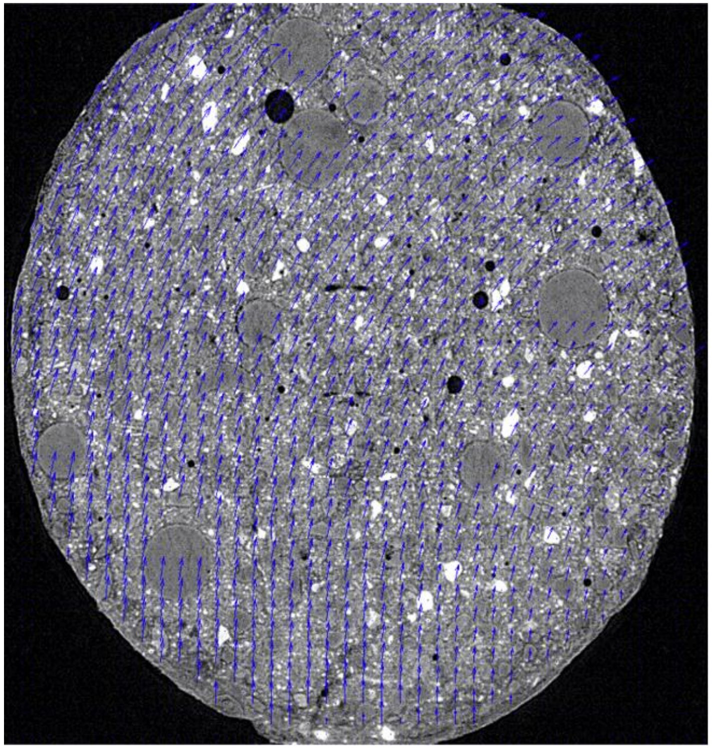
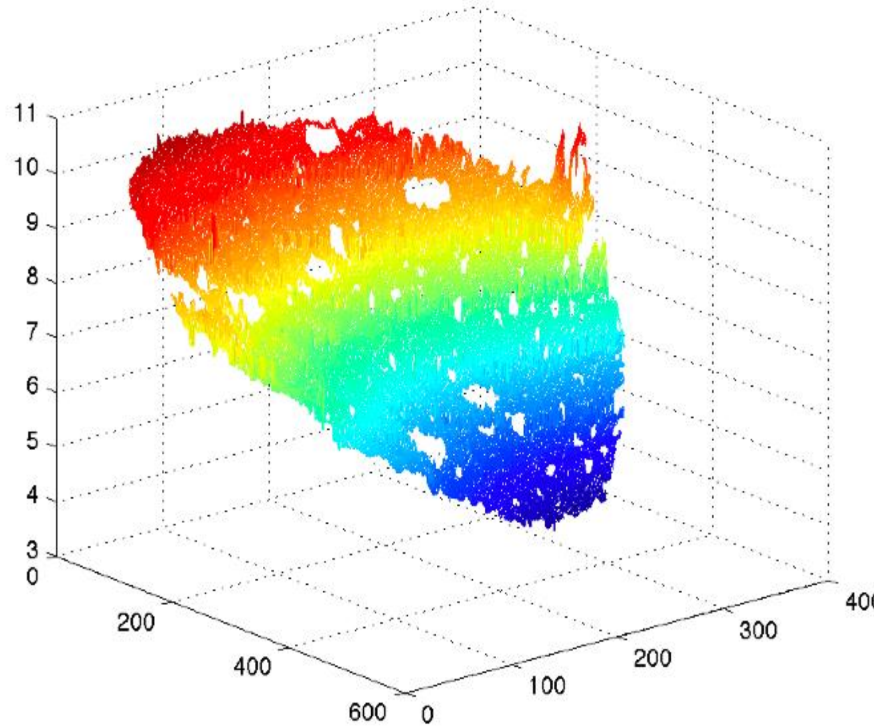
Speeding up MATLAB Applications

