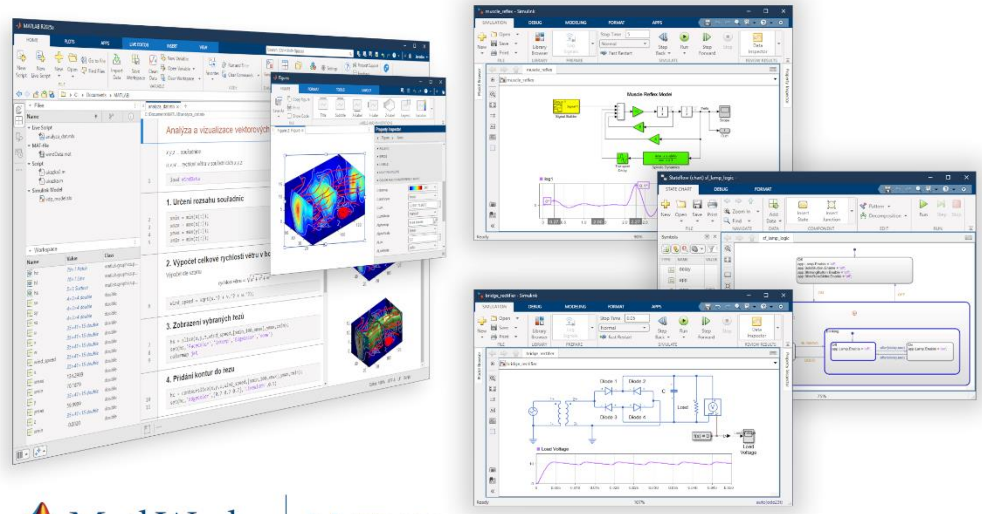
Škálovanie algoritmov a simulácií pomocou paralelných výpočtov



Michal Blaho

blaho@humusoft.sk

www.humusoft.cz
info@humusoft.cz

www.mathworks.com

[Cornell Bioacoustics Scientists Develop a High-Performance Computing Platform for Analyzing Big Data]

“High-performance computing with MATLAB enables us to process previously unanalyzed big data. We translate what we learn into an understanding of how human activities affect the health of ecosystems to inform responsible decisions about what humans do in the ocean and on land.”

[Research Engineers Advance Control System Design of the International Linear Collider]

"Using Parallel Computing Toolbox, we deployed our Simulink model on a large group cluster for distributed execution. We could simultaneously run simulations providing coverage for hundreds of scenarios. As a result, we achieved a linear speedup in the turnaround time for this task. MathWorks tools have enabled us to accomplish work that was once impossible."

Before going parallel, optimize your code

- Use the **Profiler** to find the code that runs slowest and determine possible performance improvements

```
rng(1)
x = rand(1,1e6);
for k = 1:numel(x)
    if x(k)<.5
        x(k) = 0;
    end
end
```

Time	Calls	Line	
0.035	1	1	rng(1)
0.012	1	2	x = rand(1,1e6);
< 0.001	1	3	for k = 1:numel(x)
0.061	1000000	4	if x(k)<.5
0.026	499837	5	x(k) = 0;
0.048	1000000	6	end
0.049	1000000	7	end

Use vectorization (matrix and vector operations) instead of for-loops

Time	Calls	Line	
0.007	1	1	rng(1)
0.011	1	2	x = rand(1,1e6);
0.007	1	3	x(x<.5) = 0;

