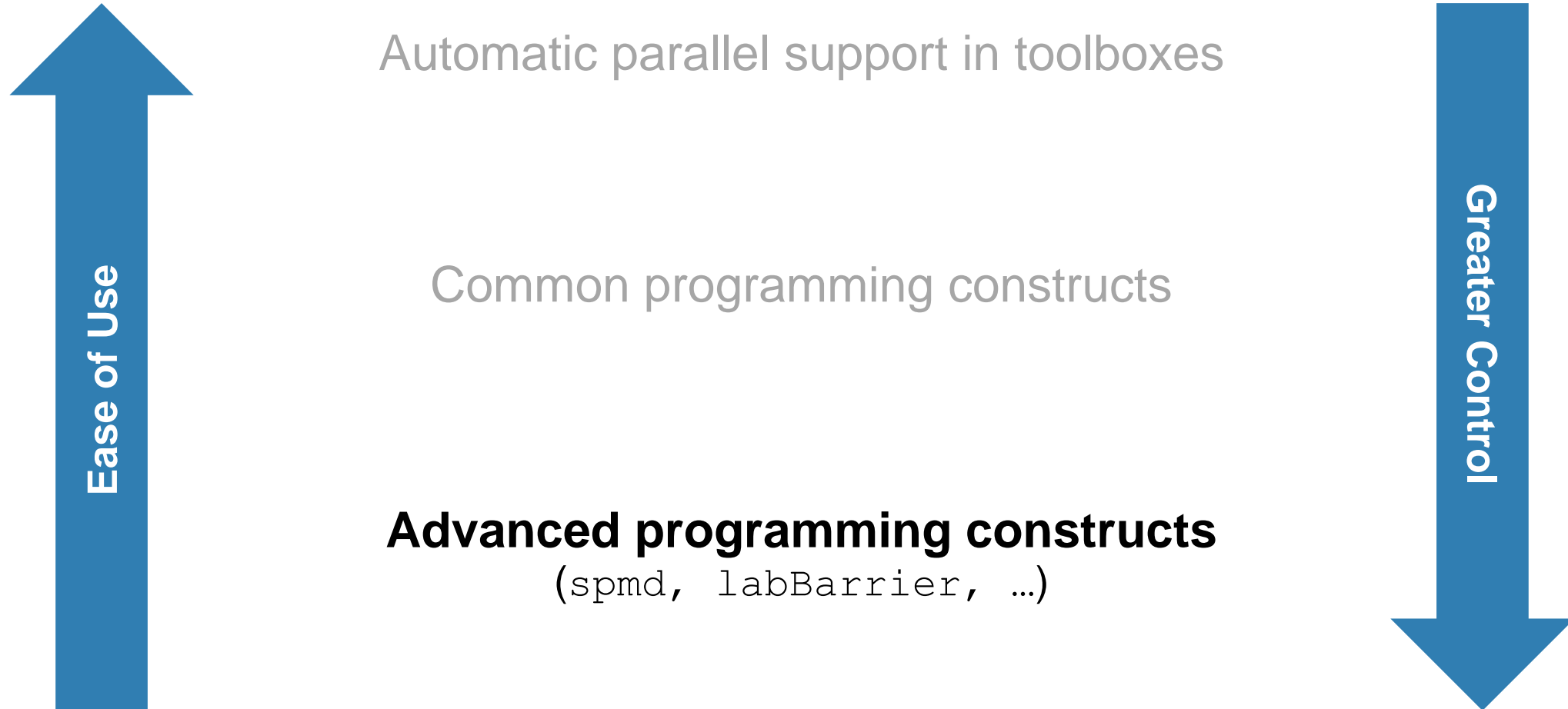
Run multiple simulations in parallel with `parsim`



- 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);
```

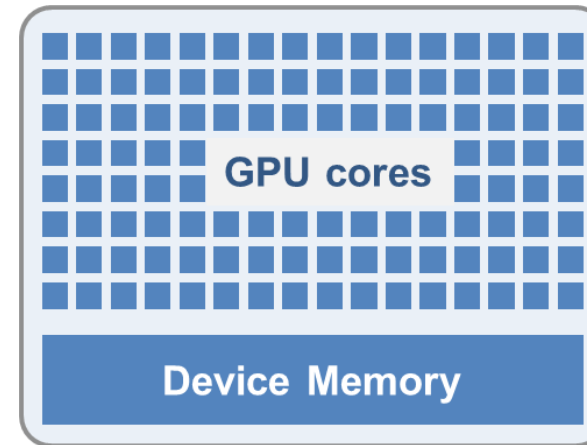
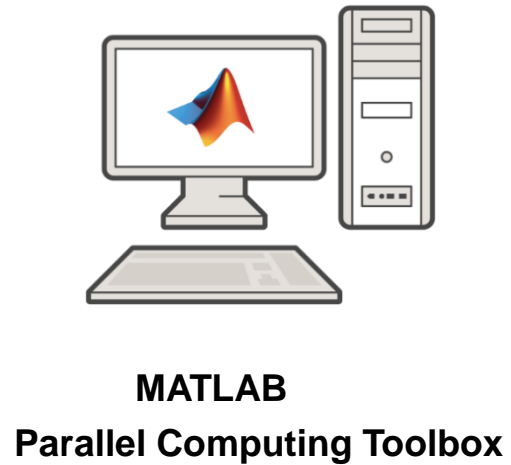
Scaling MATLAB applications and Simulink simulations



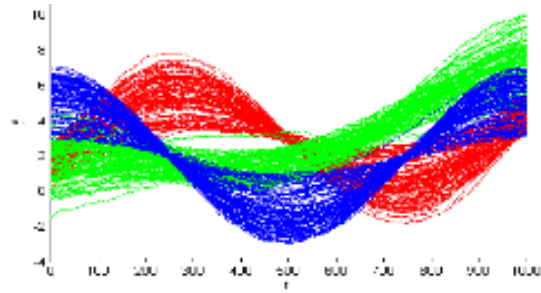
Agenda

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

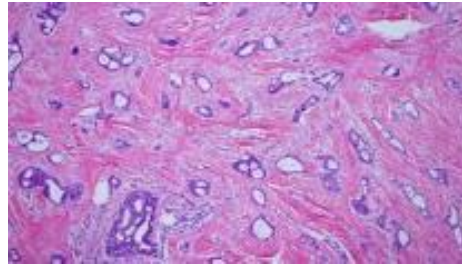
Utilizing one or multiple GPUs



