Parallel Computing with MATLAB
Todays agenda:

Phase 1
- UNSW has 99.999% of everything we make
  - What is the **UNSW Full suite MATLAB Campus License**

Phase 2
- Parallel Computing with MATLAB
  - DEMO

Phase 3
- Q/A
Phase 1

You have a FULL suite Campus License
Fast facts about the UNSW MATLAB License:

- UNSW has a **Campus Wide License** of MATLAB
  - **ALL(99%) products**
  - **ALL staff and students**
  - **ALL devices (personal and campus)**
  - **ALL access to MATLAB Online**

- Just remember
  - You MUST create a [MathWorks account](https://www.mathworks.com/login) using your **UNSW email address** … otherwise NOTHING works

- Did you know …
  - MathWorks rolls out a NEW release every 6 months
    - “a” in March
    - “b” in September
  - You can install multiple releases onto your computers if you want to
    - eg: R2015b, R2017a, R2019b

- Is anybody using it ?
  - YTD 2020, 9000 **unique** people **activated** this License
  - + 2000 users have used **MATLAB Online**
Your Full Suite Software – part 1
MATLAB for Desktops

Individual access on:
• personal and
• university-owned machines

MATLAB Online

Access MATLAB with a web browser

MATLAB Mobile

Access MATLAB on iOS/Android devices

Anytime, Anywhere Access for Faculty, Staff, Students, and Visitors

NO software installation required
Your Full Suite Software – part 2

MATLAB Parallel Server
Perform MATLAB and Simulink computations on clusters and clouds

Scale Up Computations Run compute-intensive MATLAB applications and Simulink models on compute clusters and clouds. MATLAB Parallel Server supports batch processing, parallel applications, GPU computing, and distributed memory.

FREE
Clusters & HPC

Bring your UNSW MATLAB Parallel Server License

Clusters & HPC
Hosting provider

MathWorks Cloud Center

Bring your UNSW MATLAB Parallel Server License

You PAY Amazon for compute time

Hosting provider
UNSW has a FULL SUITE Campus Wide License of MATLAB

So what does that mean?

- ALL products (approx. 90)
- ALL staff
- ALL students
- ALL campus computers
- ALL personal computers
- ALL access to MATLAB Online

Phase 2
Parallel Computing with MATLAB
Is your MATLAB code execution slow?  
Are your Simulink models taking forever to run?  
Do you need results from millions of computations?

NEVER FEAR, HELP IS HERE!
Agenda

- Accelerating serial MATLAB code and Simulink models
- Introduction to Parallel Computing with MATLAB
- Speeding up computation with the Parallel Computing Toolbox (PCT)
- Using GPUs with MATLAB
- Scaling up to a Cluster/AWS using MATLAB Parallel Server (MPS)
- Overview of Big Data Capabilities in MATLAB (optional)
- Overview of Docker Containers for GPUs (optional)
1. How can I speed up my Serial MATLAB Code?

- Use the latest version!
  - MATLAB code now runs nearly twice as fast as it did four years ago

- Use built-in functions and data-types
  - These are extensively documented and tested with each other; constantly updated.
  - Functions such as `fft`, `eig`, `svd`, and `sort` are multithreaded by default since 2008. MATLAB can use multiple CPU cores for these without any additional effort.
Try using functions instead of scripts. Functions are generally faster.

Instead of resizing arrays dynamically, **pre-allocate** memory.

Create a new variable rather than assigning data of a different type to an existing variable.

**Vectorize** — Use matrix and vector operations instead of for-loops.

Avoid printing too much data on the screen, reuse existing graphics handles.

Avoid programmatic use of `cd`, `addpath`, and `rmpath` when possible.
Example Better Coding Practices

```
% doubling an array
a = [1 2 3; 4 5 6; 7 8 9]
for i = 1:9
    b(i) = a(i)*2
end

% getting number of spaces in a string
a = ['A big brown dog jumped over the lazy fox']
count = 0;
for i =1:length(a)
    if a(i)==' '
        count = count+1;
    end
end
count
```

```
a = [1:3;4:6;7:9]
b = 2*a
s = ['A big brown dog jumped over the lazy fox']
sum(isspace(s))
```

~3x faster!
Also more compact and readable.