HUMUSOFT

Before going parallel, optimize your code

- 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
```

Preallocate the maximum amount of space required for the array instead of letting MATLAB repeatedly reallocate memory for the growing array

Elapsed time is 0.013109 seconds.

Optimize your code with efficient programming practices

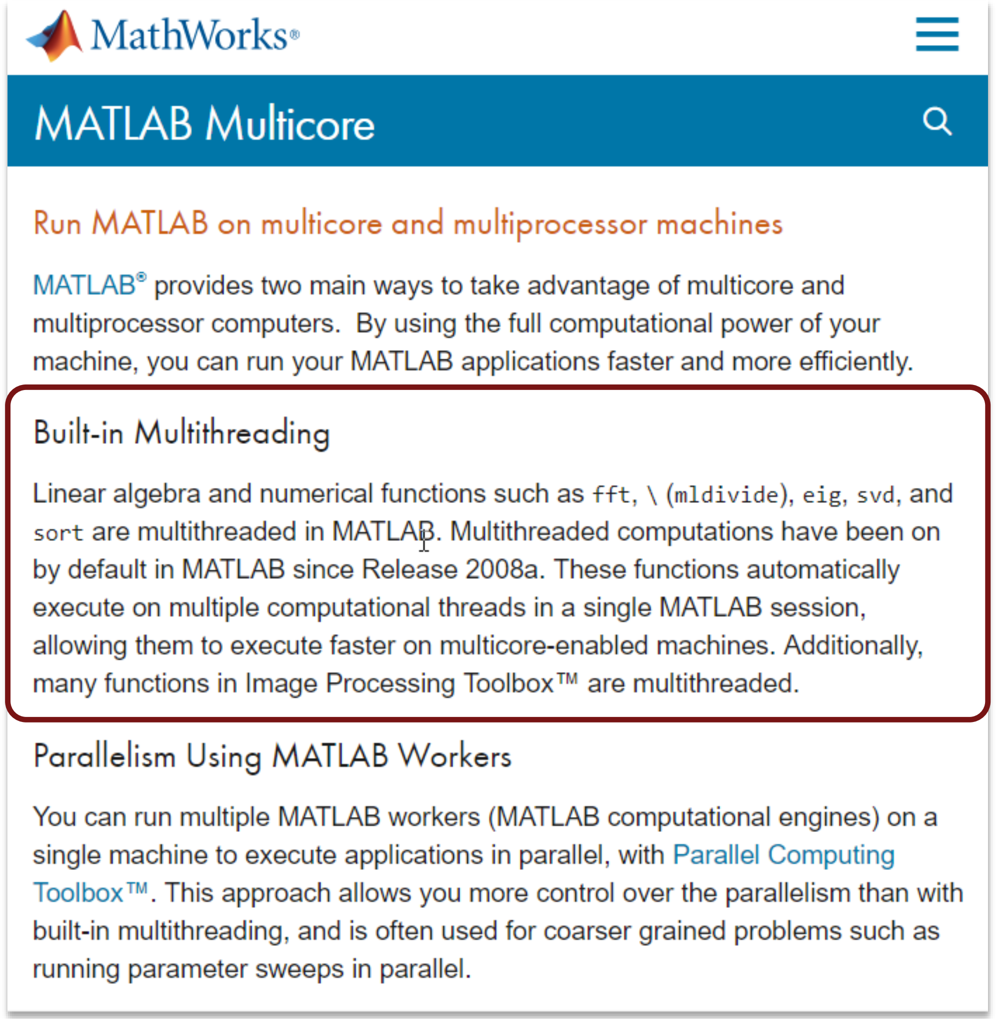
- **Pre-allocate** memory instead of letting arrays be resized dynamically
- **Vectorize** – Use matrix and vector operations instead of for-loops
- Try using functions instead of scripts. Functions are generally faster
- Create a new variable rather than assigning data of a different type to an existing variable
- Place independent operations outside loops to avoid redundant computations
- Avoid printing too much data on the screen, reuse existing graphics handles

Techniques to improve performance

MATLAB has built-in multithreading

MATLAB

Multi-core CPU



MathWorks®

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

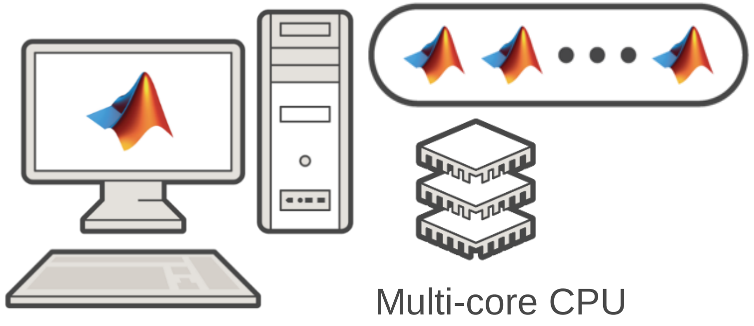
Linear algebra and numerical functions such as `fft`, `\` (`mldivide`), `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session, 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.

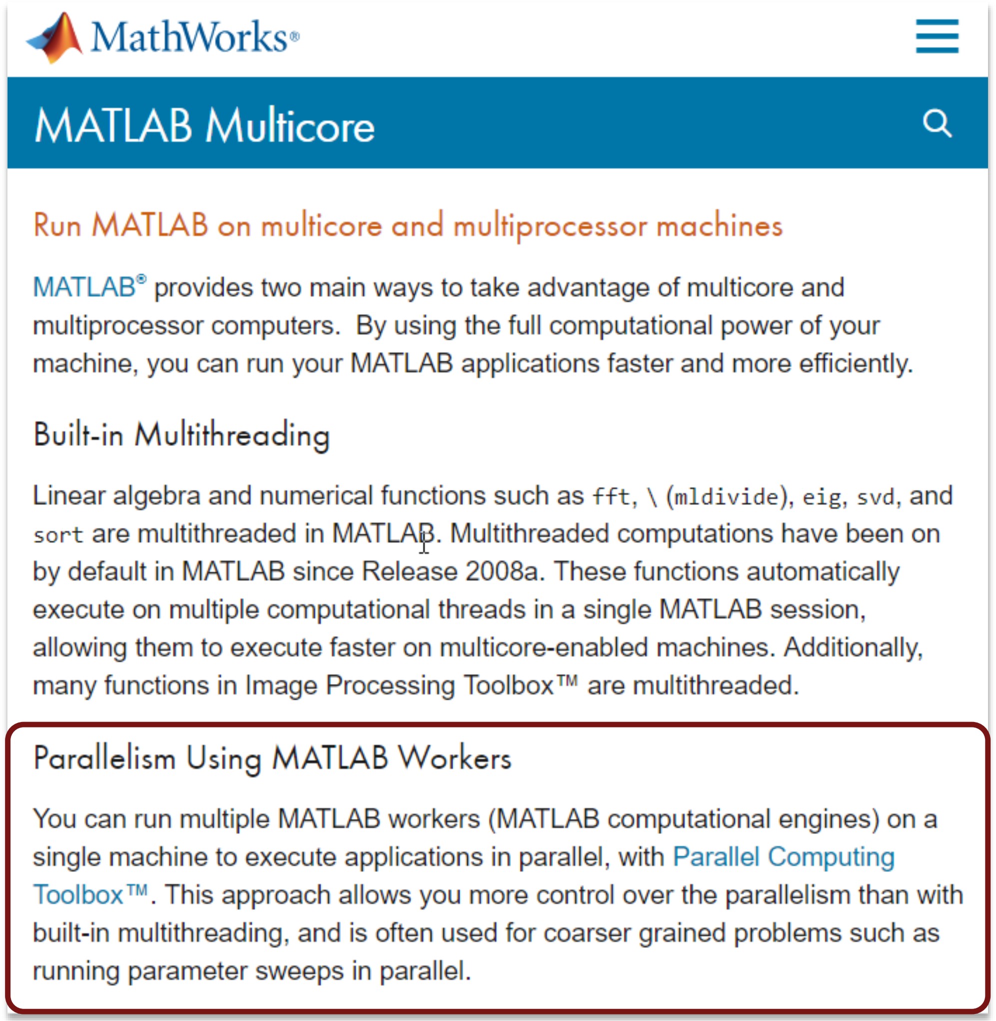
MATLAB multicore

Scale further with parallel computing



MATLAB

Parallel Computing Toolbox


MathWorks®

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

HUMUSOFT®

Automatic parallel support (MATLAB)

Property Name | Property Value

... "UseParallel",true ...

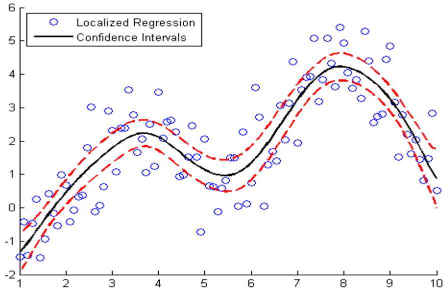
Options

Image Processing

Original Image of Peppers

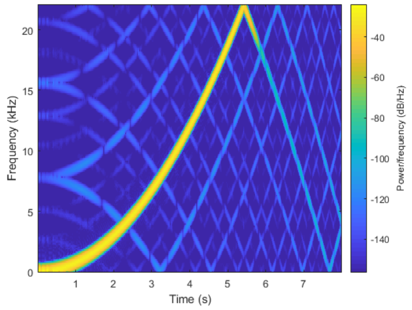
Recolored Image of Peppers

Statistics and Machine Learning

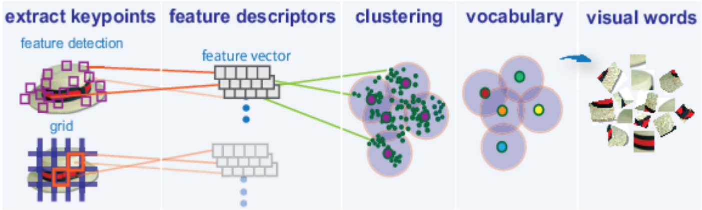


Deep Learning

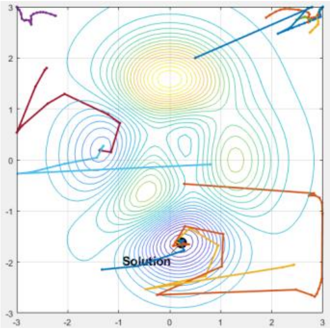
Signal Processing and Communications



Computer Vision

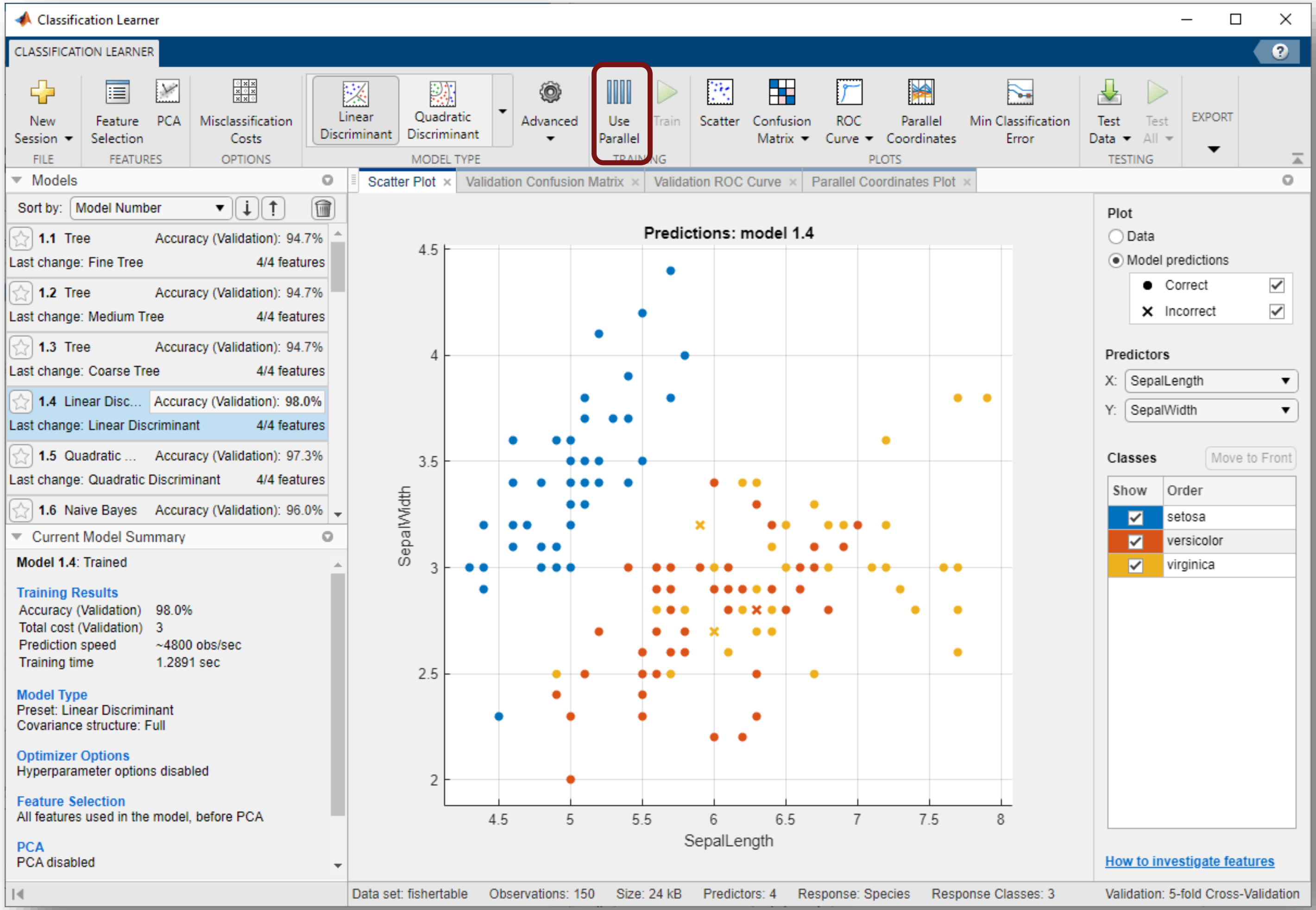


Optimization and Global Optimization

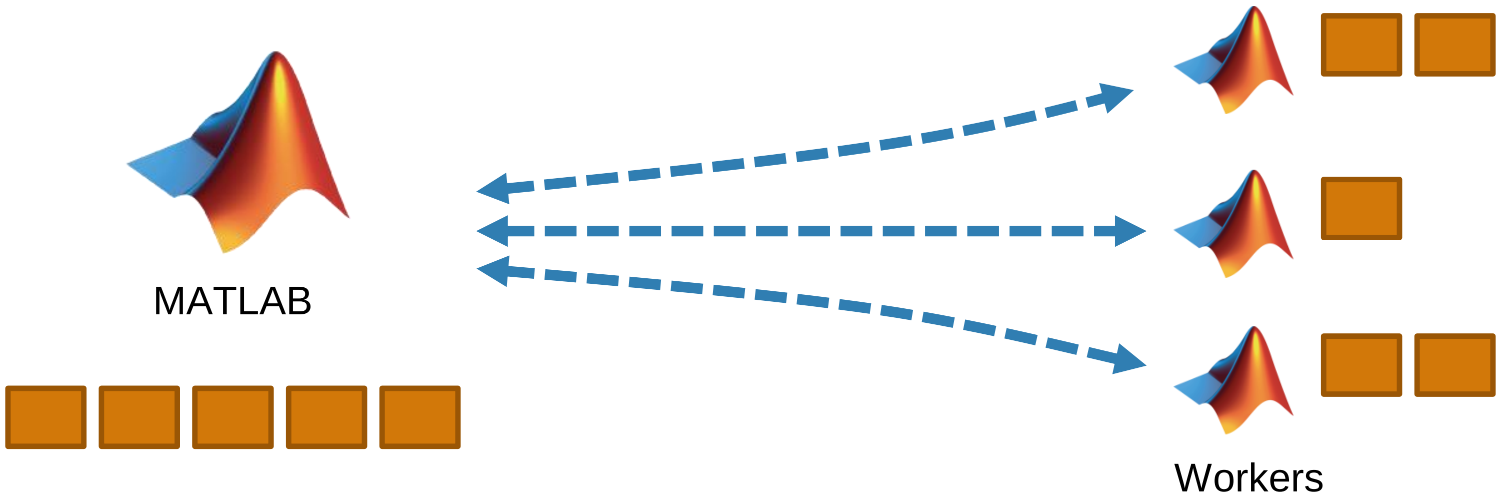


[Additional automatic parallel support]

Run multiple classifiers at once with automatic parallel support



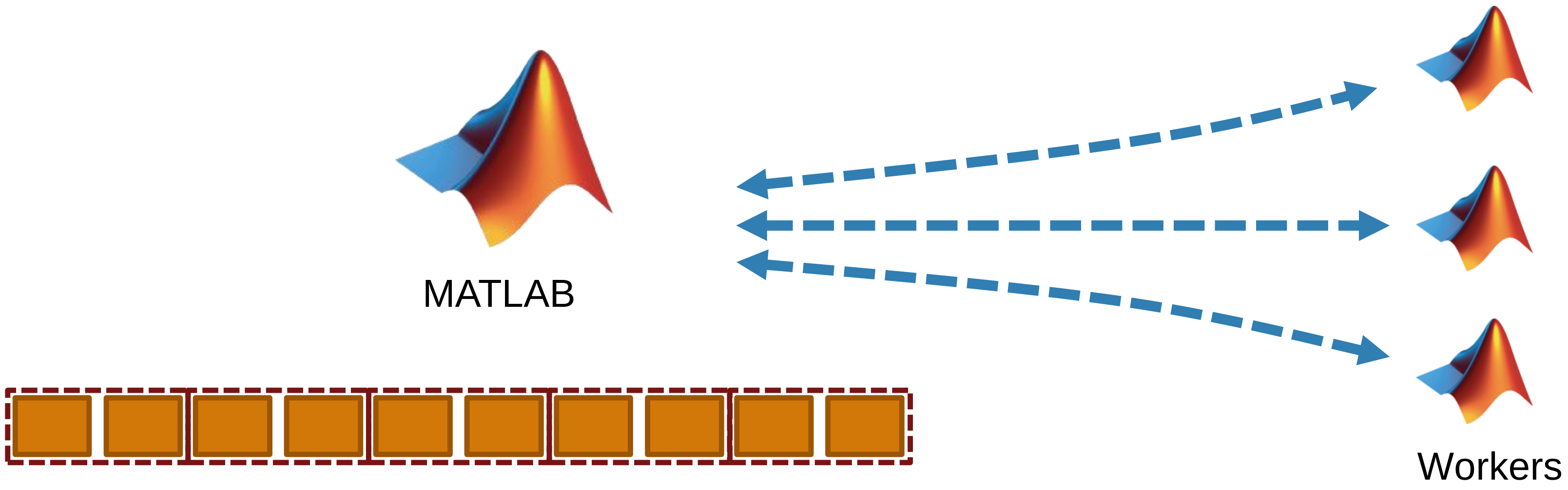
Run independent iterations in parallel using parfor




```
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)
```

Parallelize for loops with independent iterations



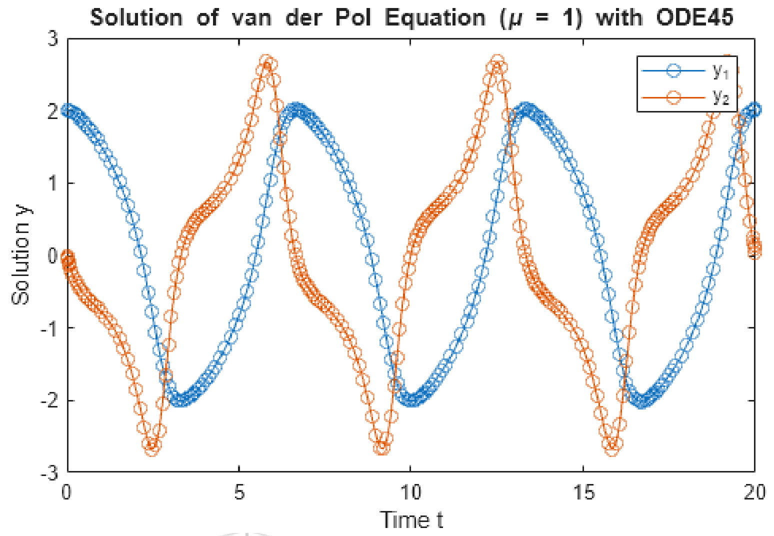