Sean de Wolski
Application Engineer

Non-rigid Displacement Vector Fields



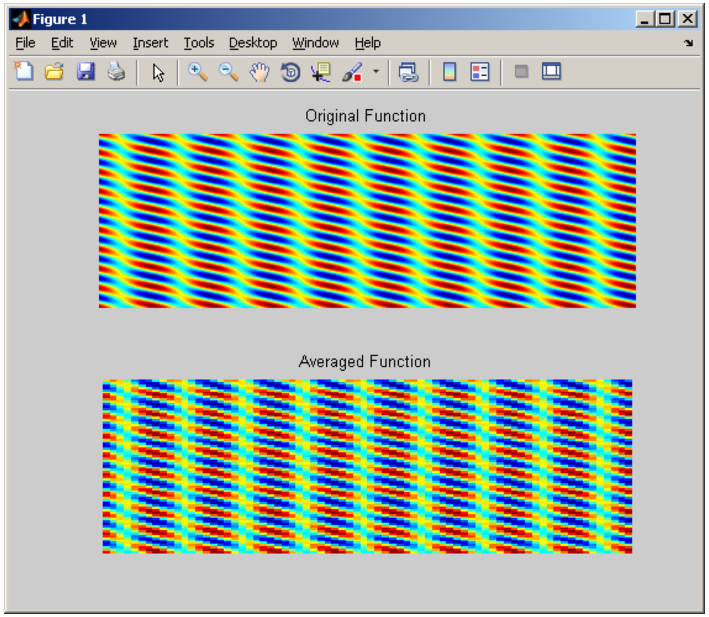


Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary

Example: Block Processing Images

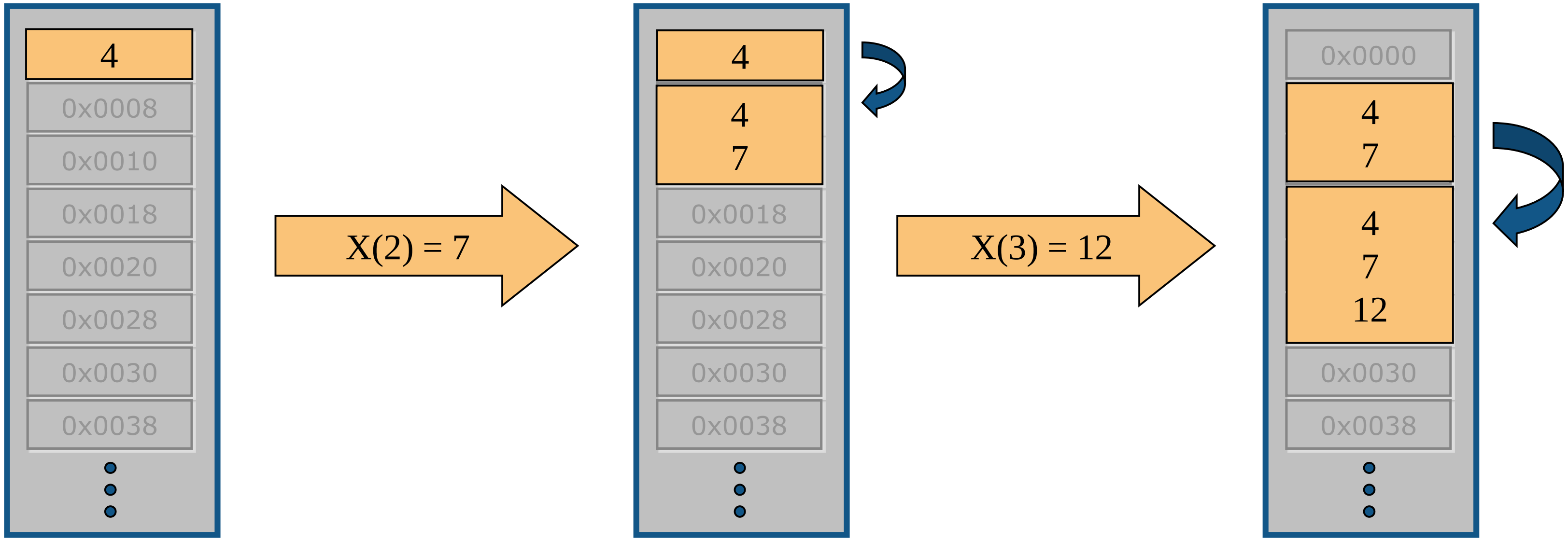
- Evaluate function at grid points
- Reevaluate function over larger blocks
- Compare the results
- Evaluate code performance



Effect of Not Preallocating Memory

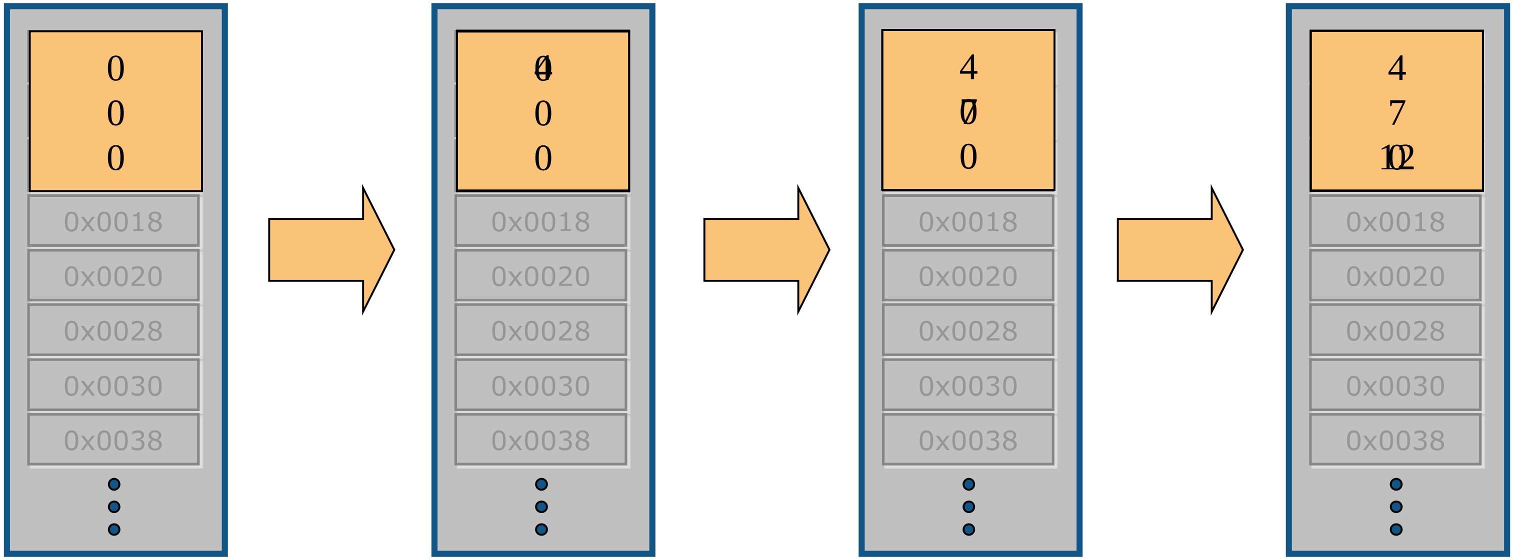
```
x = 4
x(2) = 7
x(3) = 12
```

Resizing Arrays is Expensive



Benefit of Preallocation

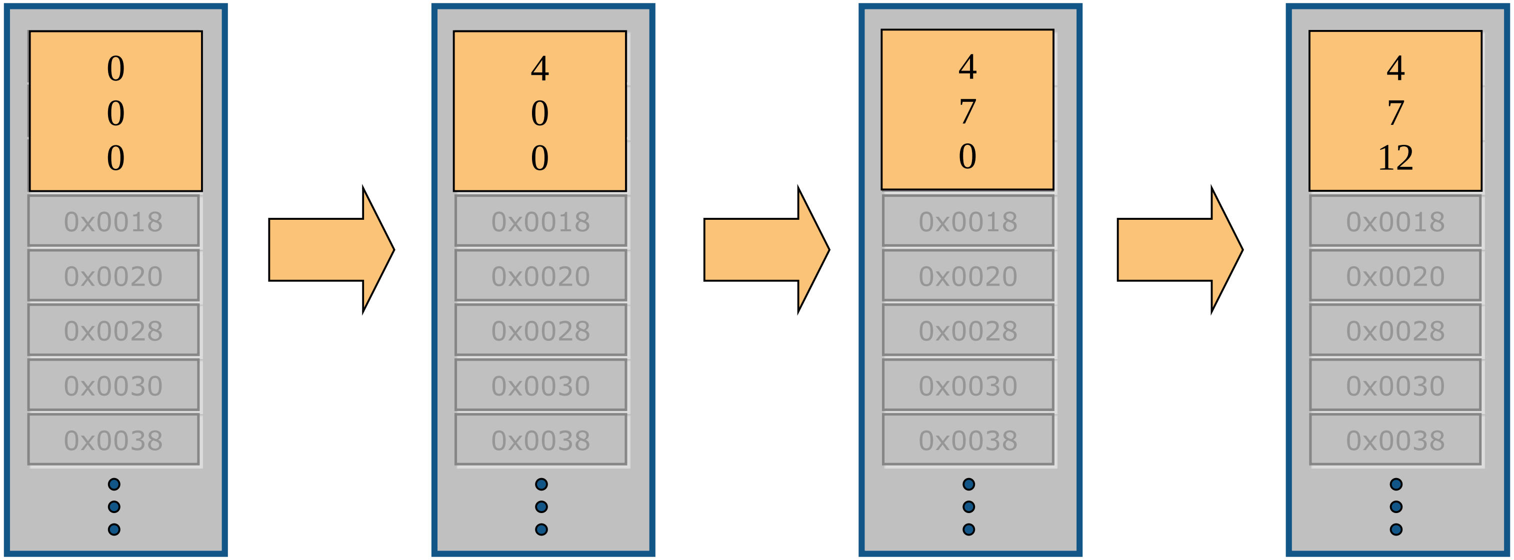
```
x = zeros(3,1)
x(1) = 4
x(2) = 7
x(3) = 12
```