Key takeaways:

>> Better programming habits lead to faster code
>> Use vectorised operations instead of loops
>> Use the built-in functions
What else can I do?

- Use ‘tic’ & ‘toc’ to time your code executions
- Use [MATLAB Profiler](https://www.mathworks.com/products/matlab-profiler.html) to analyse the execution time and find bottlenecks.
- Load common variables from a file instead of executing code to generate them repeatedly.
- For advanced users: Generate ‘mex’ (MATLAB Executable) C/C++ or CUDA code from a function.
  - Or use the MATLAB Coder or GPU Coder Apps to generate code more easily
  - Lots of supported functions
  - Massive speed-up for certain applications (sometimes up to 5x)
How to speed up Simulink?

- Try using **Accelerator** mode. This compiles certain parts of the model to C-code.
  - No limitations on type of model.

  **The speedup**
  - JIT compiles (or generates C-code for) portions of the model
  - Running compiled code has less overhead

  **The tradeoff**
  - There is overhead to generate code
  - Some run time diagnostics are disabled, e.g., inf/nan checking

- For long runs, try **Rapid Accelerator** mode.
  - Good to try for long simulations, such as batch or Monte Carlo simulations!

  **The speedup**
  - The Rapid Accelerator mode creates and runs a standalone executable from the model
  - If possible, this executable runs on a separate core than the MATLAB session

  **The tradeoff**
  - Debugging capabilities are disabled, except for scopes and viewers
  - Entire model needs to support code generation
  - It takes time to build the Rapid Acceleration target
Simulink – Comparison of Methods

- JIT accelerator is faster than normal mode in many cases unless your simulations are short.

- Rapid-accelerator has the least per-step overhead but the most initialization overhead.

- Use Fast Restart between multiple runs if model doesn’t need to be changed.

- **Additional Tip:** Try using Referenced Subsystems instead of multiple different subsystems of the same kind:
  - Less compilation overhead
  - Beneficial for Accelerator Modes
Now for something different:

- So far we’ve mostly talked about using only one core of your computer
  - But your CPU probably has many cores (2-16+), which you can utilise.
  - You may also have access to a GPU, which has hundreds of cores,
  - Or a powerful workstation or HPC Cluster or an AWS EC2 instance with multiple cores.
- Now we’ll look at how to utilise these.
- You will need the Parallel Computing Toolbox for your local machine or MATLAB Parallel Server for remote clusters/cloud computing
What is Parallel Computing?

Serial

Code executes in sequence

Parallel

Code executes in parallel
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*
Why Parallel Computing in MATLAB?

- Save time and tackle increasingly complex problems
  - Reduce computation time by using more processing power
  - Significant speed-up for certain types of problems

- Why parallel computing with MATLAB and Simulink?
  - Accelerate computation with minimal to no changes in your original code
  - Scale familiar MATLAB syntax to clusters and clouds
  - Specialized data structures and functions for Big Data applications
  - Focus on your engineering and research, not the computation
What types of problems can Parallel Computing be used for?

▪ “Embarrassingly Parallel” problems can be easily broken down into lots of simpler problems that can be solved in Parallel

▪ Term originally coined by Cleve Moler, who created the first version of MATLAB

Some Examples:

▪ Mesh-based solutions for Partial Differential Equations (PDEs)

▪ Independent Simulations with different parameters

▪ Discrete Fourier Transforms, with each harmonic calculated independently
Parameter Sweep for a Van der Pol Oscillator (a common ODE): Speeding up the same code in three different environments
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
When to use Parallel Computing?
Some questions to consider:

- Do you need to solve larger problems faster?
- Have you already optimized your serial code?
- Can your problem be solved in parallel?