```
a = zeros(10, 1);
b = pi;
parfor i = 1:10
  a(i) = i + b;
end
disp(a)
```

Example: Parameter Sweep with parfor

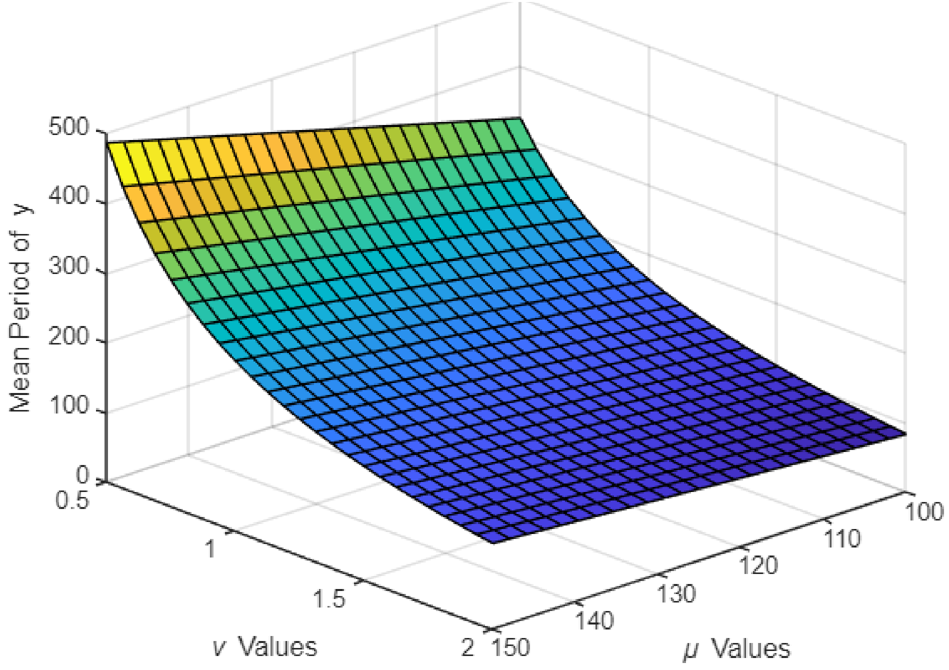
- Task
  - parameter sweep on a system
  - Van der Pol oscillator
  - find out the mean period

$$\dot{x} = \nu y$$

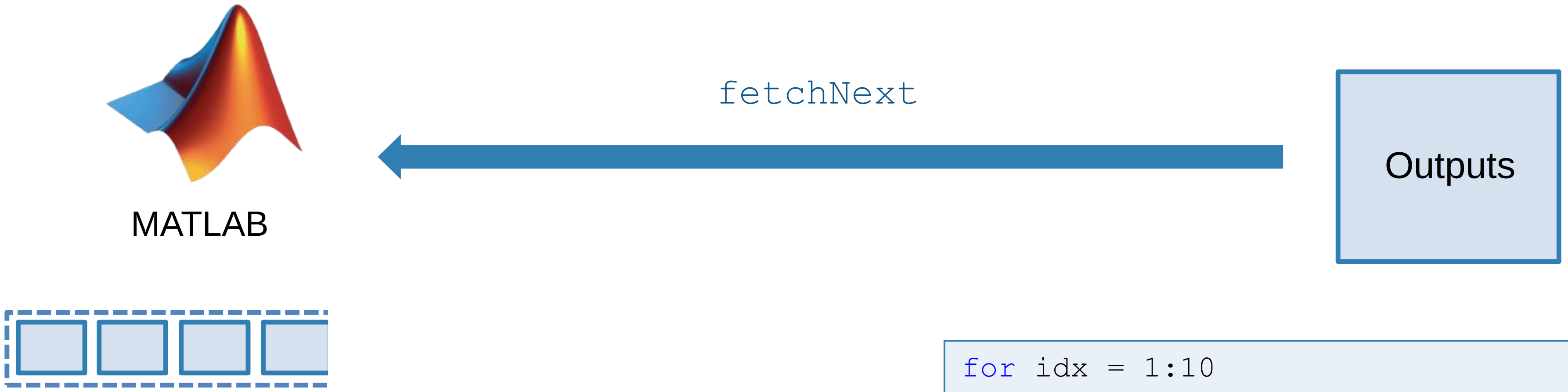
$$\dot{y} = \mu(1 - x^2)y - x$$



- Solution
  - run the parameter sweep in serial
  - start a pool of workers
  - run the parameter sweep in an interactive parallel pool
  - compare results



Execute functions in parallel asynchronously using parfeval




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

for idx = 1:10
   [completedIdx, value] = fetchNext(f);
   if value>0.95
      f.cancel
      break
   else
      mcs(completedIdx) = value;
   end
end
```

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

Big Data Workflows

ACCESS DATA

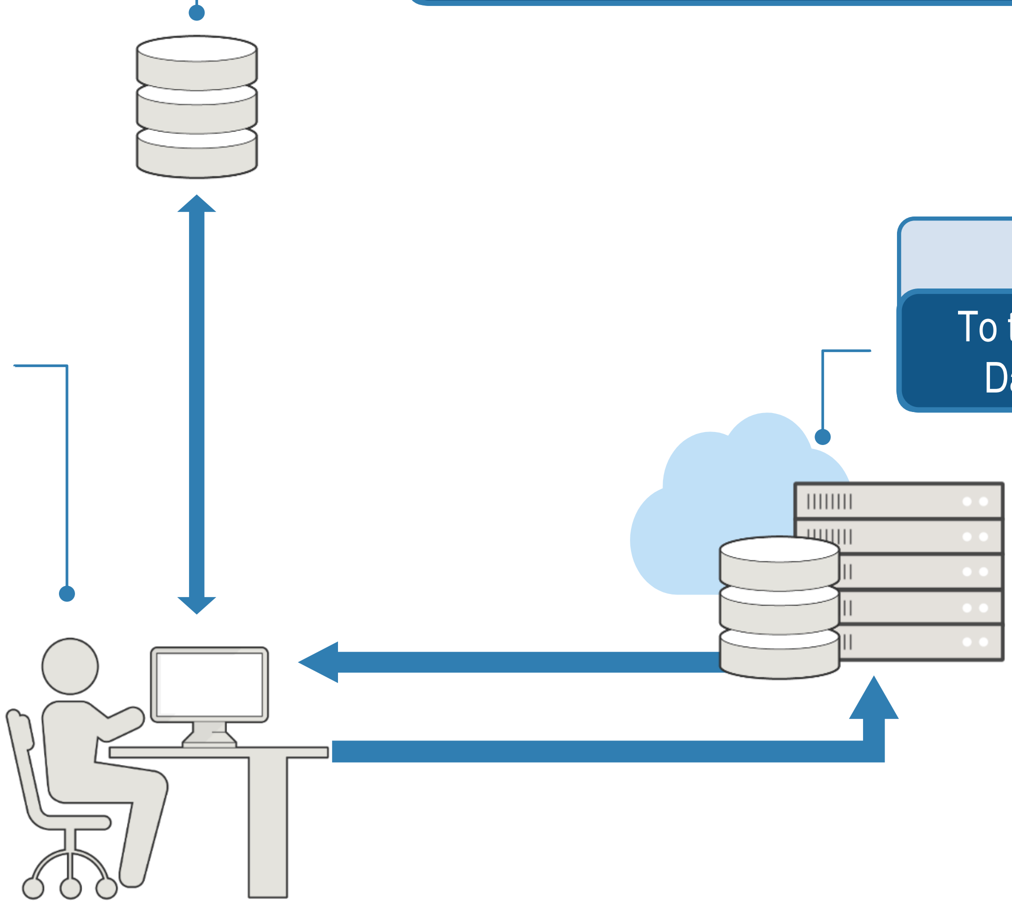
More data and collections of files than fit in memory

DEVELOP & PROTOTYPE ON THE DESKTOP

Adapt traditional processing tools or learn new tools to work with Big Data

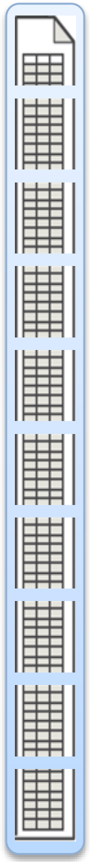
SCALE PROBLEM SIZE

To traditional clusters and Big Data systems like Hadoop



tall arrays

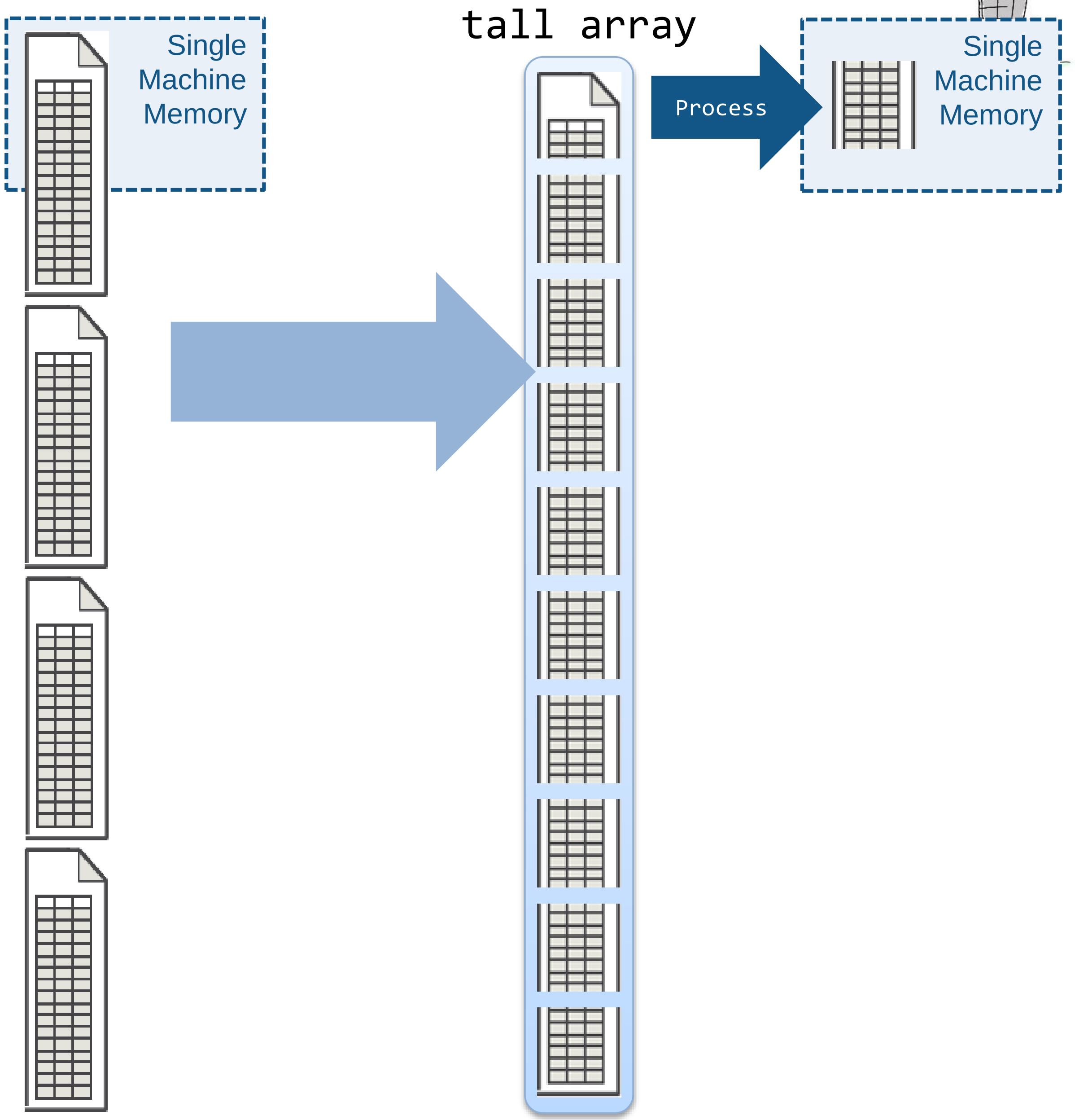
- Data type designed for data that doesn't fit into memory
- Lots of observations (hence "tall")
- Looks like a normal MATLAB array
  - Supports numeric types, tables, datetimes, strings, etc.
  - Supports several hundred functions for basic math, stats, indexing, etc.
  - **Statistics and Machine Learning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