Benefit of Preallocation

```
x = zeros(3,1)
x(1) = 4
x(2) = 7
x(3) = 12
```

Reduced Memory Operations



MATLAB Underlying Technologies

- Commercial libraries
  - BLAS: Basic Linear Algebra Subroutines (multithreaded)
  - LAPACK: Linear Algebra Package
  - etc.

BLAS and LAPACK require contiguous arrays

MATLAB Underlying Technologies

- JIT/Accelerator
  - Improves looping
  - Generates on-the-fly multithreaded code
  - Continually improving

Summary of Example: Tools

- Used built-in timing functions: **`tic, toc`**

```
>> tic; v = eig(rand(1000)); toc
Elapsed time is 1.033879 seconds.
>>
```

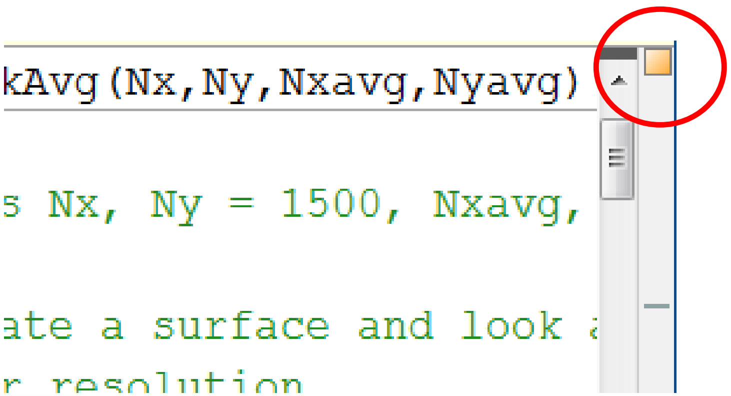
Summary of Example: Tools

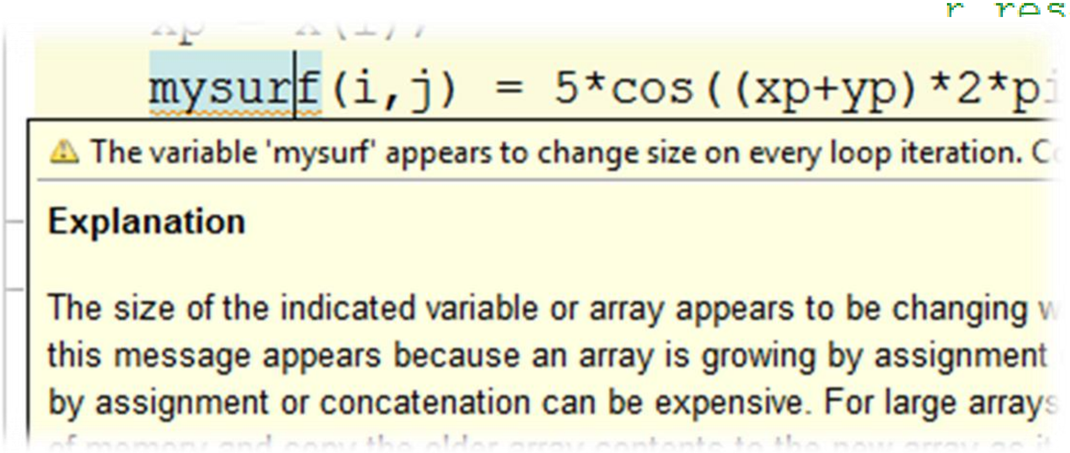
- Used built-in timing functions: **`timeit`**

```
>> t = timeit(@()svd(rand(1000)))
t =
    0.4190
>>
```

Summary of Example: Tools

- Used Code Analyzer to find suboptimal code

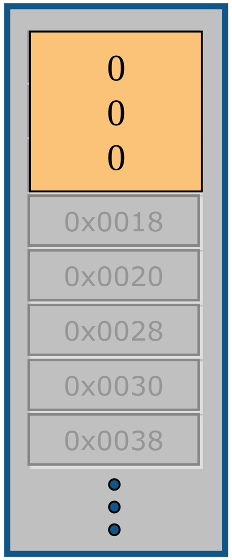




Summary of Example: Techniques

- Preallocated arrays

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



Summary of Example: Techniques

- Vectorized code

```
%% Setting up values of surface on grid

% Precomputation of inputs
[ygrid,xgrid] = meshgrid(y,x);

mysurf = 5*cos((xgrid+ygrid)*2*pi)+...
  2*sin(xgrid*2*pi)+2*cos(xgrid*2*pi);
```

Other Best Practices

- Minimize dynamically changing path

```
>> cd(...)
```

Other Best Practices

- Minimize dynamically changing path

```
>> cd( )
```

instead use:

```
>> addpath(…)
>> fullfile(…)
```

Other Best Practices

- Use the functional load syntax

```
>> load('myvars.mat')
```

Other Best Practices

- Use the functional load syntax

```
>> load('myvars.mat')
```

```
instead use:
>> x = load('myvars.mat')
x =
         a: 5
         b: 'hello'
```

Other Best Practices

- Minimize changing variable class

```
>> x = 1;
>> x = 'hello';
```

Other Best Practices

- Minimize changing variable class

```
>> x = 1;
>> x = 'hello';
```

instead use:

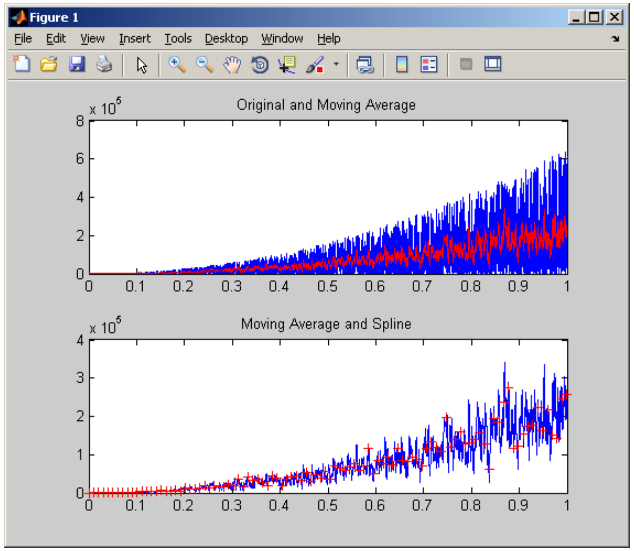
```
>> x = 1;
>> xnew = 'hello';
```

Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary

Example: Fitting Data

- Load data from multiple files
- Extract a specific test
- Fit a spline to the data
- Write results to Microsoft Excel



Summary of Example: Tools

- Profiler
  - Total number of function calls
  - Time per function call

Parents (calling functions)
No parent

Lines where the most time was spent

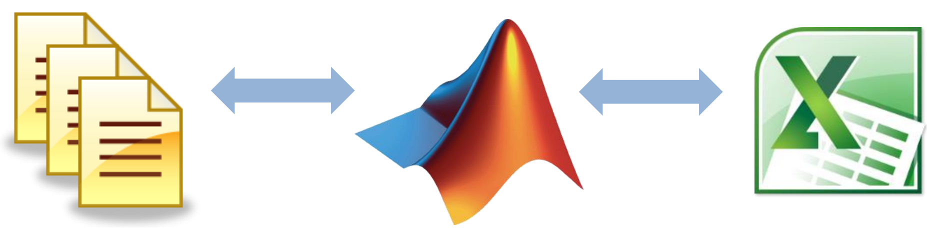
Line Number	Code	Calls	Total Time	% Time	Time Plot
79	`xlswrite(filexlsName,[splineTi...`	10	11.638 s	66.2%	
37	`textscan(fid, '%*f %*f \n',nTi...`	590	2.642 s	15.0%	
74	`saveas(gcf, fullfile('PlotFigs...`	10	2.115 s	12.0%	
58	`figure;`	10	0.526 s	3.0%	
28	`nTimes = textscan(fid,'%s',1);`	10	0.191 s	1.1%	
All other lines			0.481 s	2.7%	
Totals			17.592 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
xlswrite	function	10	11.638 s	66.2%	
saveas	function	10	2.114 s	12.0%	
newplot	function	40	0.069 s	0.4%	
subplot	function	20	0.054 s	0.3%	

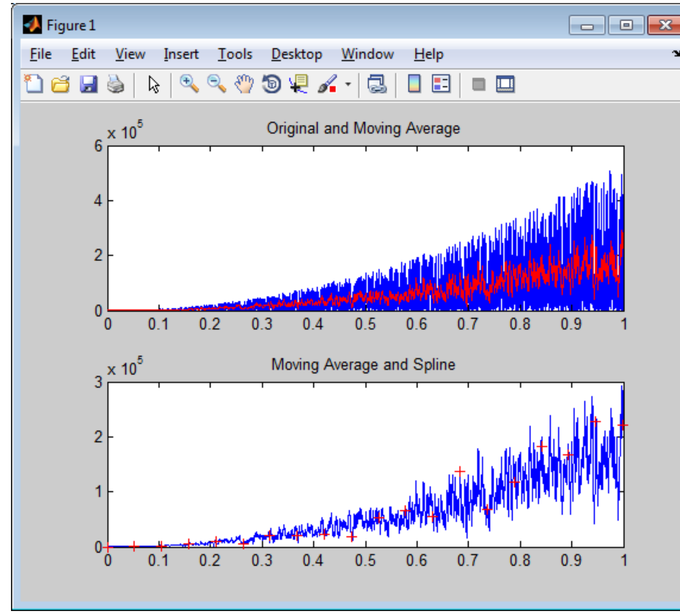
Summary of Example: Techniques

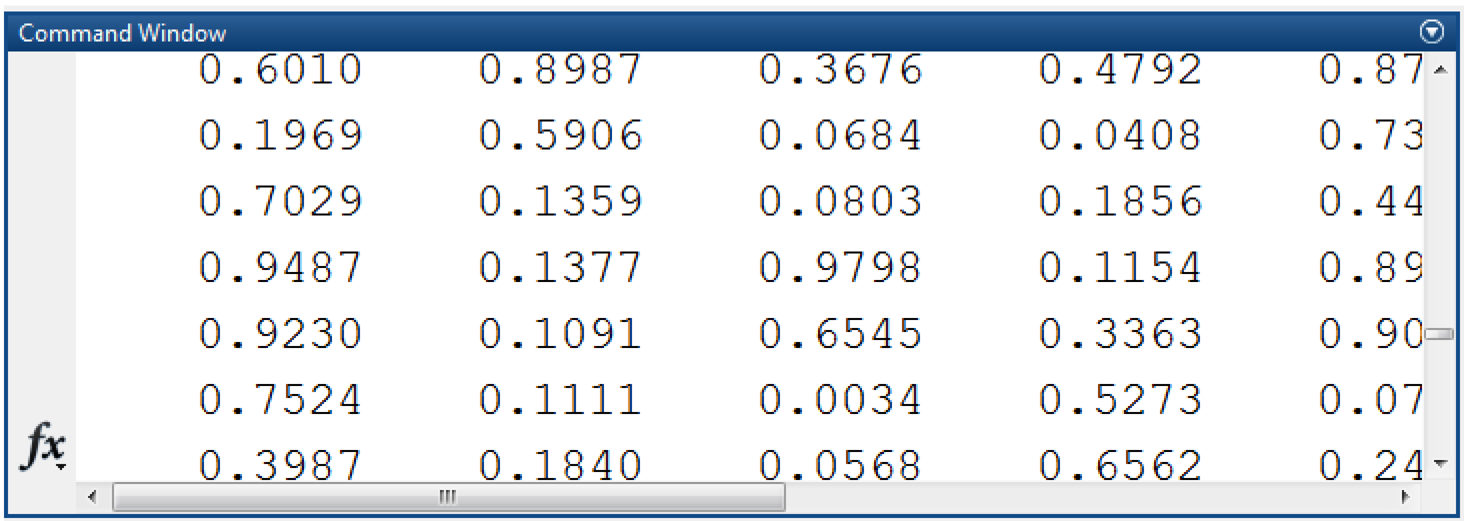
- Target significant bottlenecks
  - Reduce file I/O
  - Disk is slow compared to RAM
  - When possible, use `load` and `save` commands



Summary of Example: Techniques

- Target significant bottlenecks
  - Reuse figure
  - Avoid printing to command window





Steps for Improving Performance

- First focus on getting your code working
- Then speed up the code within core MATLAB
- Consider other languages (i.e. C or Fortran MEX files) and additional processing power

Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary

Why engineers and scientists translate MATLAB to C today?