- If so, do you have access to:
  - A multi-core or multi-processor computer?
  - A graphics processing unit (GPU)?
  - Access to a Cluster or AWS?
A couple of user stories…

**NASA Langley Research Center**
Accelerates Acoustic Data Analysis with GPU Computing

“Our legacy code took up to 40 minutes to analyze a single wind tunnel test; by using MATLAB and a GPU, computation time is now 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, NASA

**RTI International and University of Pennsylvania**
Model the Spread of Epidemics Using MATLAB and Parallel Computing

“Using Parallel Computing Toolbox we added four lines of code and wrote some simple task management scripts. Simulations that took months now run in a few days. MathWorks parallel computing tools enabled us to capitalize on the computing power of large clusters without a tremendous learning curve.”
- Diglio Simoni, RTI
Most of your MATLAB code runs on one core

(Though many linear algebra and numerical functions such as \texttt{fft}, \texttt{eig}, \texttt{svd}, and \texttt{sort} are multithreaded by default since 2008)
The Parallel Computing Toolbox (PCT) can help you by using multiple CPU cores on your local machine.
PCT requires only simple modifications to your code

Three good commands to know:

- `for` → `parfor` (parallel for-loop)
- `feval` → `parfeval` (parallel function evaluations)
- `sim` → `parsim` (parallel Simulink runs)
Explicit parallelism with `parfor`

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

Learn more about `parfor`
Explicit parallelism with `parfor`

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

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

```matlab
a = zeros(10, 1);
b = pi;
parfor i = 1:10
    a(i) = i + b;
end
```
Hands-On Exercise: Introduction to parfor
Factors that govern speedup of `parfor` loops

- May not be much speedup when computation time is too short

- Execution may be slow because of:
  - Memory limitations (RAM)
  - File access limitations

- Implicit multithreading
  - MATLAB uses multiple threads for speedup of some operations
  - Use Resource Monitor or similar on serial code to check on that

- Unbalanced load due to iteration execution times
  - Avoid some iterations taking multiples of the execution time of other iterations
Parallelize Simulink Model Execution with `parsim` 
Example: Parameter Sweep of ODEs

- Parameter sweep of ODE system
  - Damped spring oscillator in Simulink
  - Sweep through different values of damping and stiffness
  - Record peak value for each simulation

- Convert `sim` to `parsim`

- Use pool of MATLAB workers

\[
\ddot{x} + \frac{b}{m} \dot{x} + \frac{k}{m} x = 0
\]

\[
\begin{align*}
5
\end{align*}
\]

\[
\begin{align*}
\frac{b}{m},... & \frac{k}{m},... \\
1,2,... & 1,2,...
\end{align*}
\]
Run independent Simulink simulations in parallel using the `parsim` function

```matlab
for i = 10000:-1:1
    in(i) = Simulink.SimulationInput(my_model);
    in(i) = in(i).setVariable('my_var', i);
end
out = parsim(in);
```
Hands-On Exercise: Introduction to *parsim*
Using NVIDIA GPUs with the Parallel Computing Toolbox
Why GPUs

- **GPU: Graphics Processing Unit**
  - Simpler than a CPU, but has a lot more cores (commonly 2000+)

- **Ideal for:**
  - Massively parallel problems and/or vectorized operations
  - Computationally intensive applications

- **MATLAB Advantage:**
  - 500+ GPU-enabled MATLAB functions
  - Simple programming constructs: `gpuArray`, `gather`
Run Same Code on CPU and GPU
Solving 2D Wave Equation

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>1536</td>
<td>2</td>
</tr>
<tr>
<td>2048</td>
<td>3</td>
</tr>
</tbody>
</table>

CPU
Intel(R) Xeon(R) W3550 3.06GHz
4 cores
memory bandwidth 25.6 Gb/s

GPU
NVIDIA Tesla K20c
706MHz
2496 cores
memory bandwidth 208 Gb/s
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)
How do I know if I have a supported GPU?

- In MATLAB, type:

  ```
  >> gpuDevice
  ans =
  CUDA Device with properties:
  Name: 'Tesla V100-DCXS-16GB'
  Index: 1
  ComputeCapability: '7.0'
  SupportsDouble: 1
  DriverVersion: 10
  ToolkitVersion: 10
  MaxThreadsPerBlock: 1024
  MaxShmemPerBlock: 49152
  MaxThreadBlockSize: [1024 1024 64]
  MaxGridSize: [2.1475e+09 65535 65535]
  SMWidth: 32
  TotalMemory: 1.6900e+10
  AvailableMemory: 1.6130e+10
  MultiprocessorCount: 80
  ClockRateKHz: 1530080
  ComputeMode: 'Default'
  GPUOverlapsTransfers: 1
  KernelExecutionTimeout: 1
  CanMapHostMemory: 1
  DeviceSupported: 1
  DeviceSelected: 1
  ```

- If you see a CUDA Device, you are good to go.