Working with tall arrays

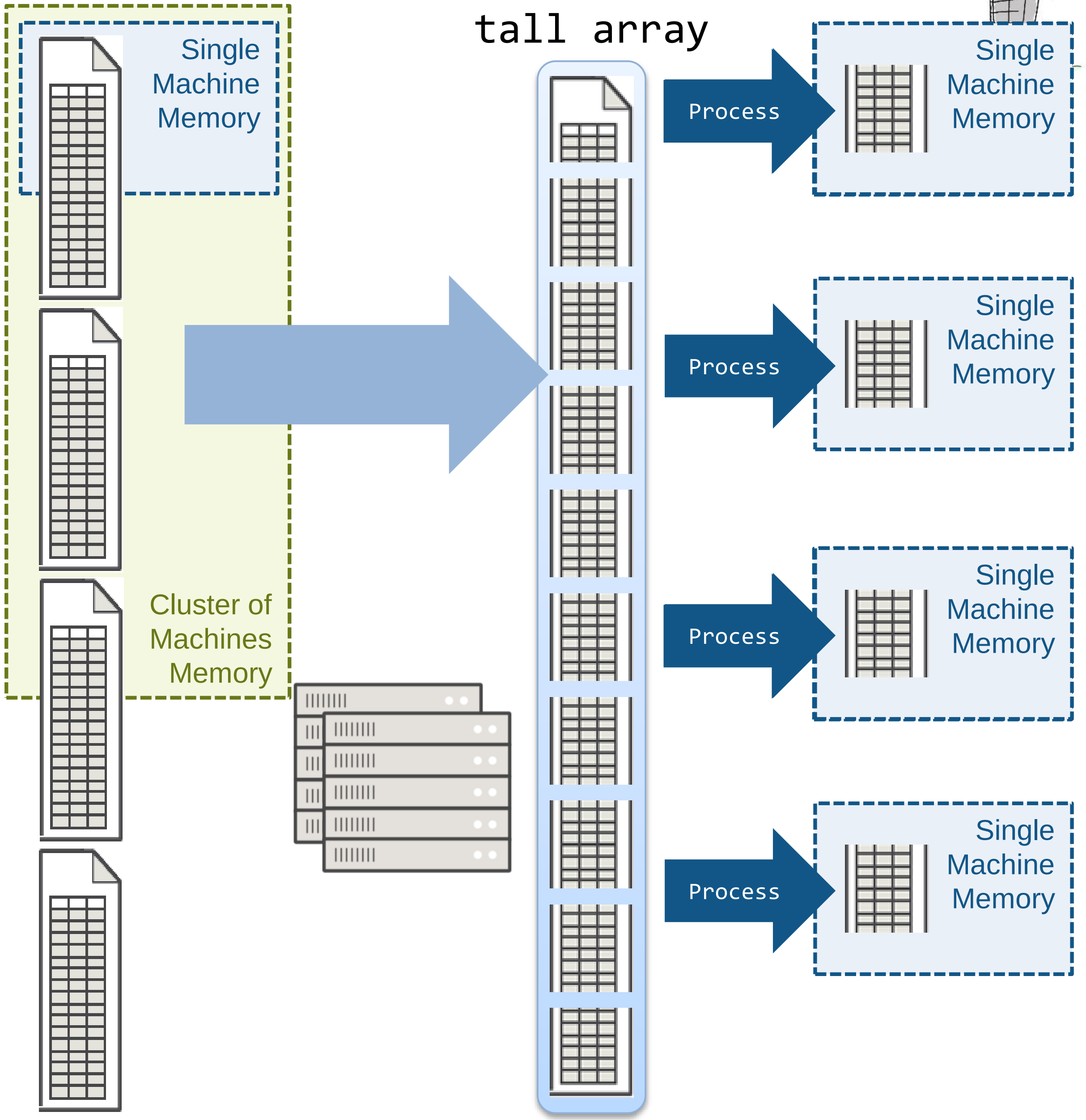
tall arrays

- 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
- Support for Spark and Hadoop



Scale preprocessing with tall arrays

One file

Access Data

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

Preprocess Data

Select data of interest

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

Work with missing data

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

Calculate statistics

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

One hundred files

Access Data

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

Preprocess Data

Select data of interest

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

Work with missing data

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

Calculate statistics

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

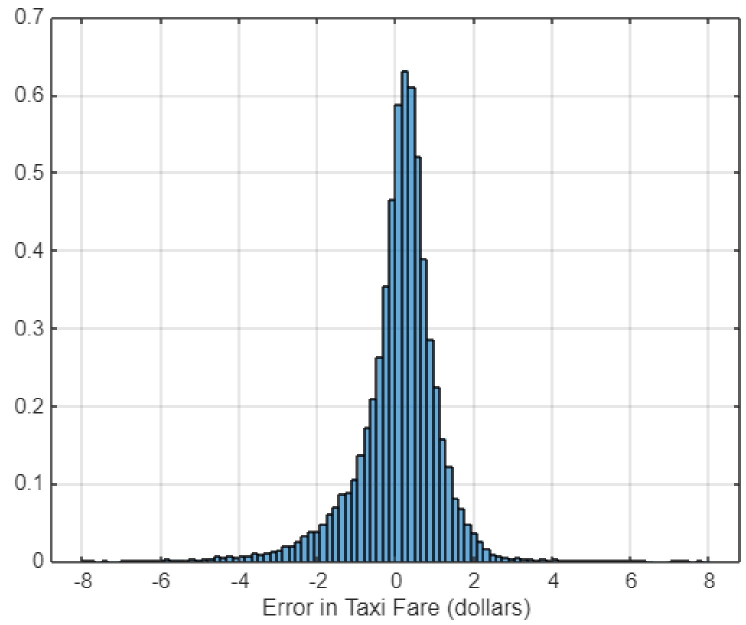
```
[m,s] = gather(m,s);
```

Example: Predict Cost of Taxi Ride in New York City

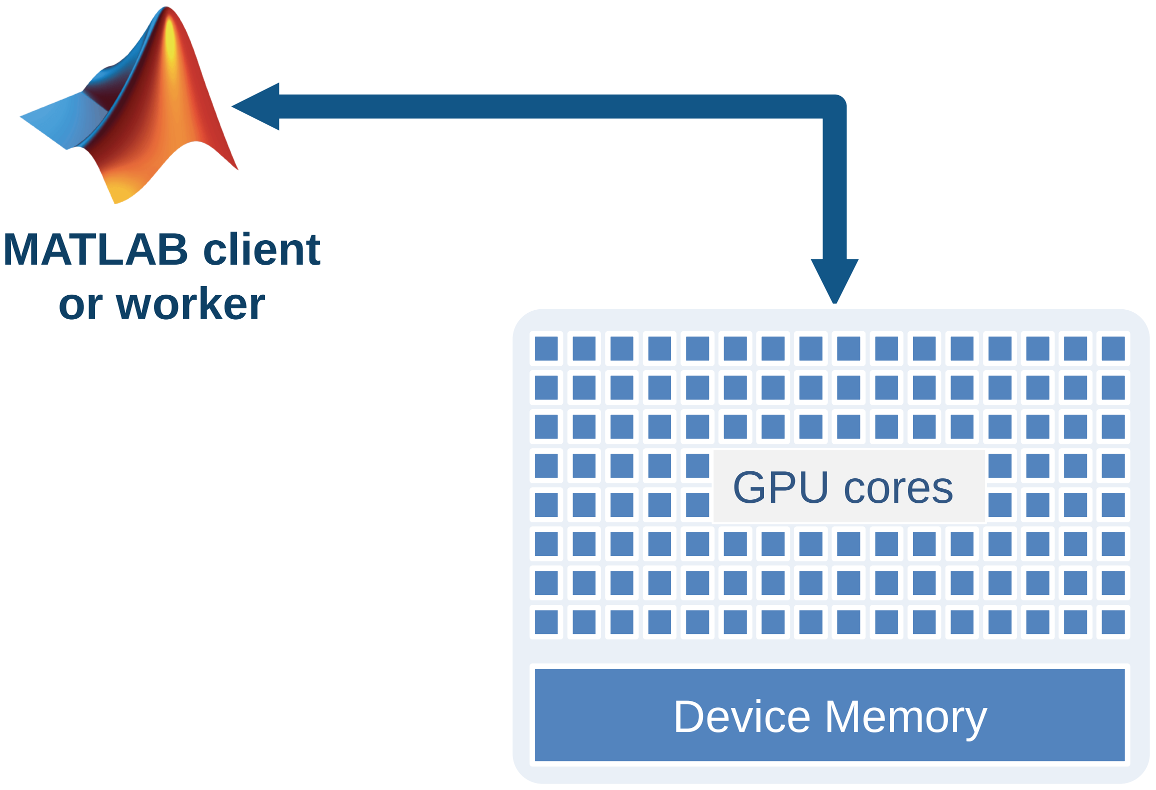
- Task
  - analyze data from .csv files
  - calculate average trip duration
  - predict taxi fare

- Solution
  - create datastore
  - create a tall array
  - plot fare amount vs trip distance
  - fit predictive model
  - predict and validate

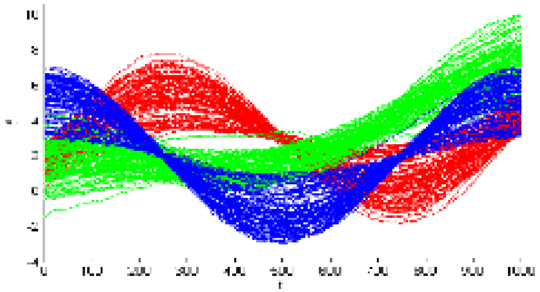
```
Compact linear regression model:
    fare_amount ~ 1 + hr_of_day + trip_distance*trip_minutes

Estimated Coefficients:
                                   Estimate         SE         tStat       pValue  
                                  __________    __________    _______    __________

    (Intercept)                       2.2373      0.022069     101.38             0
    trip_distance                     2.4731     0.0041884     590.46             0
    hr_of_day                      0.0048934      0.001178     4.1539    3.2711e-05
    trip_minutes                     0.26187     0.0010771     243.11             0
    trip_distance:trip_minutes    -0.0075663    9.1186e-05    -82.977             0


Number of observations: 118322, Error degrees of freedom: 118317
Root Mean Squared Error: 2.62
R-squared: 0.938,  Adjusted R-Squared: 0.938
F-statistic vs. constant model: 4.46e+05, p-value = 0
```



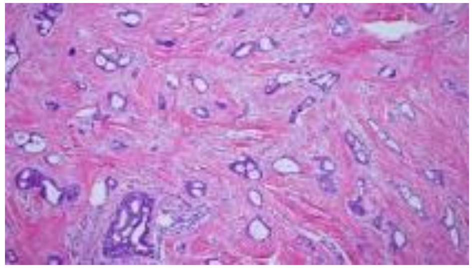
Leverage NVIDIA GPUs without learning CUDA