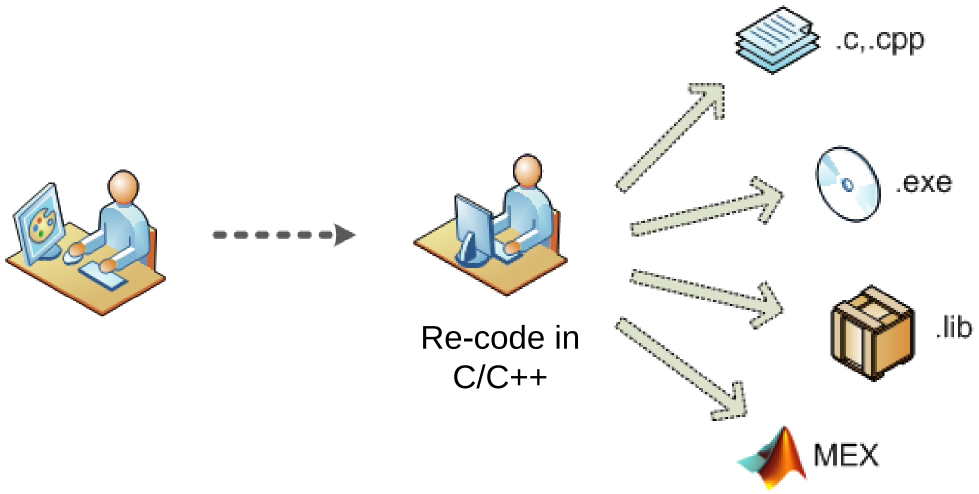
.lib — **Integrate** MATLAB algorithms w/ existing C environment using source code and static/dynamic libraries

.exe — **Prototype** MATLAB algorithms on desktops as standalone executables

MEX — **Accelerate** user-written MATLAB algorithms

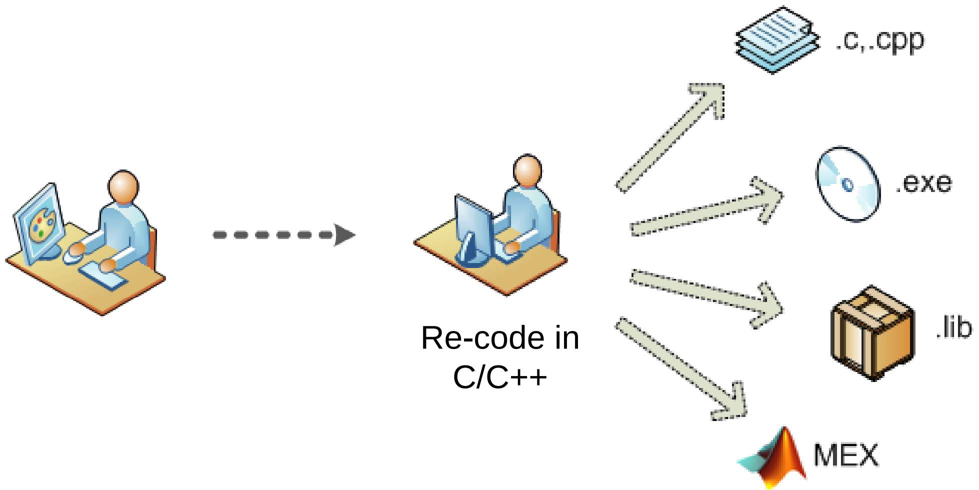
.c,.cpp — **Implement** C code on processors or hand-off to software engineers

Challenges with Manual Translation
from MATLAB to C



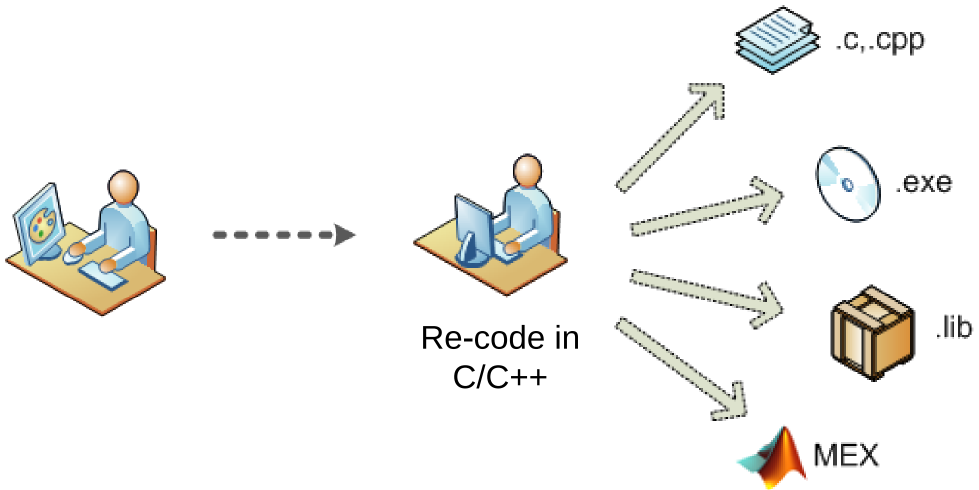
- Separate functional and implementation specification
  - Leads to multiple implementations that are inconsistent
  - Hard to modify requirements during development
  - Difficult to keep reference MATLAB code and C code in-sync