Accelerating MATLAB applications with GPUs



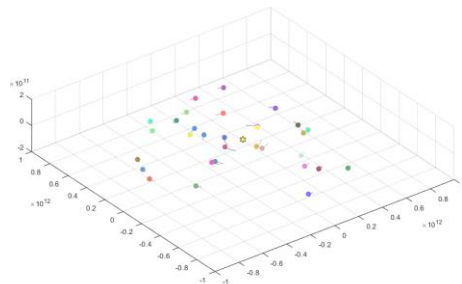
10x speedup
K-means clustering algorithm



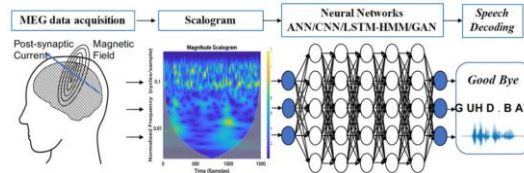
14x speedup
template matching routine



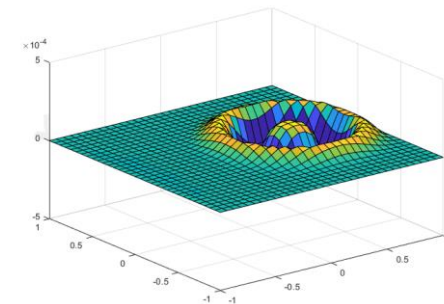
12x speedup
using Black-Scholes model



44x speedup
simulating the movement of celestial objects



10x speedup
deep learning training

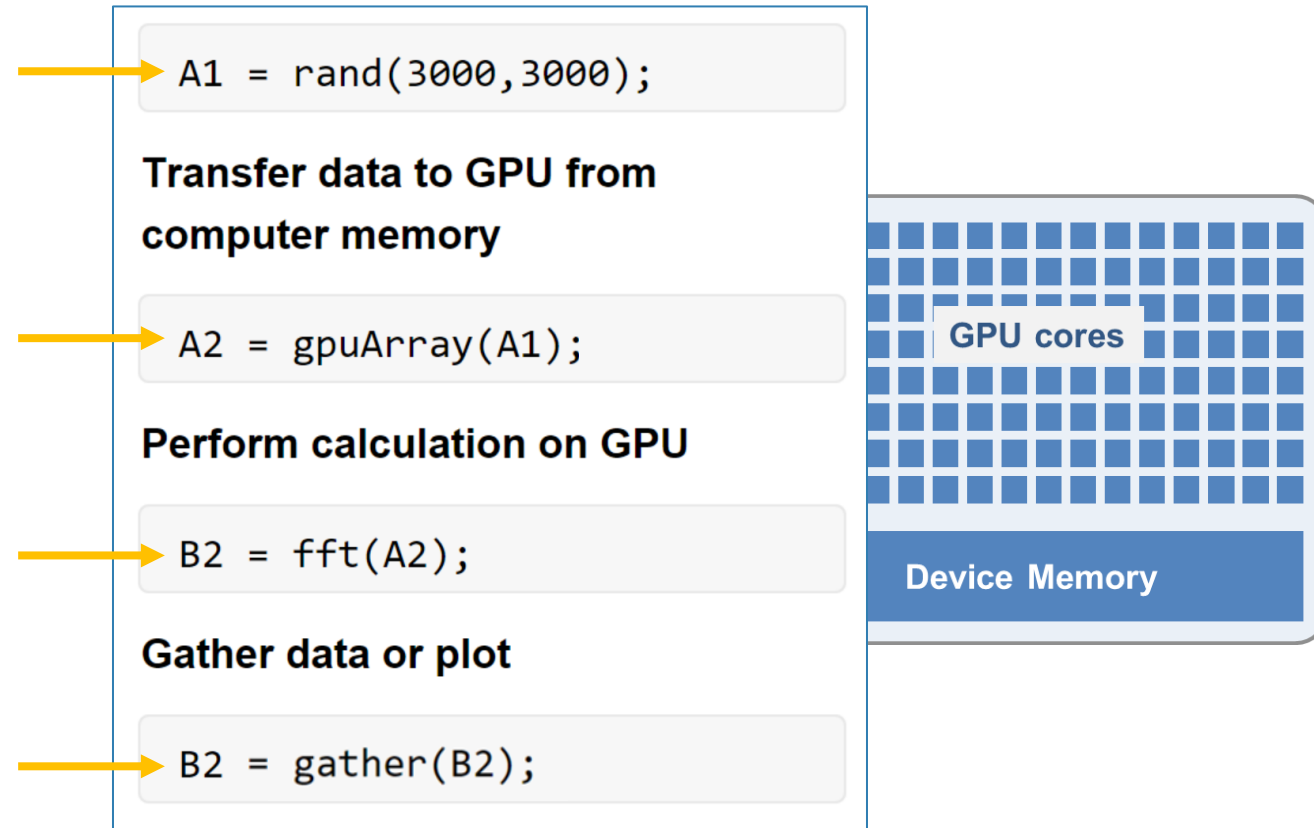


77x speedup