- The key number to note is the ‘ComputeCapability’
  - This should be above 3.2 for Deep Learning applications
GPU Demo – Mandelbrot set

If you have an NVIDIA GPU

>> doc mandelbrot

→ Illustrating Three Approaches to GPU Computing: The Mandelbrot Set
Parallel computing paradigm
Clusters and clouds

- Prototype on the desktop
- Integrate with HPC infrastructure
- Access directly through MATLAB

MATLAB Parallel Server

MATLAB Parallel Computing Toolbox

GPU
Multi-core CPU
Migrate to Cluster / Cloud

- Use MATLAB Parallel Server
- Change hardware without changing algorithm:
  - Just replace local with the name of your profile
  - Via command line for parallel pools:
    >>> parpool('MyCluster', N)
  - Via default cluster
  - Via command line for batch jobs:
    >>> clust = parcluster('MyCluster');
Using Cloud Clusters using AWS EC2

- Amazon Web Services – Elastic Cloud Compute
  - Allows custom HPC clusters to be made very quickly for on-demand usage.
  - Relatively inexpensive compared to conventional HPC setups.

- Easy interface via MATLAB Cloud Center

- If an AWS account is in place, create a cluster from Cloud Center (10 minute process)

- Then import into MATLAB using Parallel → Discover Clusters
Creating a Cloud Cluster using Amazon Web Services (AWS)

1. Go to MathWorks Cloud Center: cloudcenter.mathworks.com

2. Create a Cluster

3. Name your Cluster

4. Select the Configuration

5. Start the Cluster
Parameter Sweep for a Van der Pol Oscillator (a common ODE): Speeding up the same code in three different environments
**batch** can be used to submit Jobs to a Cluster

Tasks will be automatically added to the queue of the configured Scheduler

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

This can also be used to:
- Queue up tasks for a Parallel Pool
- Offload any computation from a client machine onto a Cluster for faster processing
- Lets you close MATLAB or even shut down your computer while code runs on cluster
Get results and clean up

- When batch job has finished, you can obtain results from it
  
  ```
  results = fetchOutputs(job)
  ```

- `results` is a cell array
  - Number of elements = number of outputs returned from batch job

  - Accessing k-th output argument:
    ```
    outk = results{k}
    ```

  - Delete the job when you’re done
    ```
    delete(job)
    ```
Use Job Monitor to check status of jobs without leaving MATLAB

- Open Job Monitor from **Parallel** menu
- Select the profile you want to look at
- Shows own and (optionally) other people’s jobs
- Right-click job for more information and actions
Advantages of batch jobs over interactive parallel pools

- Interactive parallel pools:
  - MATLAB ("client") session that starts the parallel pool needs to remain open
  - Only one interactive parallel pool can run at a time

- For batch jobs
  - MATLAB can be closed on client
  - Client can be shut down
  - Batch job can include a parallel pool, and multiple batch+pool jobs can run simultaneously

- Batch jobs are particularly suitable for
  - Working on a cluster of computers
  - Long-running jobs
Using Clusters on AWS EC2

- Very easy interface via MATLAB Cloud Center
- If an AWS account is in place, create a cluster from Cloud Center (10 minute process)
- Then import into MATLAB using Parallel → Discover Clusters
Big data workflow

ACCESS DATA
More data and collections of files than fit in memory

DEVELOP & PROTOTYPE ON THE DESKTOP
Adapt traditional processing tools or learn new tools to work with Big Data

SCALE PROBLEM SIZE
To traditional clusters and Big Data systems like Hadoop
**distributed arrays**

- Keep large datasets in-memory, split among workers running on a cluster
- Common Actions: Matrix Manipulation & Linear Algebra and Signal Processing
- Several hundred MATLAB functions overloaded for distributed arrays