Challenges with Manual Translation
from MATLAB to C



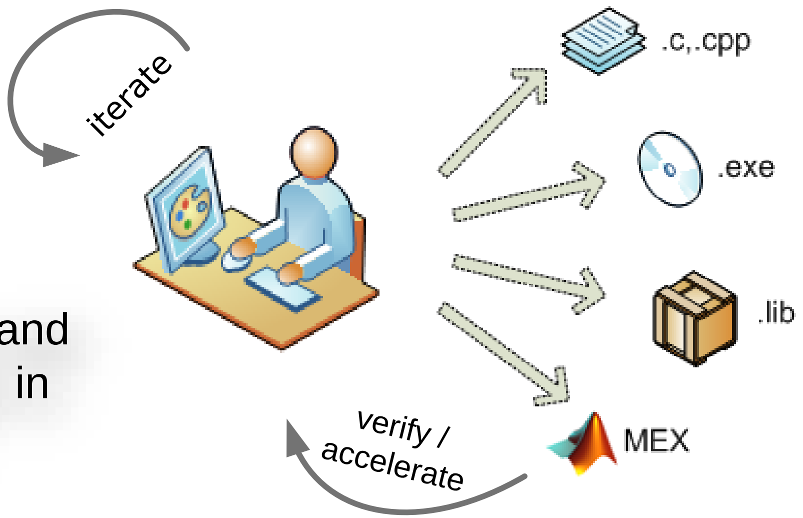
- Manual coding errors

Challenges with Manual Translation
from MATLAB to C



- Time consuming and expensive

Automatic Translation of MATLAB to C



Algorithm Design and Code Generation in MATLAB

With MATLAB Coder, design engineers can

- Maintain one design in MATLAB
- Design faster and get to C quickly
- Test more systematically and frequently
- Spend more time improving algorithms in MATLAB

Acceleration using MEX

- Speed-up factor will vary

- When you **may** see a speedup
  - Often for Communications and Signal Processing
  - Always for Fixed-point
  - Likely for loops with states or when vectorization isn’t possible

- When you **may not** see a speedup
  - MATLAB implicitly multithreads computation
  - Built-functions call IPP or BLAS libraries

Supported MATLAB Language Features and Functions

Matrices and Arrays	Data Types	Programming Constructs	Functions

• Matrix operations
• N-dimensional arrays
• Subscripting
• Frames
• Persistent variables
• Global variables
	• Complex numbers
	• Integer math
	• Double/single-precision
	• Fixed-point arithmetic
	• Characters
	• Structures
	• Numeric class
	• Variable-sized data
	• MATLAB Classes
	• System objects
		• Arithmetic, relational, and logical operators
		• Program control (if, for, while, switch)
			• MATLAB functions and sub-functions
			• Variable length argument lists
			• Function handles
			
			Supported algorithms
			• > 800 MATLAB operators and functions
			• > 200 System objects for
			  • Signal processing
			  • Communications
			  • Computer vision

[Supported Functions]

More Information

- To learn more visit the product page
  - www.mathworks.com/products/matlab-coder

- On-Demand Webinar:

  "MATLAB to C Made Easy"