wave equation solving

[More info on running MATLAB functions on a GPU](#)

Speed-up using NVIDIA GPUs

- Ideal Problems
 - massively parallel and/or vectorized operations
 - computationally intensive
- Hundreds of GPU-supported functions
- Use `gpuArray` and `gather` to transfer data between CPU and GPU

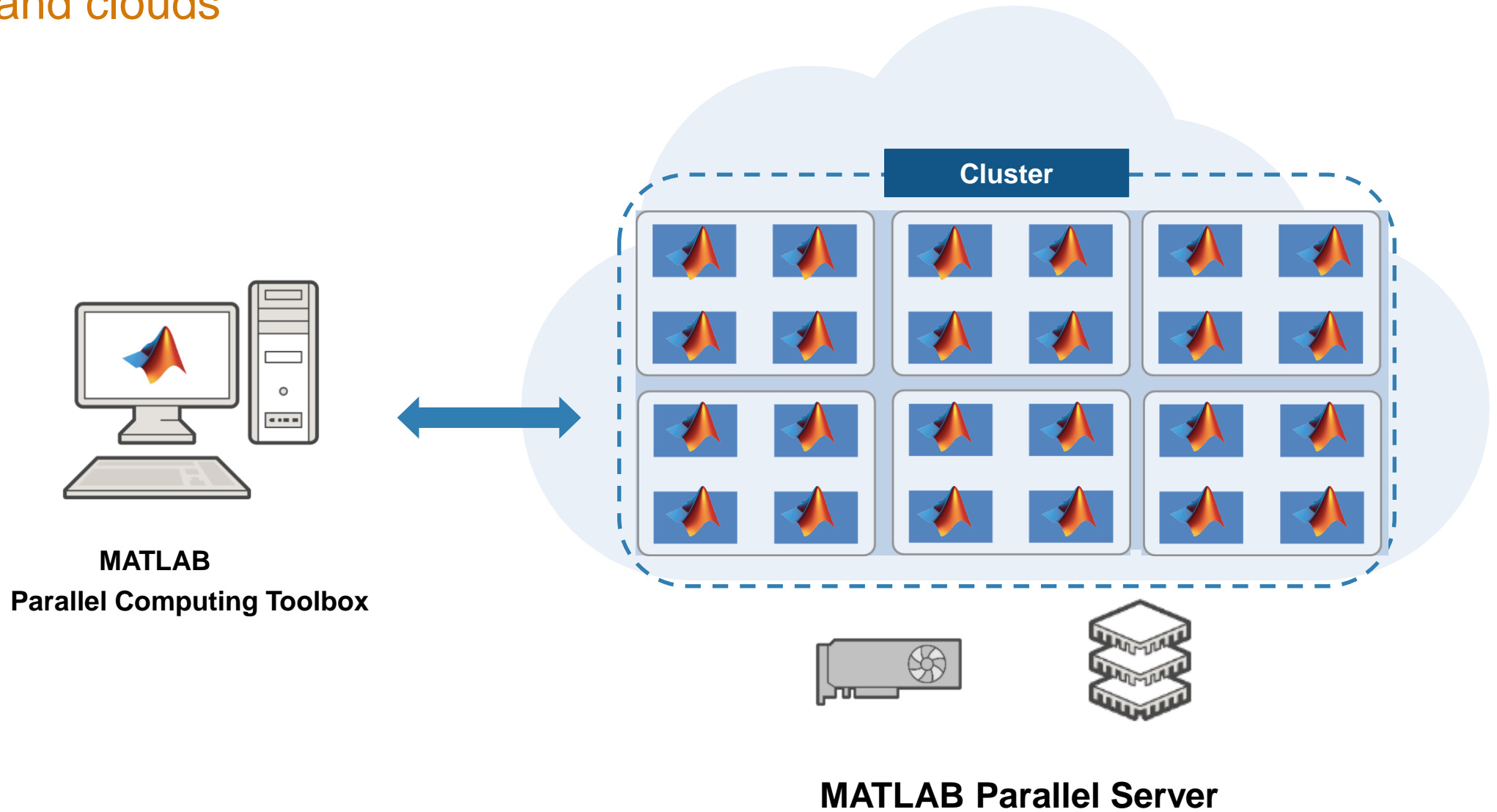


Agenda

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

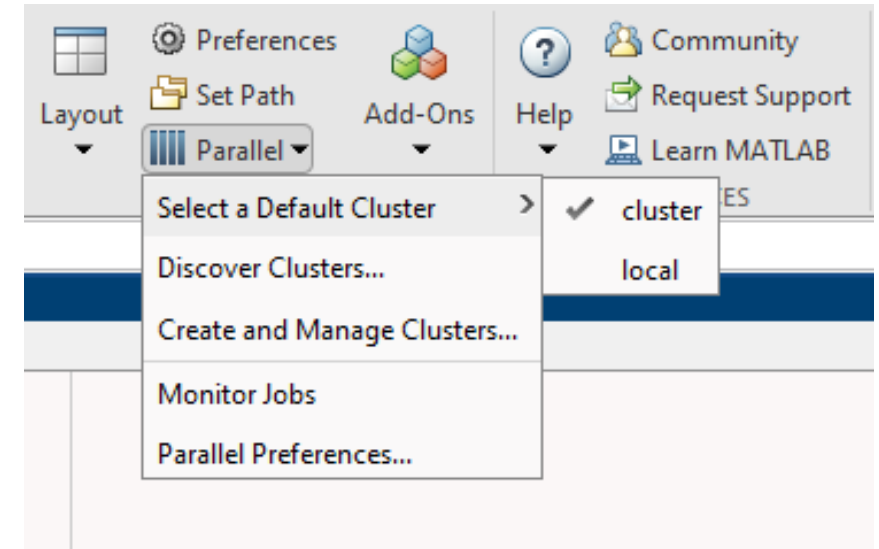
Parallel computing paradigm