![MATLAB Parallel Computing Toolbox](image)

![MATLAB Parallel Server](image)
distributed arrays

Develop and prototype locally and then scale to the cluster

MATLAB Parallel Computing Toolbox

% prototype with small A, b
parpool('local')
spmd
    A = codistributed(m1);
    b = codistributed(m2);
end
x = A \ b;
xg = gather(x);

MATLAB Parallel Server

% scale with large A, b
parpool('cluster')
spmd
    A = codistributed(m1);
    b = codistributed(m2);
end
x = A \ b;
xg = gather(x);
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
- 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
Big Data Without Big Changes

One file

Access Data

```matlab
measured = readtable('PumpData.csv');
measured = table2timetable(measured);
```

Preprocess Data

Select data of interest

```matlab
measured = measured(timerange(seconds(1),seconds(2)), 'Speed');
```

Work with missing data

```matlab
measured = fillmissing(measured, 'linear');
```

Calculate statistics

```matlab
m = mean(measured.Speed);
s = std(measured.Speed);
```

One hundred files

Access Data

```matlab
measured = datastore('PumpData*.csv');
measured = tall(measured);
measured = table2timetable(measured);
```

Preprocess Data

Select data of interest

```matlab
measured = measured(timerange(seconds(1),seconds(2)), 'Speed');
```

Work with missing data

```matlab
measured = fillmissing(measured, 'linear');
```

Calculate statistics

```matlab
m = mean(measured.Speed);
s = std(measured.Speed);
```

```matlab
[m,s] = gather(m,s);
```
Big Data Capabilities in MATLAB with Parallel Computing

Distributed Arrays

Datastores

Apache Spark™ on Hadoop

Tall Arrays

1. Distributed Arrays
2. Datastores
3. Apache Spark™ on Hadoop
4. Tall Arrays

```
Tt = tall(ds)
fitlm(tTTrain)
```
## Datatypes for Scaling

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Memory Location</th>
<th>Use case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tall</strong></td>
<td>Disks</td>
<td>Pre-processing, statistics, machine learning</td>
</tr>
<tr>
<td><strong>distributed</strong></td>
<td>Cluster</td>
<td>Sparse and dense numerics</td>
</tr>
<tr>
<td><strong>gpuArray</strong></td>
<td>GPU</td>
<td>GPU computations</td>
</tr>
</tbody>
</table>
Summary – Working with Big Data

- Use **datastores** to manage data processing from large collections of files.
- Use **Tall Arrays** to process files too big to fit in memory.
- Use **Distributed Arrays** and **GPU Arrays** to parallelize problems for solving on multiple workers at once.
- Use **Parallel Computing Toolbox** (on Desktop) or **MATLAB Parallel Server** (on clusters) to scale-up solutions.
Summary of Big Data capabilities in MATLAB

1. ACCESS DATA
   - More data and collections of files than fit in memory
   - Datastores
     - Images
     - Spreadsheets
     - Tabular Text
     - Custom Files
     - SQL
     - Hadoop (HDFS)
     - Tabular Text
     - Custom Files
     - SQL
     - Hadoop (HDFS)

2. PROCESS ON THE DESKTOP
   - Adapt traditional processing tools or learn new tools to work with Big Data
   - Tall Arrays
     - Math
     - Statistics
     - Visualization
     - Machine Learning
   - MapReduce

3. SCALE PROBLEM SIZE
   - To traditional clusters and Big Data systems like Hadoop
   - Tall Arrays
     - Math, Stats, Machine Learning on Spark
   - Distributed Arrays
     - Matrix Math on Compute Clusters
   - SPMD
   - MapReduce
   - MATLAB API for Spark
Summary

- Use **Parallel Computing Toolbox** on the Desktop to speed up your computationally intensive applications using multiple CPU cores or GPUs.
- Scale up to Clusters or Cloud using **MATLAB Parallel Server**
- Use Big Data capabilities such as Tall and Distributed Arrays, Datastores to further scale up solutions.

Dhruv Chandel, PhD
*Education Technical Evangelist, MathWorks*
dchandel@mathworks.com