Accelerating with NVIDIA GPUs

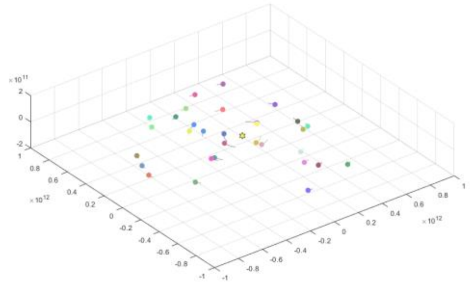


10x speedup
K-means clustering algorithm

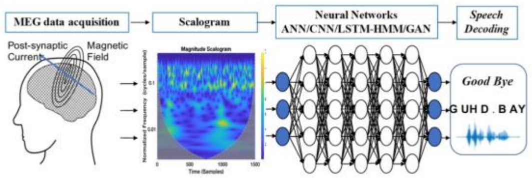


14x speedup
template matching routine

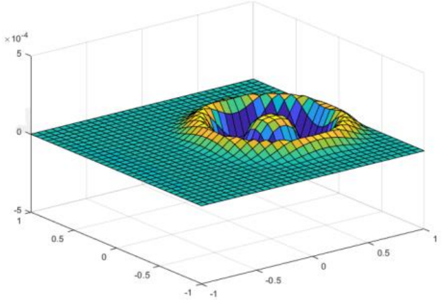
12x speedup
using Black-Scholes model



44x speedup
simulating the movement of celestial objects



10x speedup
deep learning training

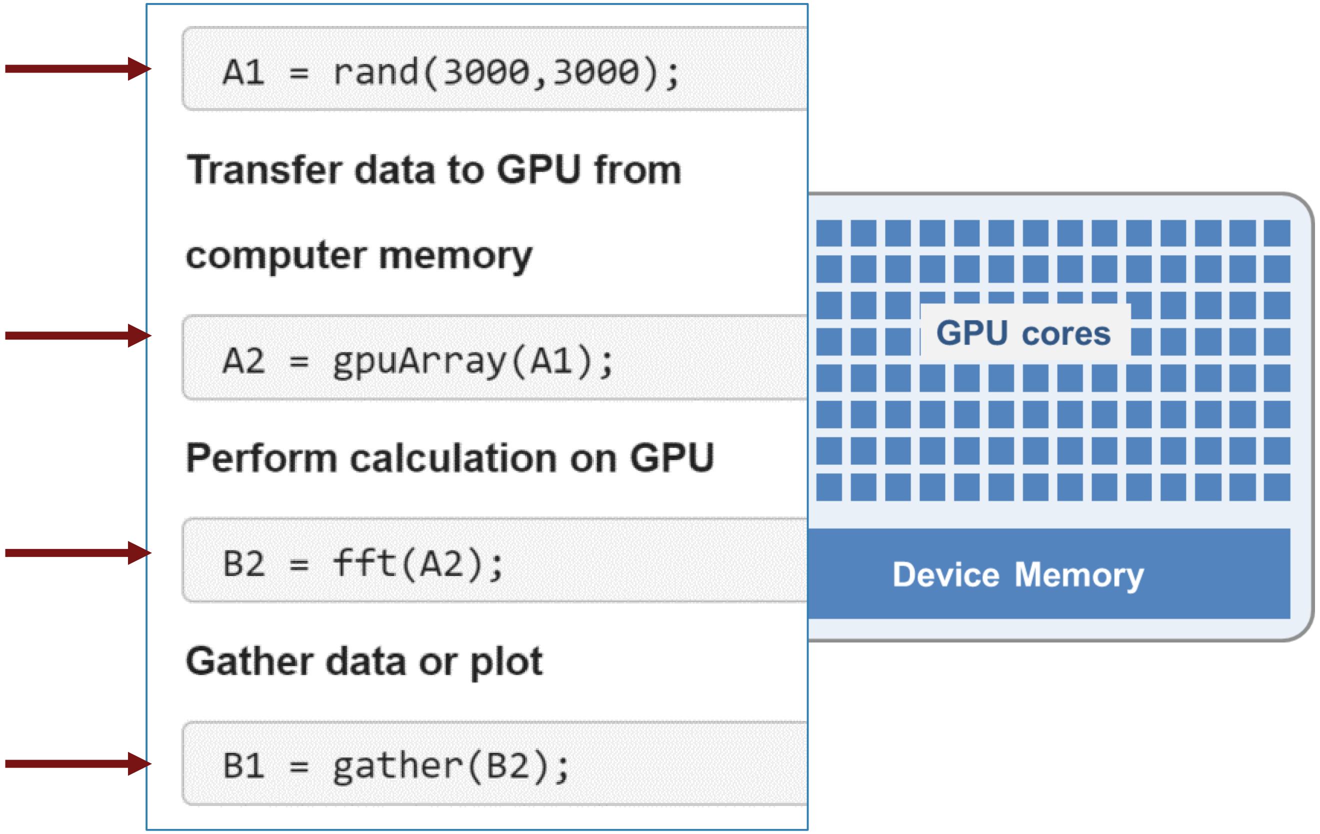


77x speedup
wave equation solving

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

Leverage your GPU to accelerate your MATLAB code

- Ideal Problems
  - massively parallel and/or vectorized operations
  - computationally intensive
- 1000+ GPU-supported functions
- Use `gpuArray` and `gather` to transfer data between CPU and GPU

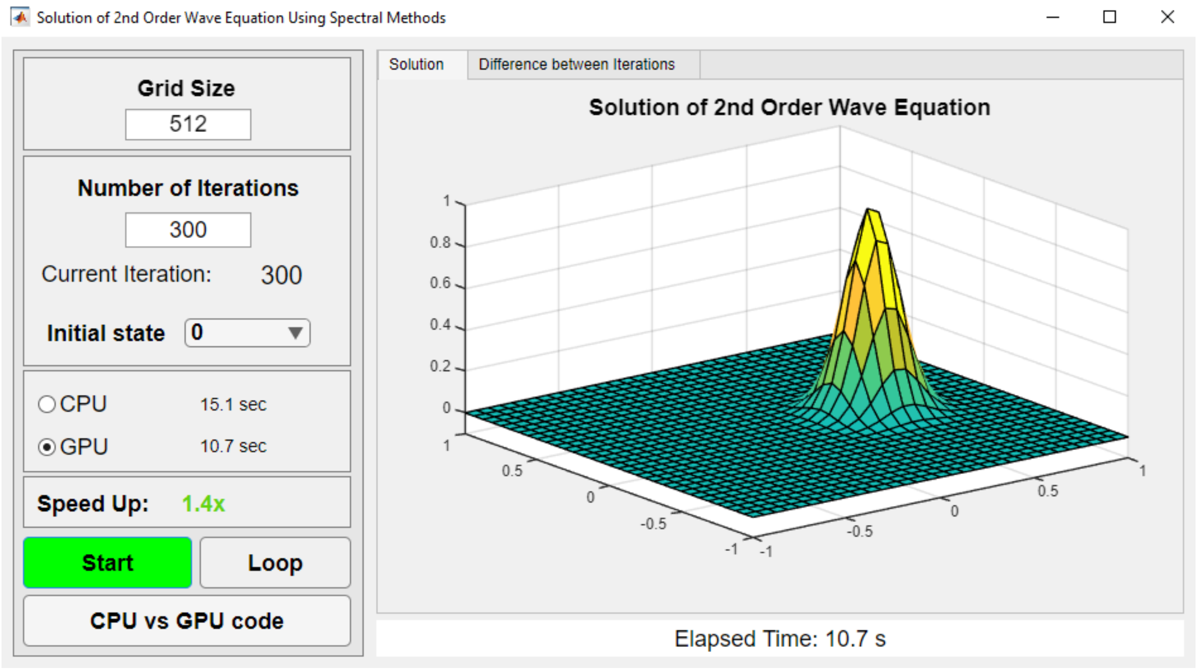


MATLAB GPU Computing

Example: Solving Equation on the GPU

- Task
  - solve a 2nd order wave equation
  - 2nd order central finite difference
  - Chebyshev spectral method (FFT)

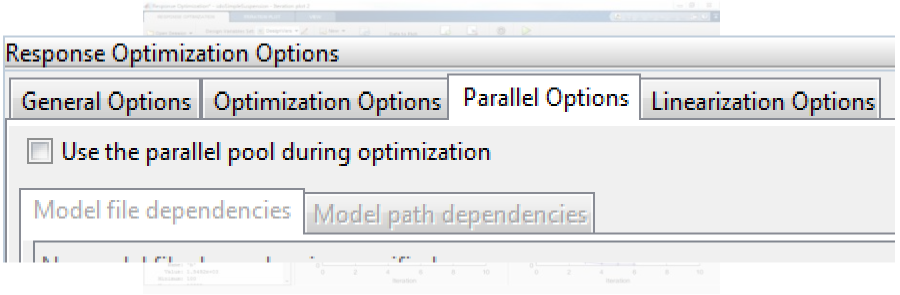
- Solution
  - set up parameters
  - run on the CPU
  - run on the GPU
  - compute speed up of solution



Automatic parallel support (Simulink)