Clusters and clouds



Scale to cluster and cloud

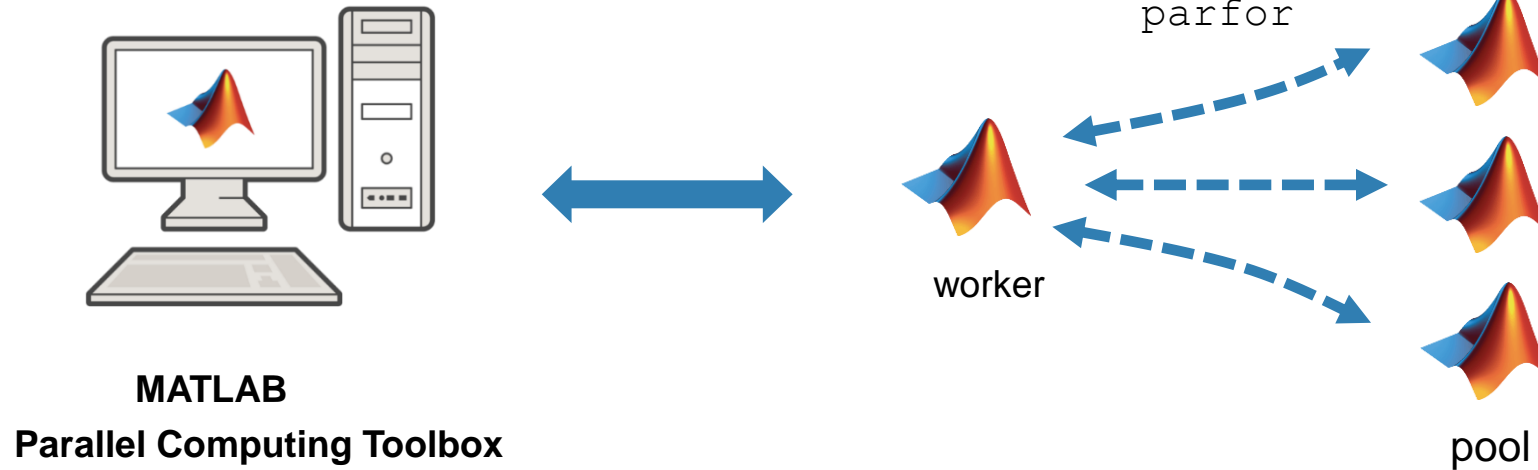
- Use MATLAB Parallel Server
- Change hardware without changing algorithm
- Cross-platform submission



batch simplifies offloading computations

Submit jobs to the cluster

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



```
>> j.State
ans =
    'running'
>> j.diary
Warning: The diary of this batch job might be incomplete
because the job is still running.