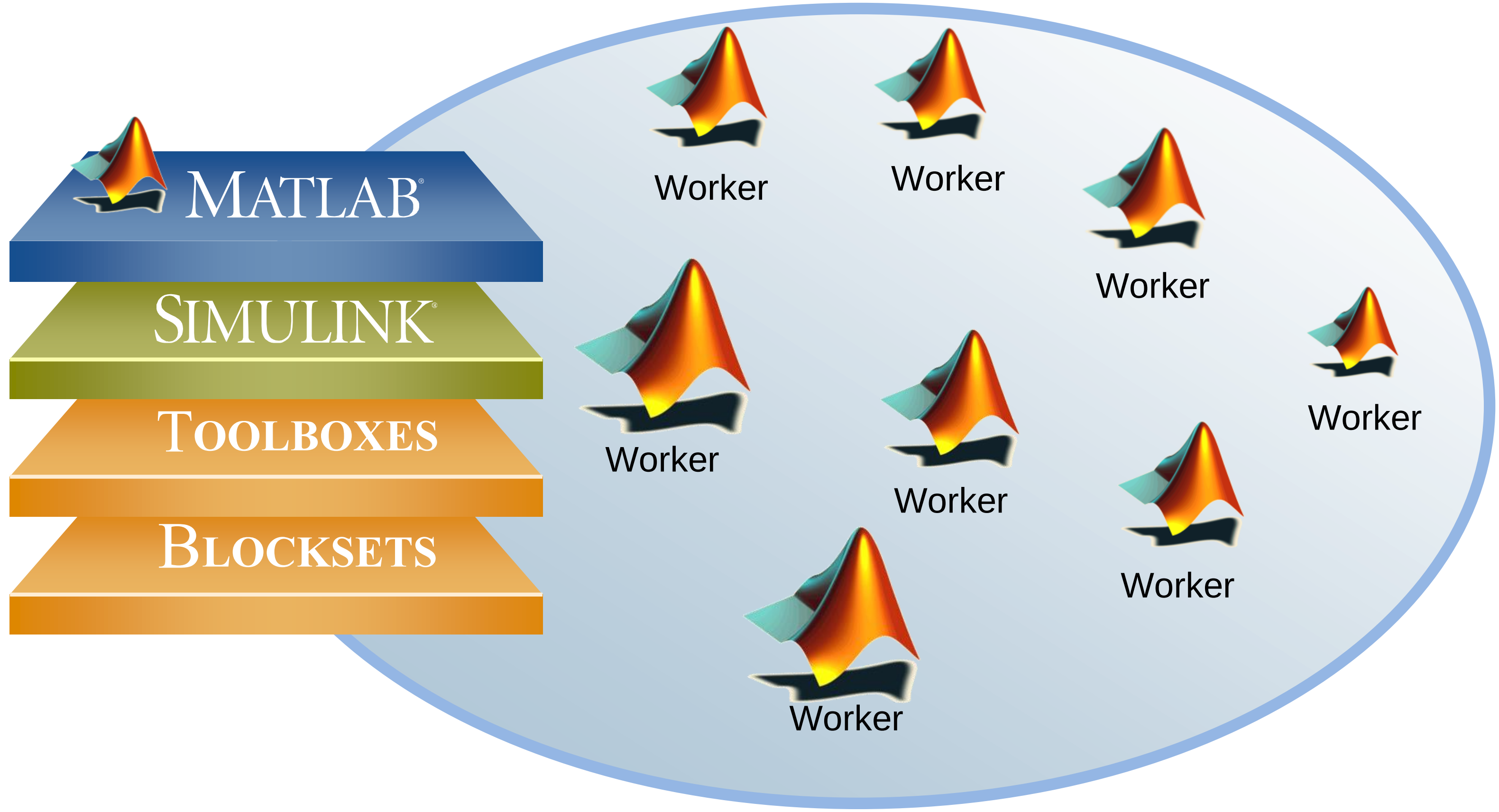
  Search at

  http://www.mathworks.com/company/events/webinars/index.html

Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary

Going Beyond Serial MATLAB Applications



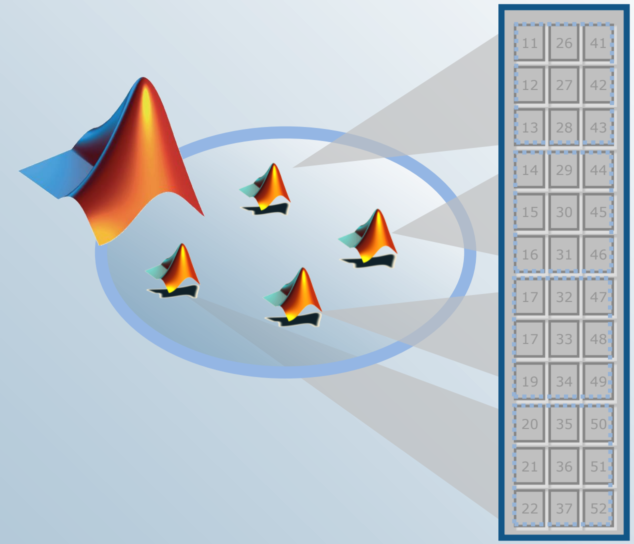
Parallel Computing enables you to …

Larger Compute Pool

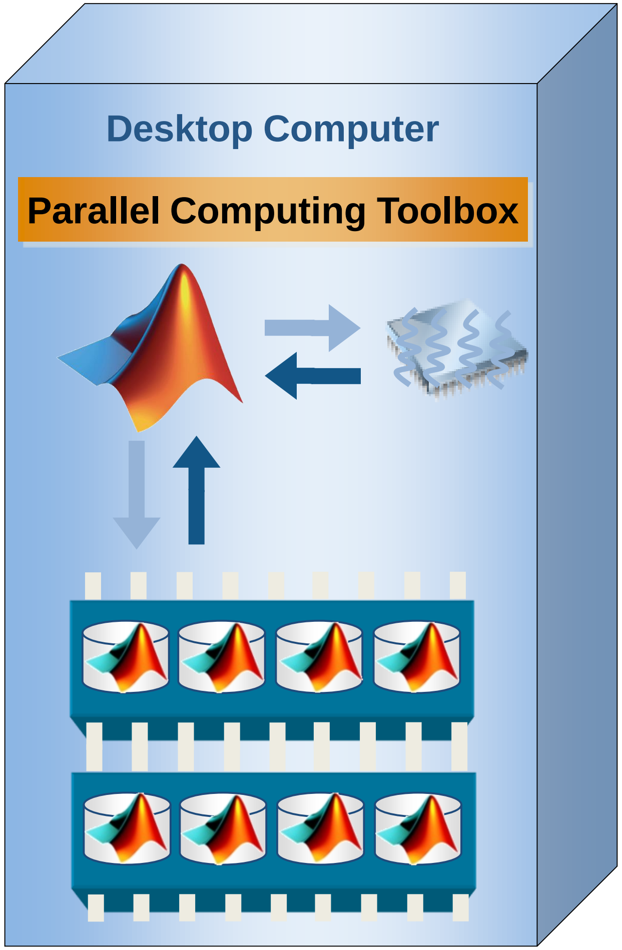
Speed up Computations

Larger Memory Pool

Work with Large Data



Parallel Computing on the Desktop



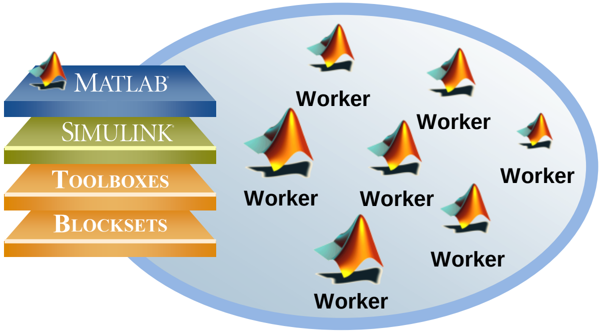
- Speed up parallel applications on local computer
- Take full advantage of desktop power by using CPUs and GPUs
- Separate computer cluster not required

Using Additional Cores/Processors (CPUs)

- Support built into Toolboxes

Tools Providing Parallel Computing Support

- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- Communications System Toolbox
- Simulink Design Optimization
- Bioinformatics Toolbox
- Image Processing Toolbox
- …



Directly leverage functions in Parallel Computing Toolbox

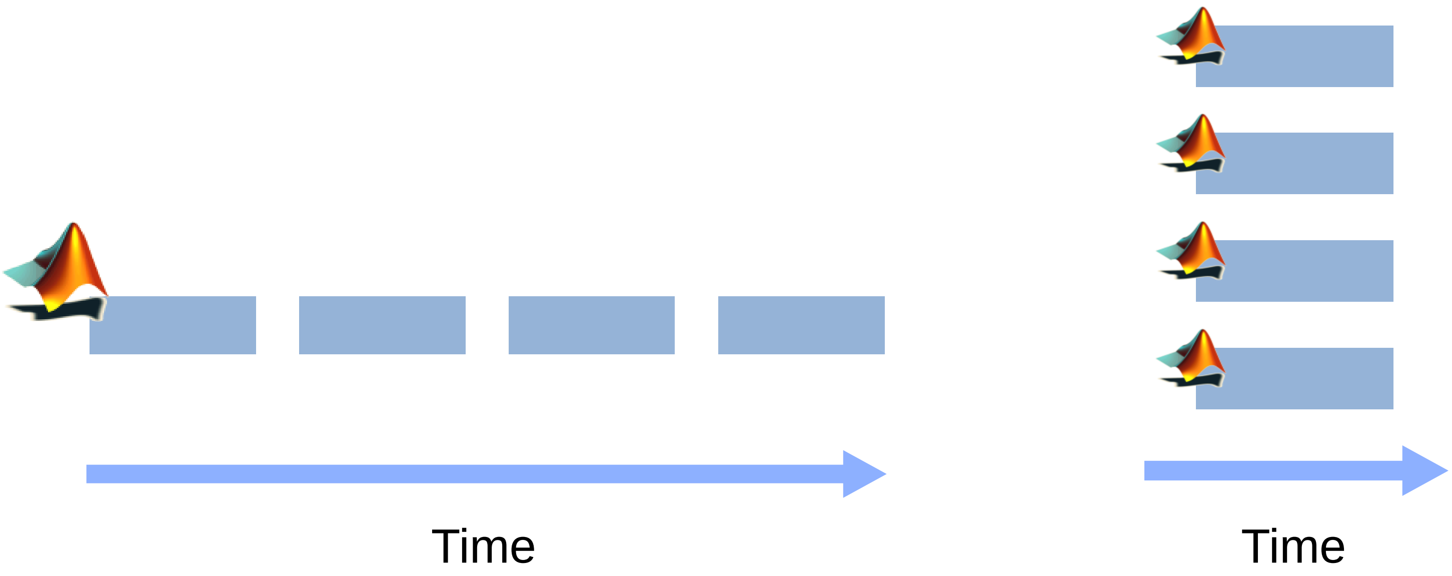
http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html