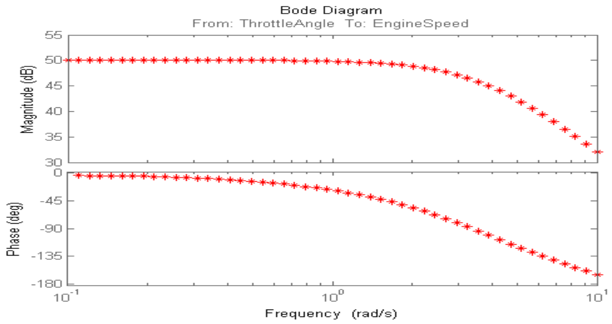
Simulink Design Optimization

Response optimization, sensitivity analysis, parameter estimation



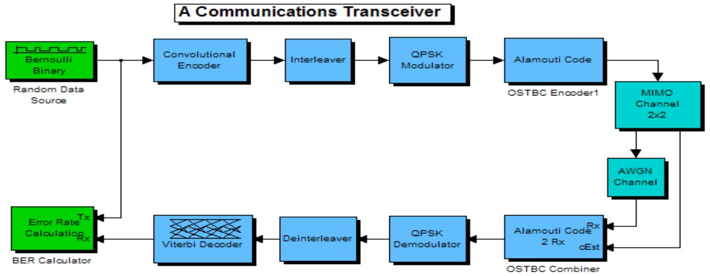
Simulink Control Design

Frequency response estimation



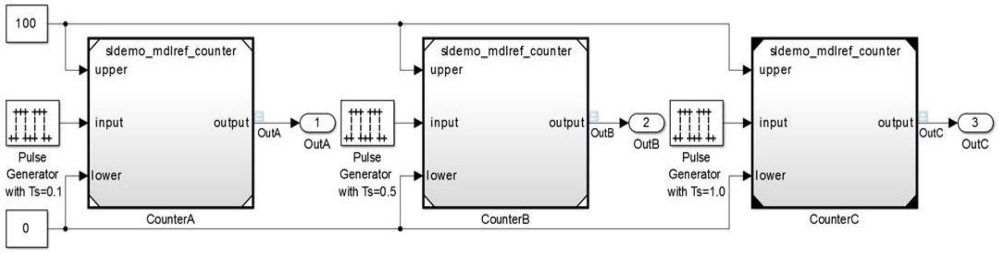
Communication Systems Toolbox

GPU-based System objects for Simulation Acceleration



Simulink/Embedded Coder

Generating and building code

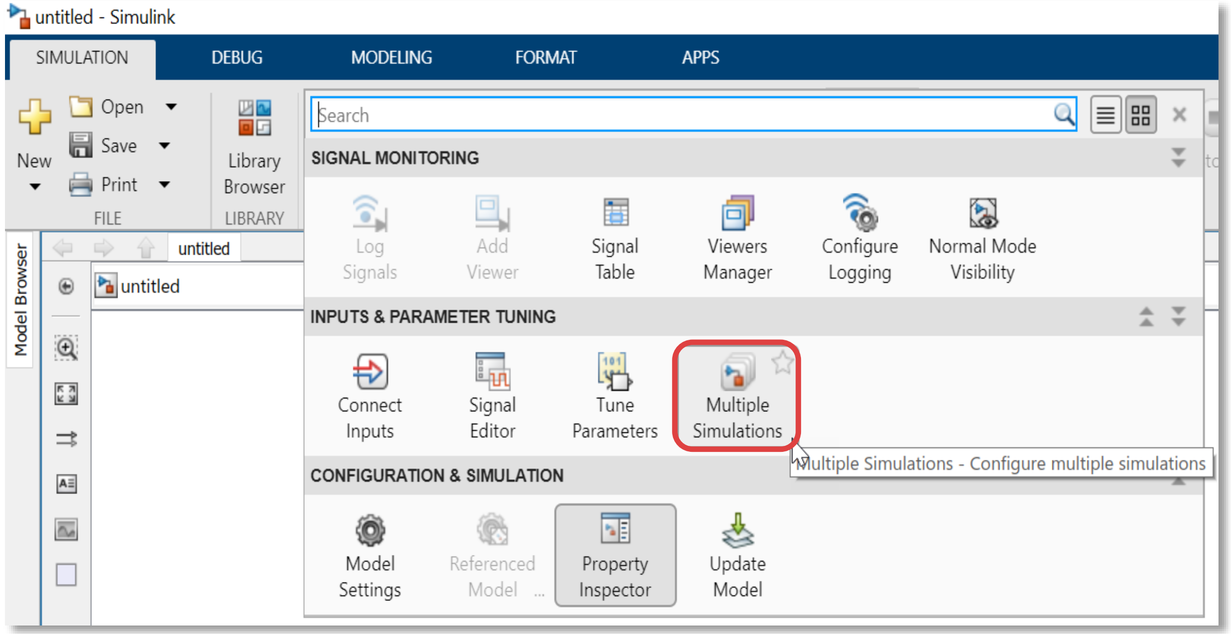


Learn more here

Run massive simulations in parallel with just a few clicks

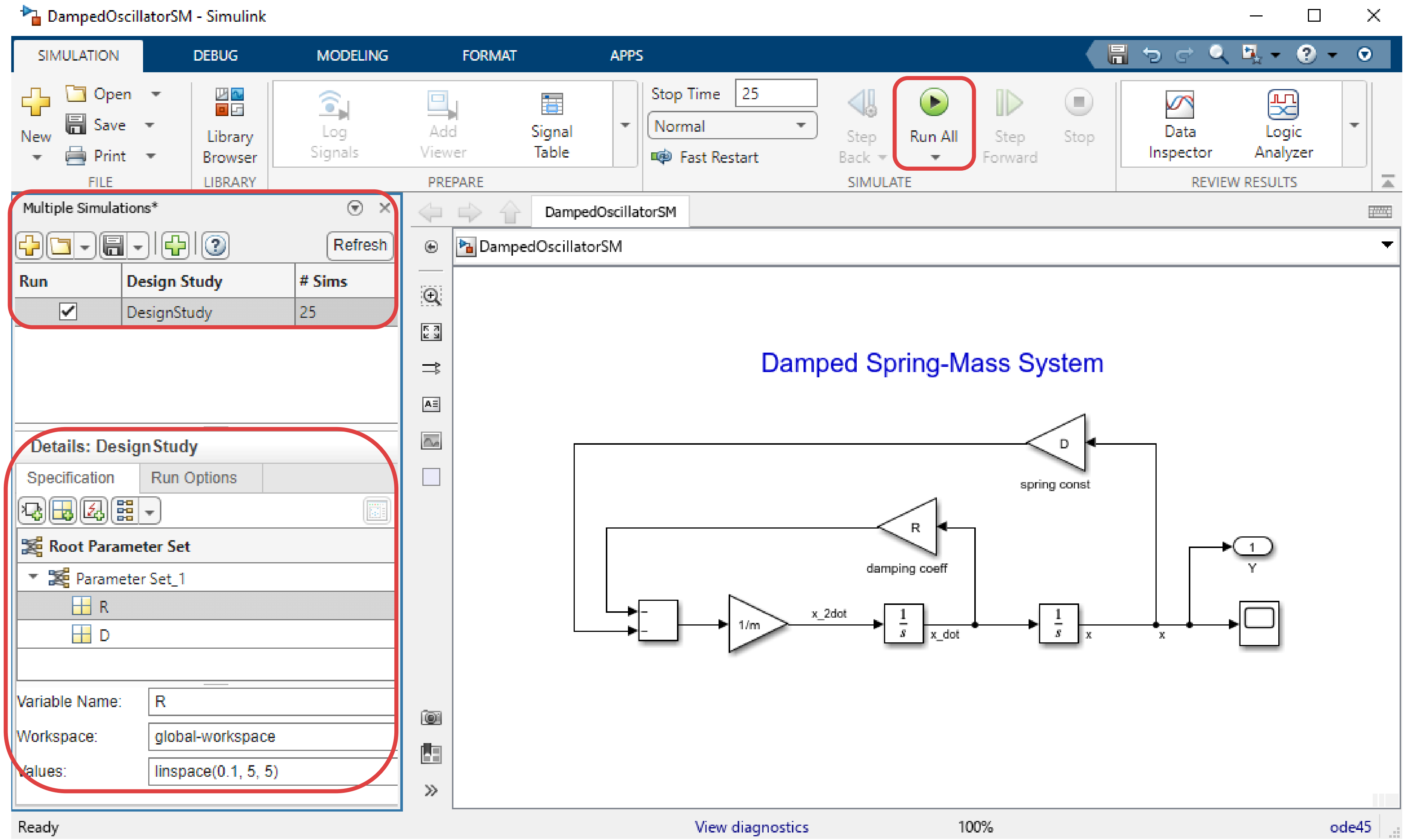
- Setup multiple simulations through a graphical user interface
- Specify variations on multiple block parameters or variables
- Integration with Simulation Manager
- Integration with parallel computing

Specify variations on multiple block parameters or variables

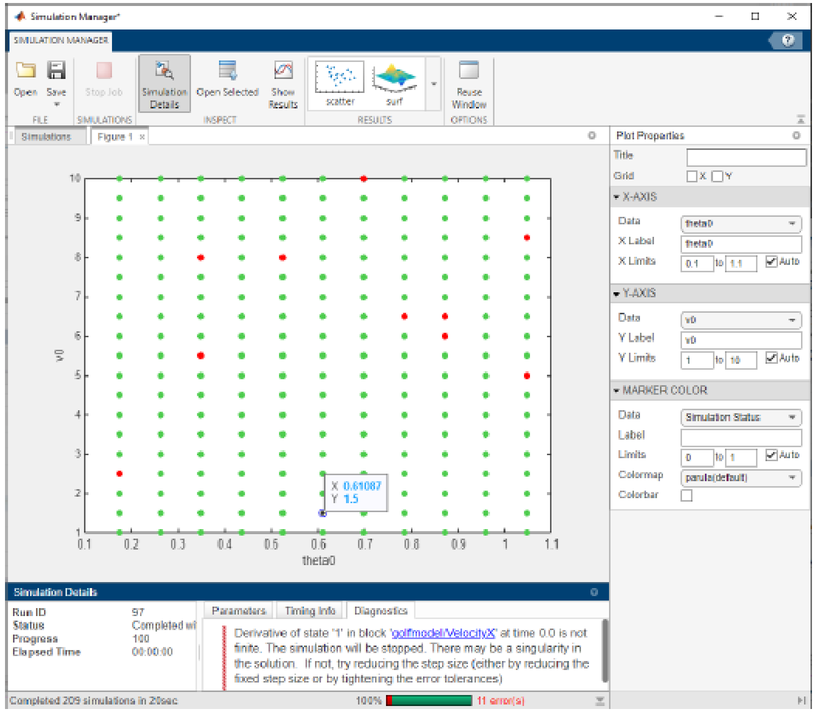


Learn more here

Run massive simulations in parallel with just a few clicks



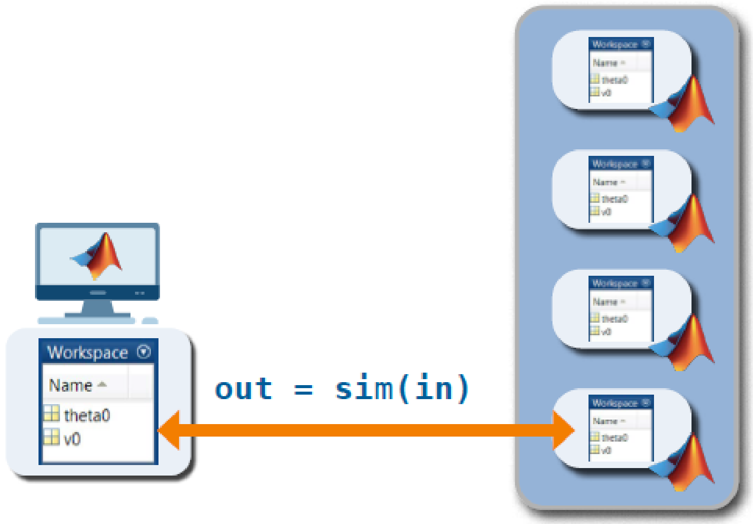
Automating Simulations with sim



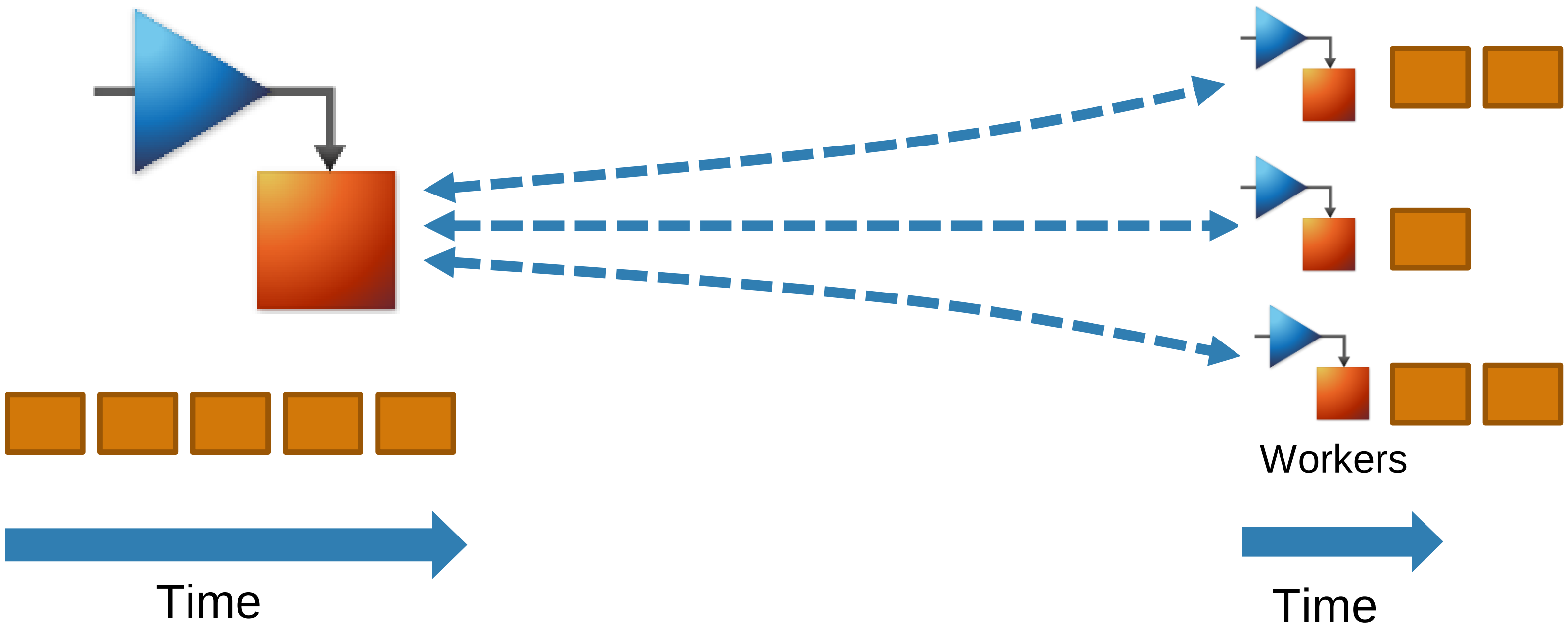
- single or multiple simulations can be run using the sim command