--- Start Diary ---

Analyzed 1 image.
Analyzed 2 images.
Analyzed 3 images.
Analyzed 4 images.

--- End Diary ---
```

The Job Monitor window displays the following data:

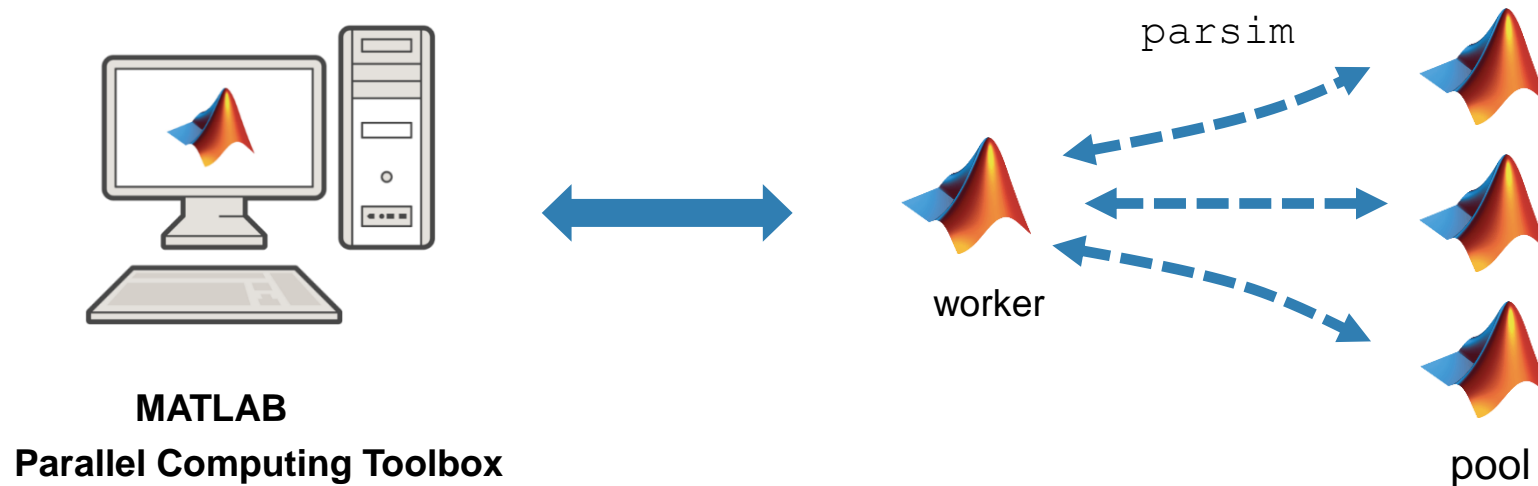
ID	Username	Submit Time	Finish Time	Tasks	State	Description
1	user200	Wed Jul 18 14:30:55 BST 2018		20	running	Independent Job
2	user200			11	pending	Independent Job

Additional interface elements include: 'Select Profile: local (default)', 'Show jobs from all users' checkbox, 'Last updated at Wed Jul 18 14:30:58 BST 2018', 'Auto update: Every 5 minutes', and an 'Update Now' button.