- control how the simulations are performed
  - *UseFastRestart* – skipping the compilation
  - *ShowProgress* – showing simulation progress
  - *ShowSimulationManager* – interactive monitoring

```
in(1:numSims) = Simulink.SimulationInput(model);
for idx = 1:numSims
   in(idx) = in(idx).setVariable(Var,values(idx));
   in(idx) = in(idx).setModelParameter(Name,Value);
end
```



Run multiple Simulink simulations in parallel with parsim



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

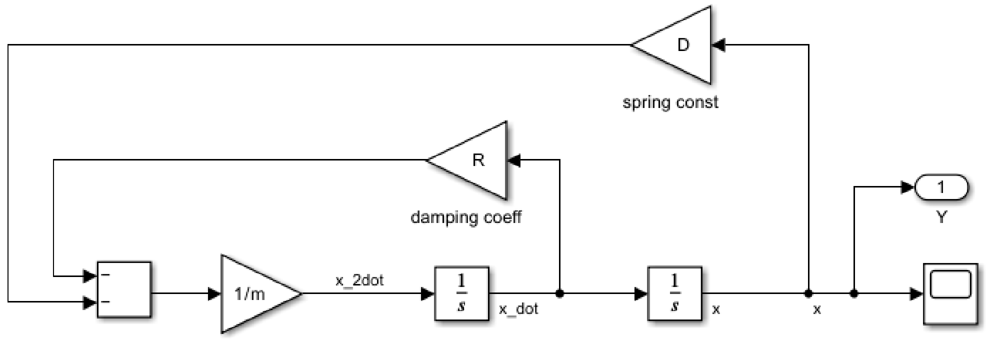
```
for i = 10000:-1:1
    in(i) = Simulink.SimulationInput(my_model);
    in(i) = in(i).setVariable(my_var, i);
end
out = parsim(in);
```

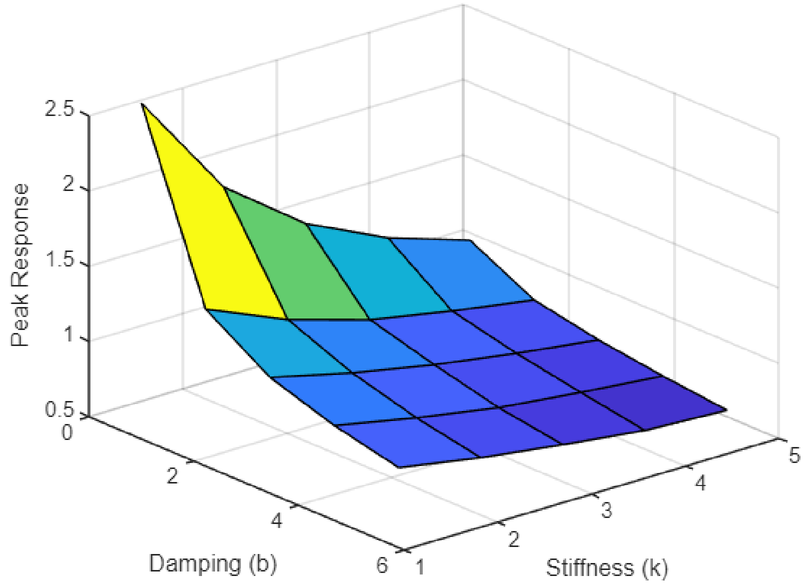
Example: Parallel Simulation with parsim

- Task
  - solve a Spring-Mass System
  - set up the SimulationInput object
  - call parsim

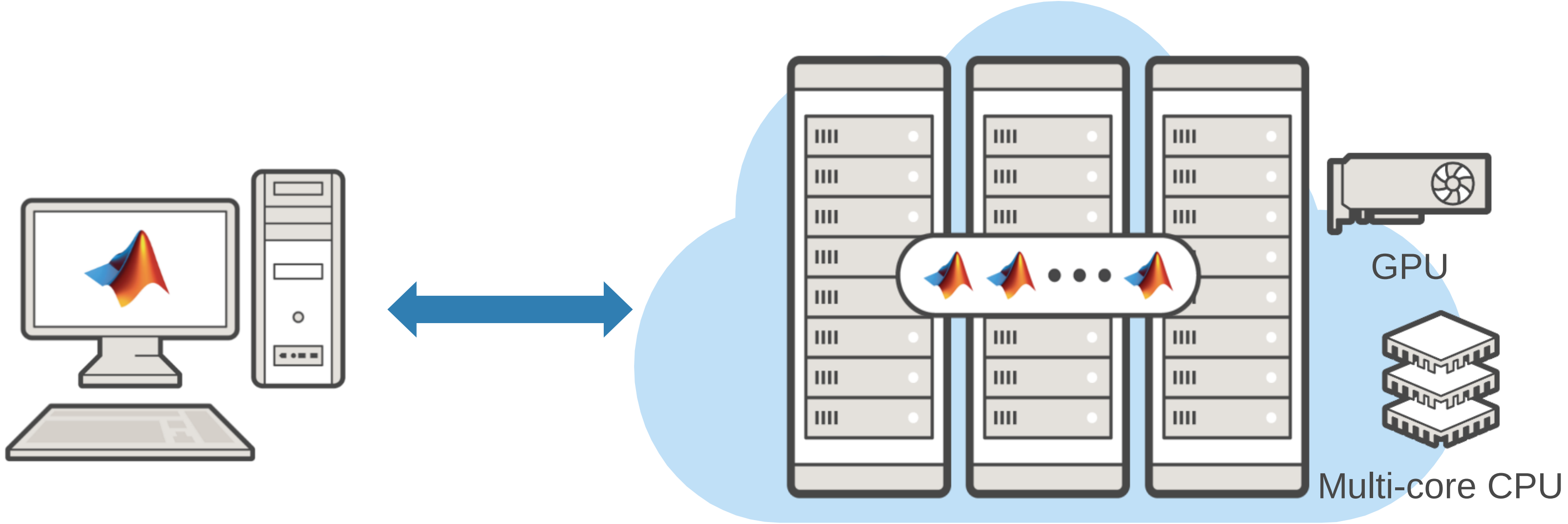
- Solution
  - set up the simulation
  - set the variables that are changing
  - simulate in parallel
  - unpack output
  - visualize parameter sweep

Damped Spring-Mass System





Access remote cluster resources



MATLAB
Parallel Computing Toolbox

MATLAB Parallel Server

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

Run a parallel pool from specified profile

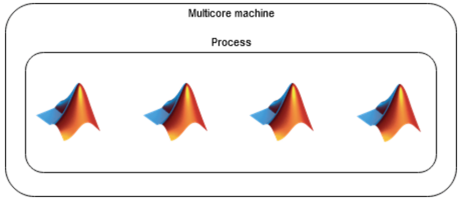
On local machine

- Start parallel pool of local workers

```
parpool('Processes');
```

- Start parallel pool of thread workers

```
parpool('Threads');
```



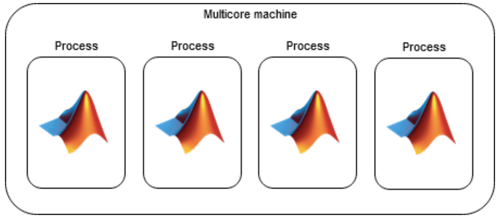
☺ Reduced memory usage, faster scheduling, lower data transfer costs

☹ Thread-based environments support only a subset of the functions available for process workers

On cluster

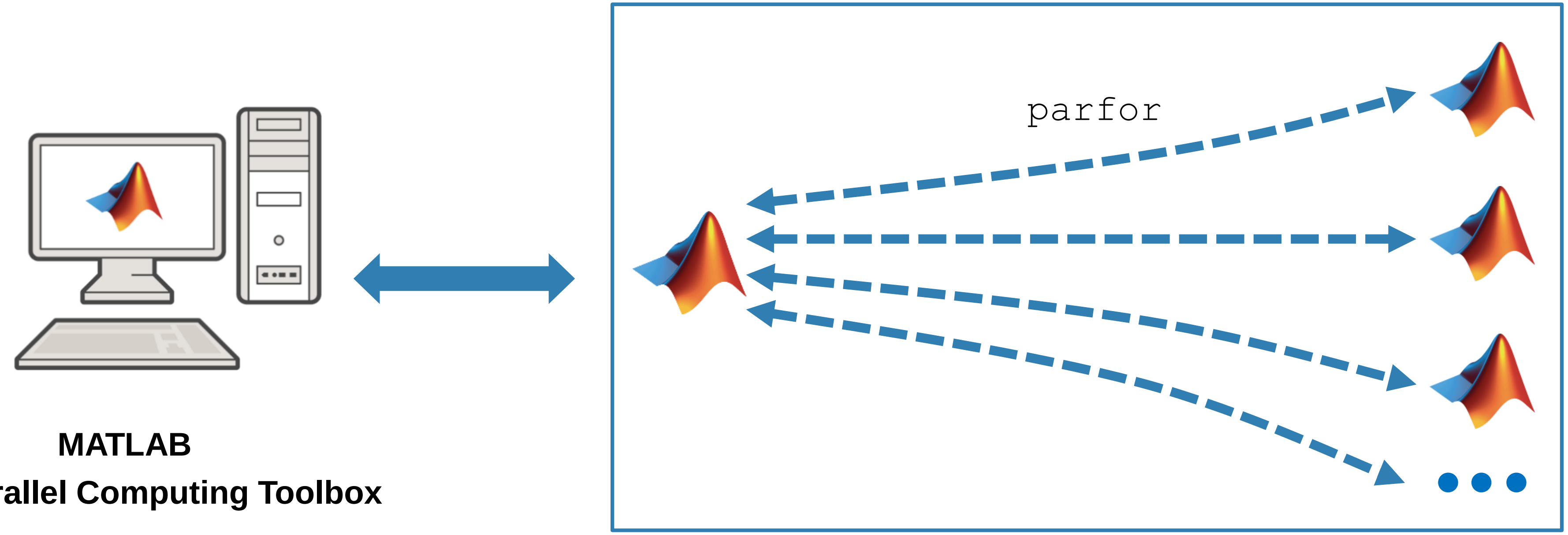
- Start parallel pool using cluster object

```
c = parcluster;
parpool(c);
```



Submit MATLAB jobs to the cluster

```
>> job = batch(myCluster,"myScript","Pool",1000);
```

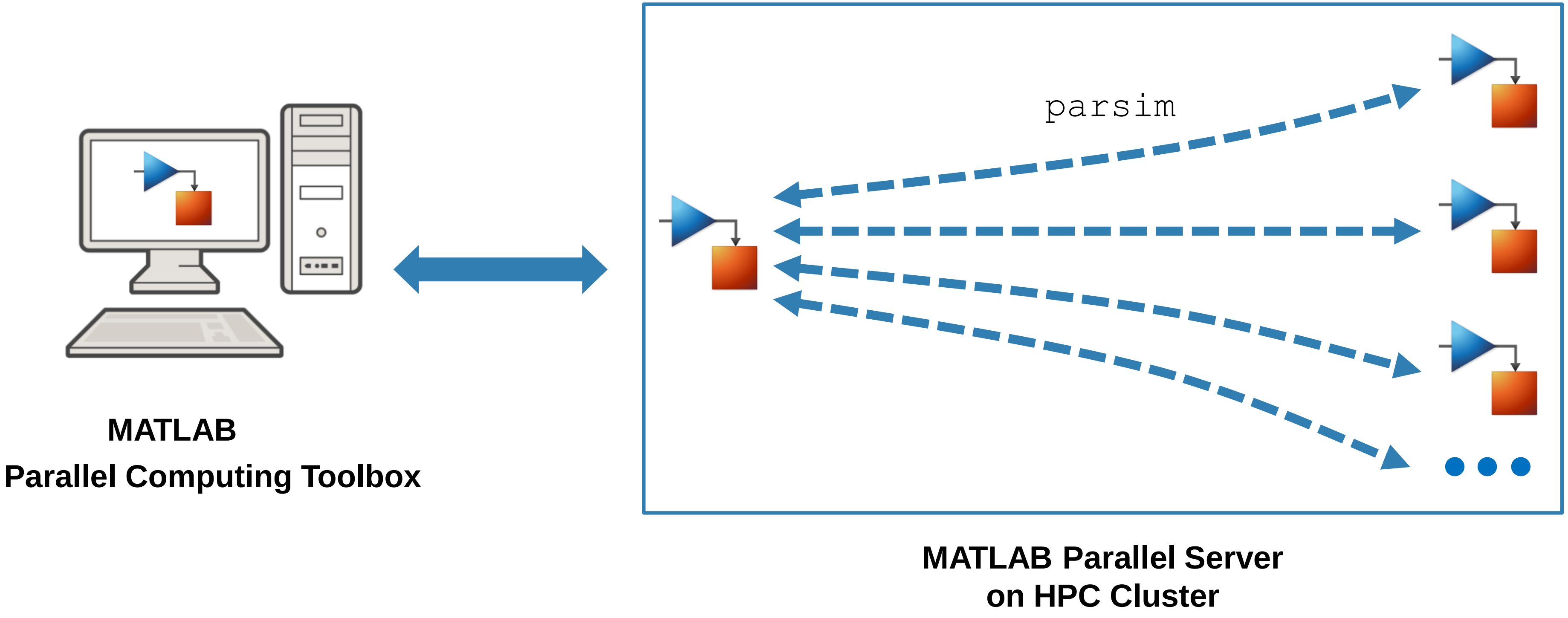


MATLAB
Parallel Computing Toolbox

MATLAB Parallel Server
on HPC Cluster

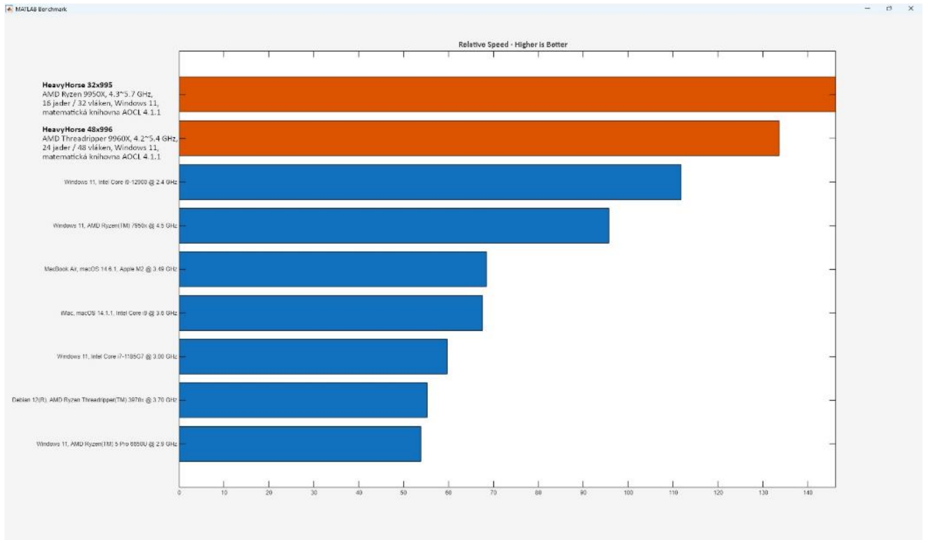
Submit Simulink jobs to the cluster

```
>> job = batchsim(myCluster,in,"Pool",1000);
```

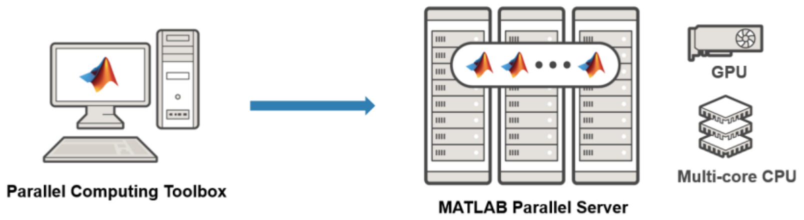


HeavyHorse - High-Performance Computing Workstations

- Hardware
  - CPU AMD Ryzen, Threadripper, EPYC
  - up to 384 cores, 768 threads
  - 32 ~ 768 GB RAM (up to 3072 GB RAM)
  - GPU – NVIDIA RTX Ada / RTX PRO Blackwell
  - 1000 GB ~ 4000 GB M.2 NVMe SSD hard drive
  - PC Case: Mid Tower/5U rack and Mid Tower
- Applications
  - High-Performance Computing
  - Finite Element Method
  - Processing of large data

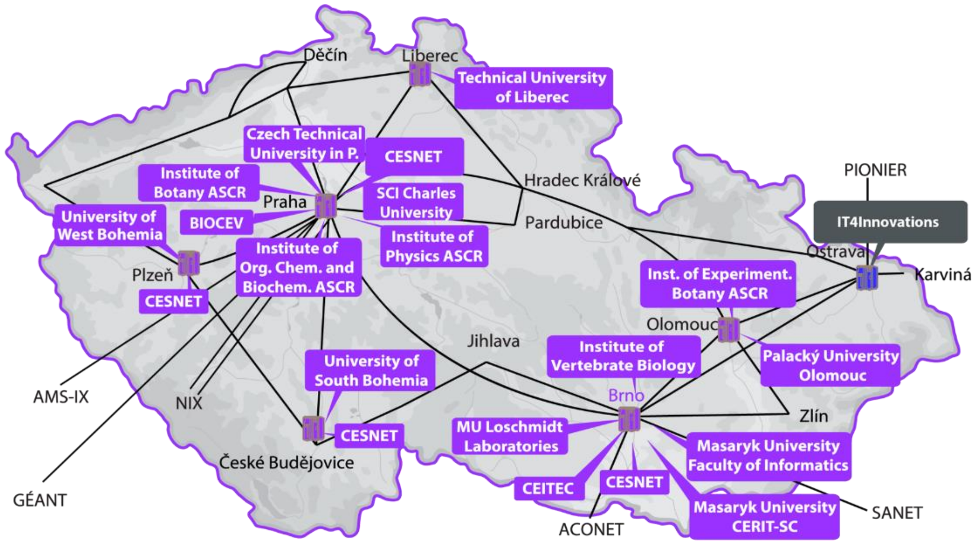


HPC / Big Data v ČR a SR



- Používatelia – akademická obec
  - CWL Univerzity
  - Akadémie Vied

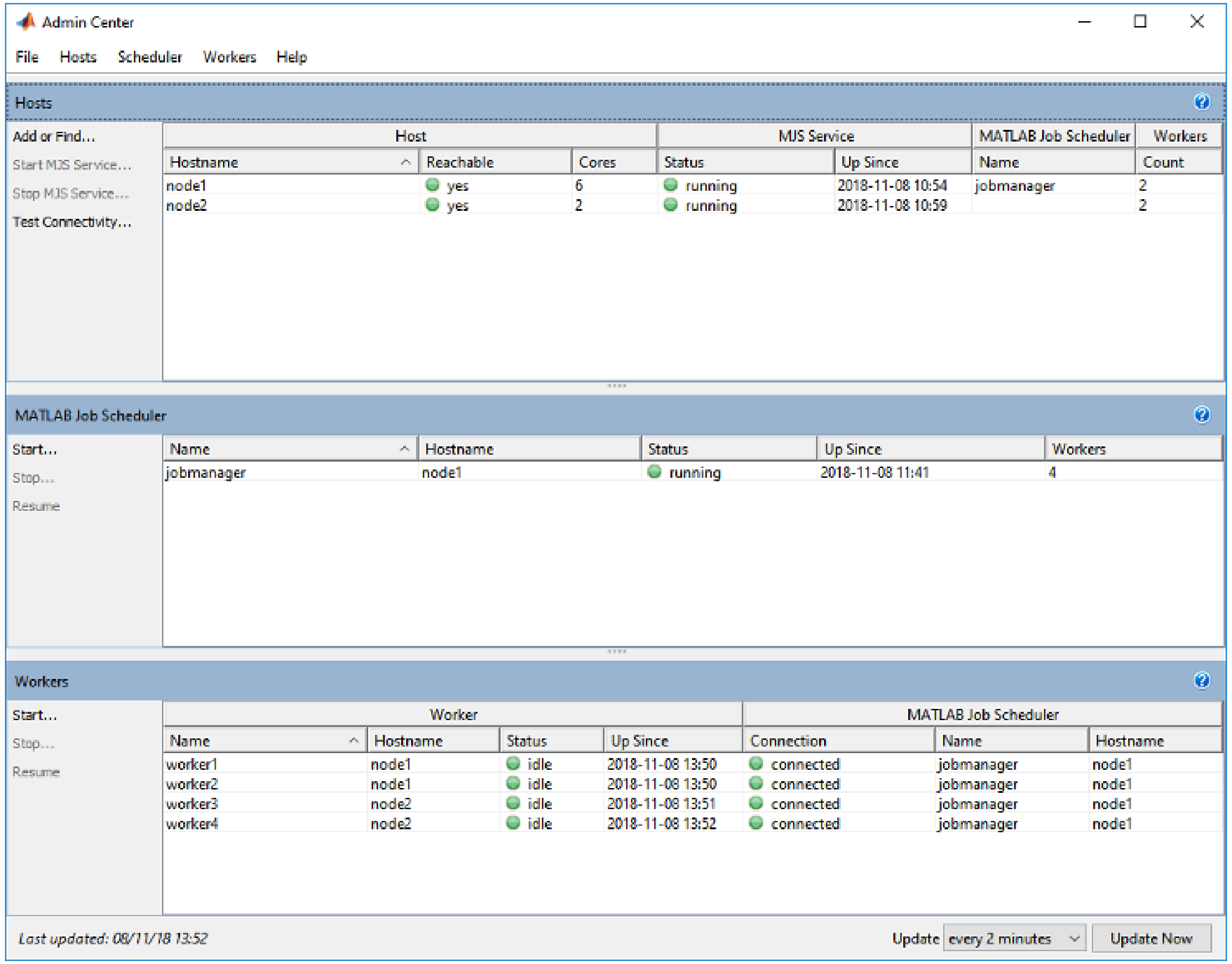
- HPC Infraštruktúra
  - CZ: e-INFRA
  - SK: NSCC & SAV



Example: Offloading to a cluster

- Task
  - perform parameter sweep
  - offload computation to HPC cluster
  - call batch

- Solution
  - create cluster object
  - runs a batch job on the cluster
  - wait for it to finish
  - fetch the outputs
  - visualize parameter sweep



Školenia - Paralelné výpočty v prostredí MATLAB

- 3.12.2025 – online & prezenčne

- Náplň školenia
  - architektúra a konfigurácia výpočtového klastra
  - distribuované a dávkové úlohy
  - paralelné cykly
  - dátovo paralelné úlohy a distribuované polia
  - GPU výpočty

Ďakujem za pozornosť