batch simplifies offloading computations

Submit jobs to the cluster

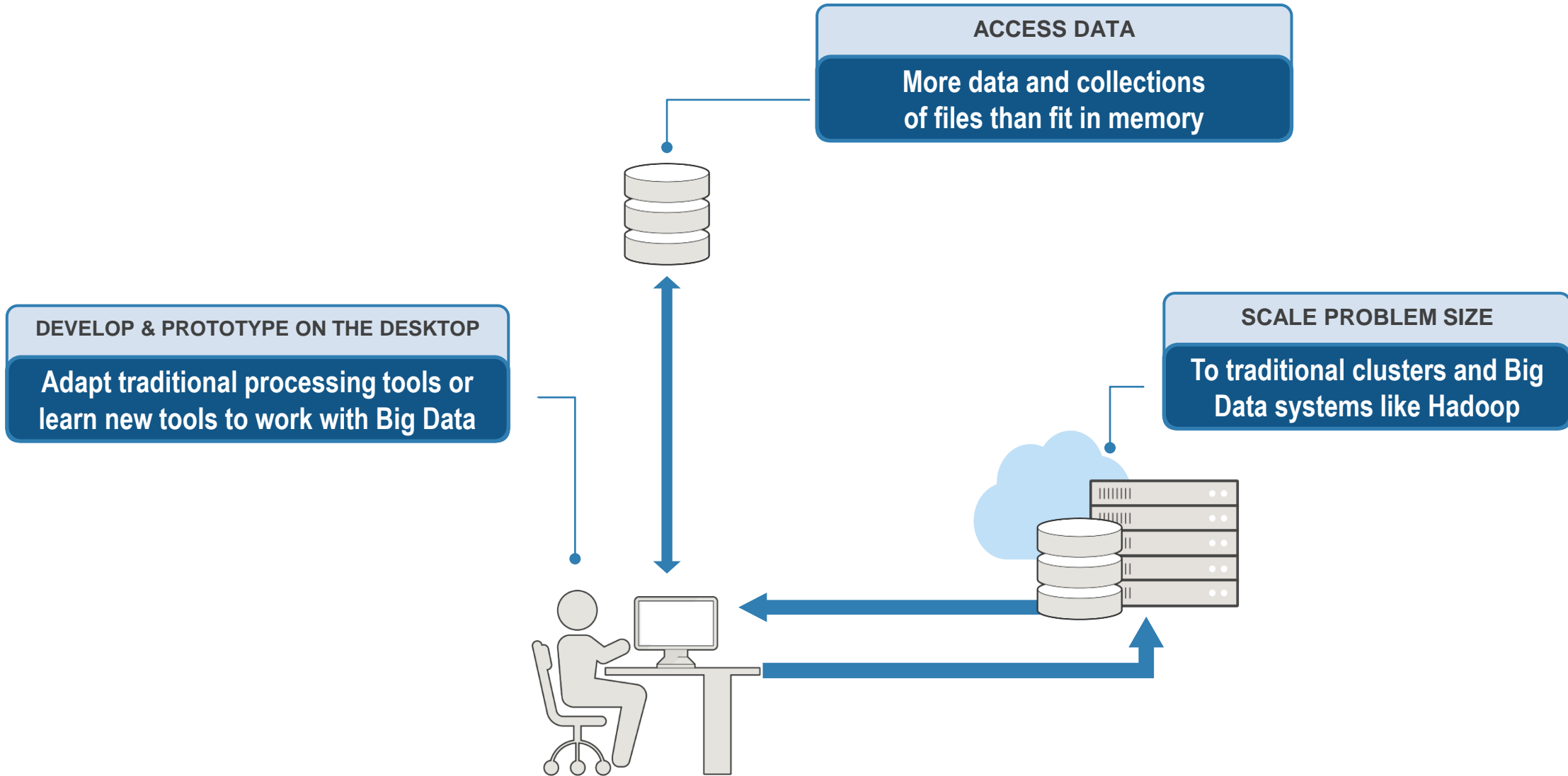
```
job = batchsim(in, 'Pool', 3);
```



Agenda

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

Big data workflows



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.)

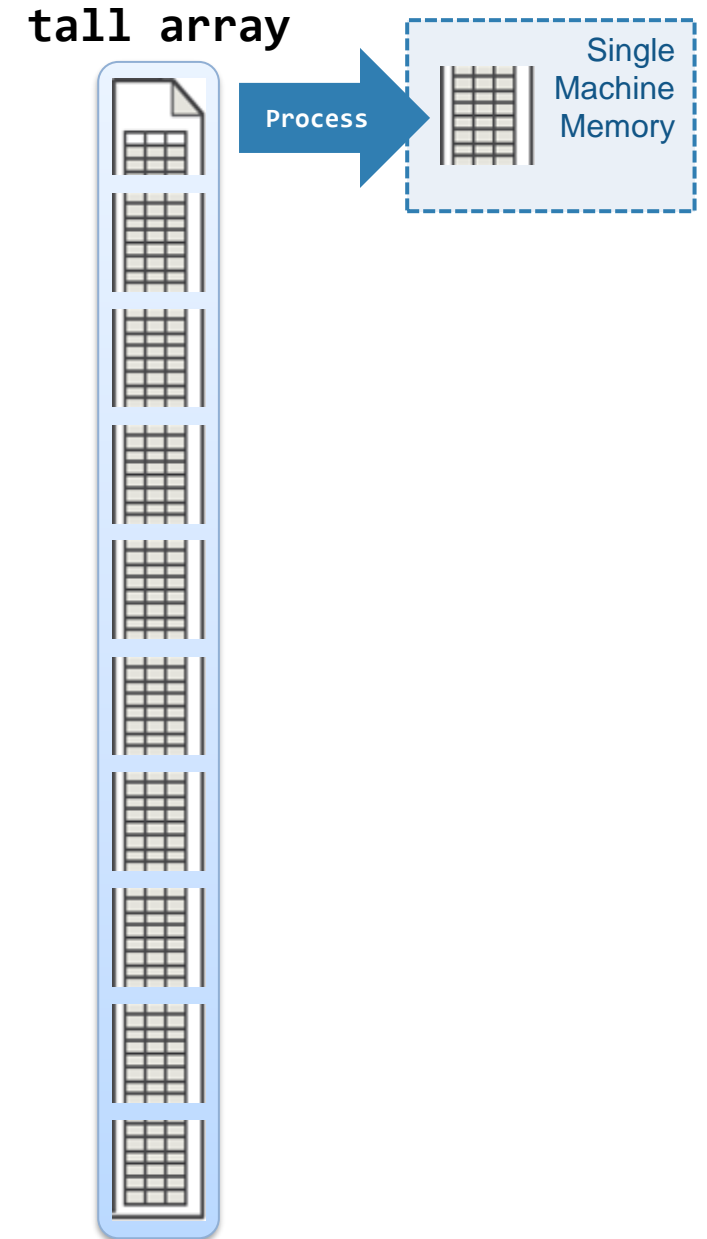
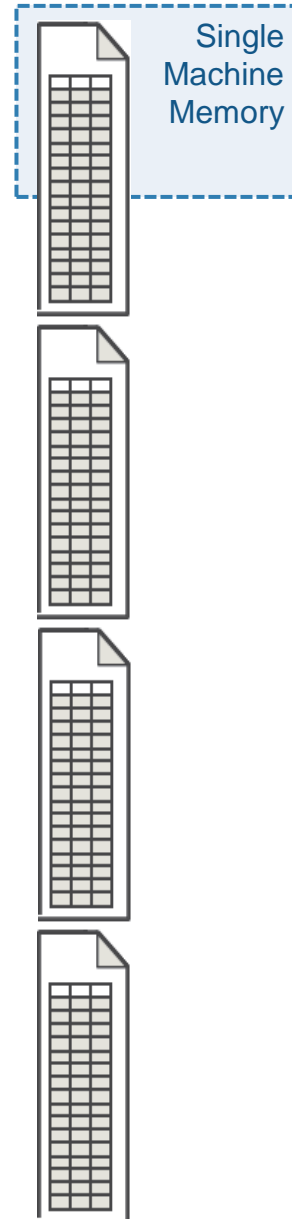


tall arrays

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

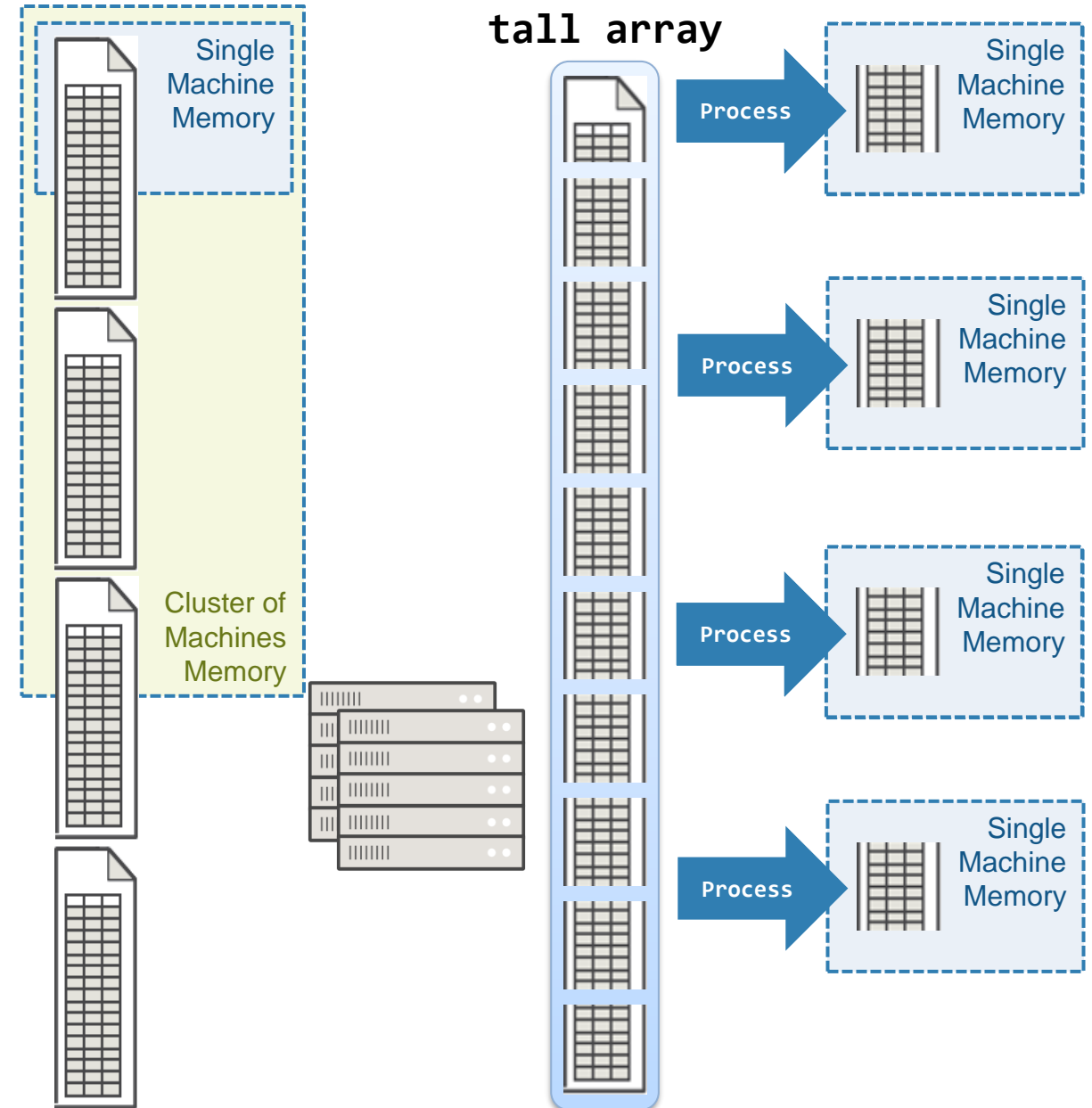
```
tt = tall(ds)
mDep = mean(tt.DepDelay, 'omitnan')
mDep = gather(mDep)
```

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



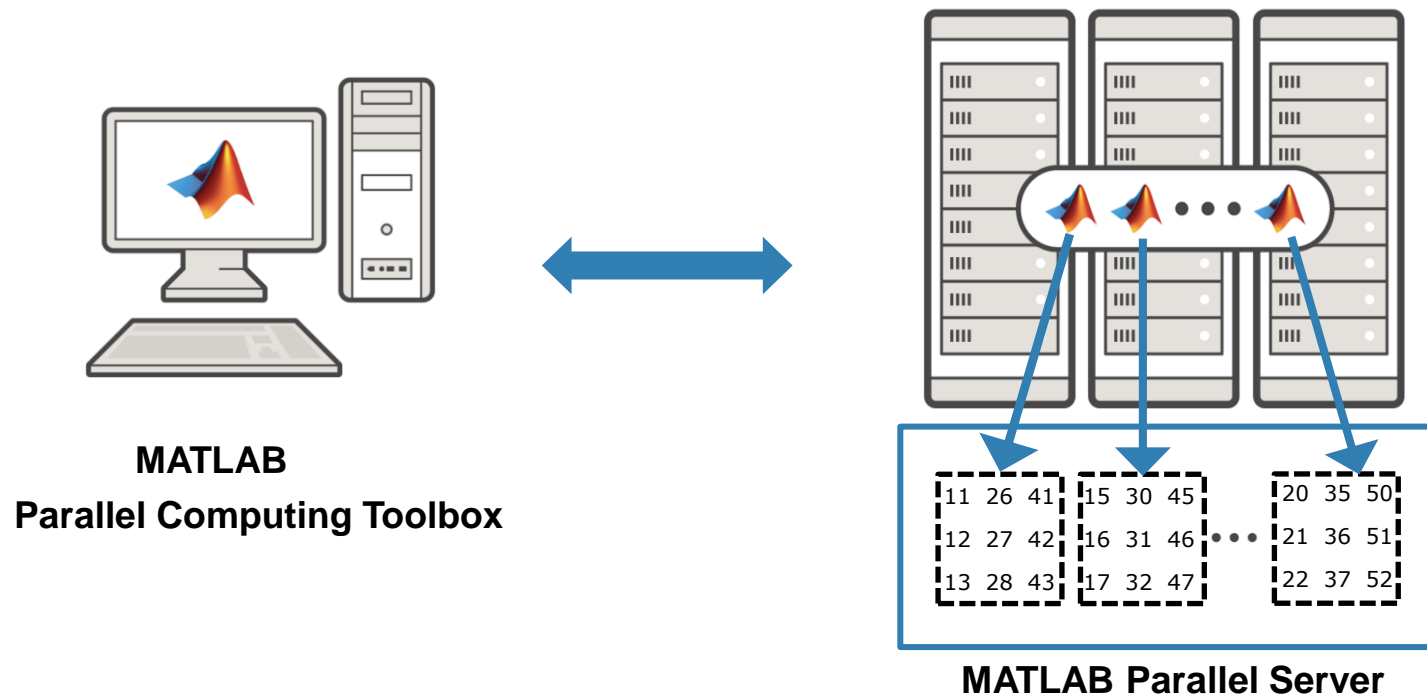
tall arrays

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



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);
```

Agenda

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

Summary

- Leverage parallel computing without needing to be parallel expert
- Speed up the execution of MATLAB applications using additional hardware
- Develop parallel applications on the desktop, easily scale to clusters as needed

Some other valuable resources

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

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