Using Additional Cores/Processors (CPUs)

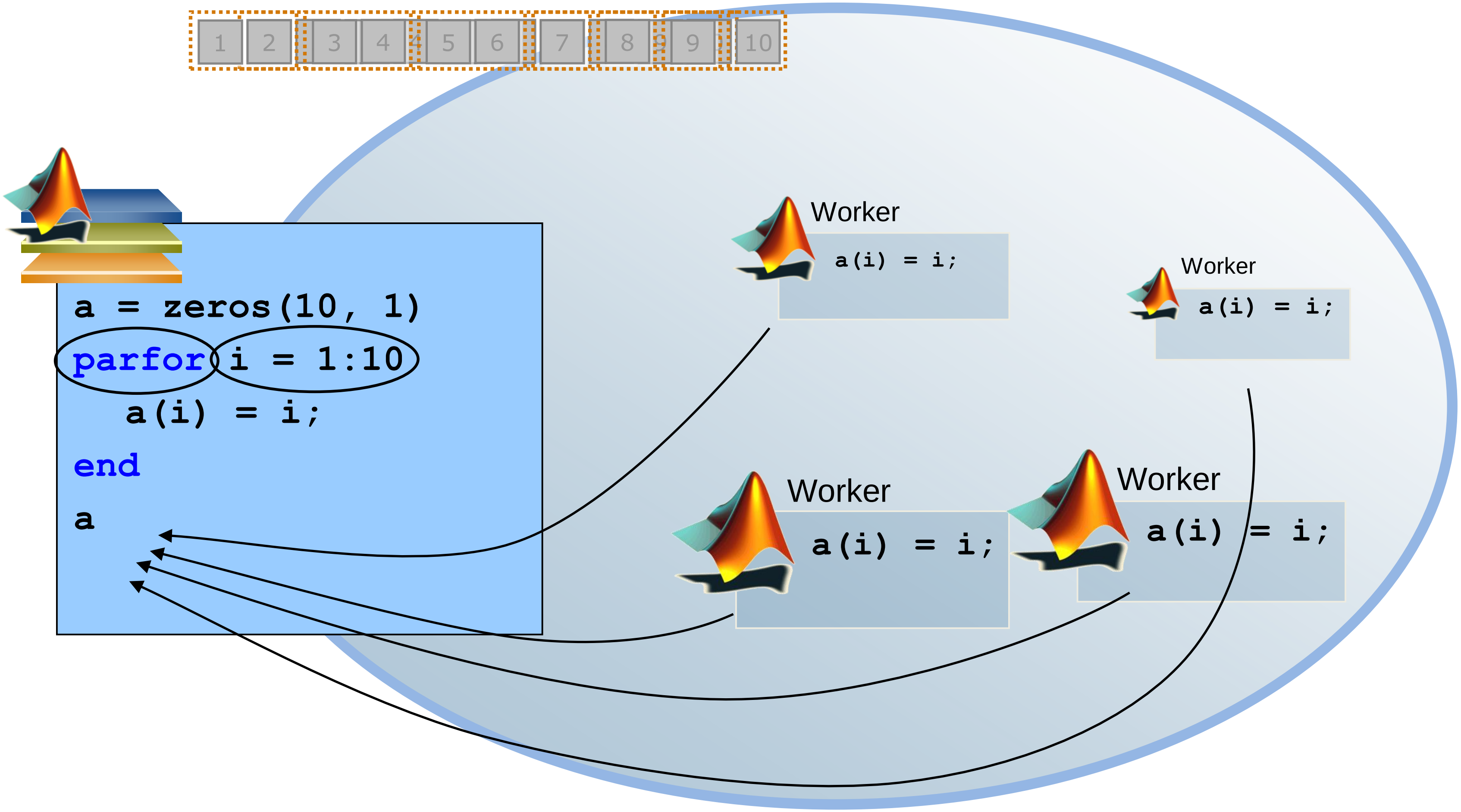
- Support built into Toolboxes
- Simple programming constructs:
  `parfor, batch, distributed`

Running Independent Tasks or Iterations

- Ideal problem for parallel computing
- No dependencies or communications between tasks
- Examples include parameter sweeps and Monte Carlo simulations



The Mechanics of `parfor` Loops

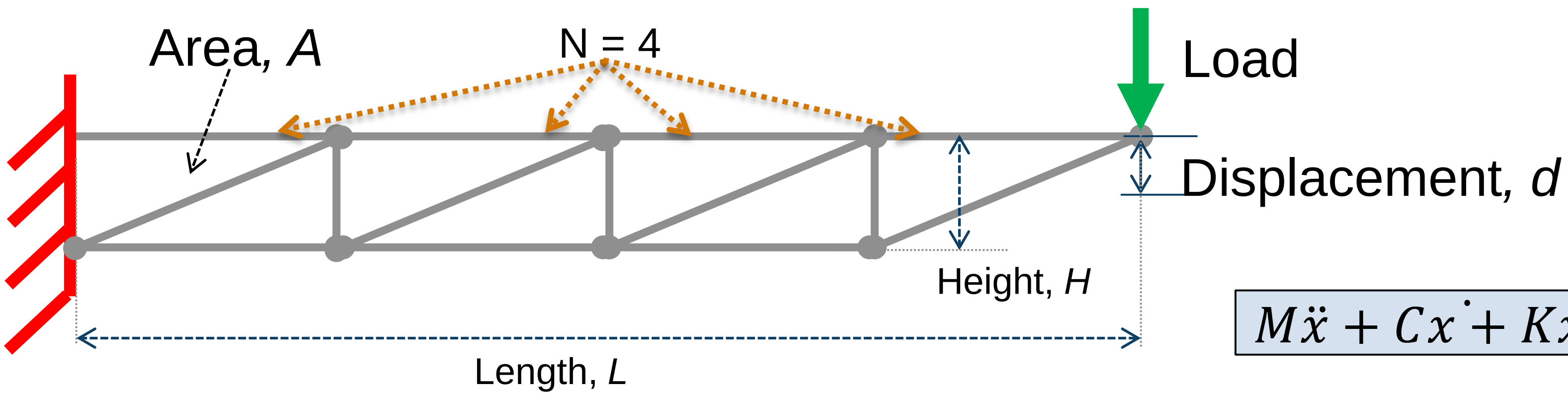


Pool of MATLAB Workers

Example: Parameter Sweep of ODEs

Parallel for-loops

- Parameter sweep of ODEs
  - Deflection of a truss under a dynamic load

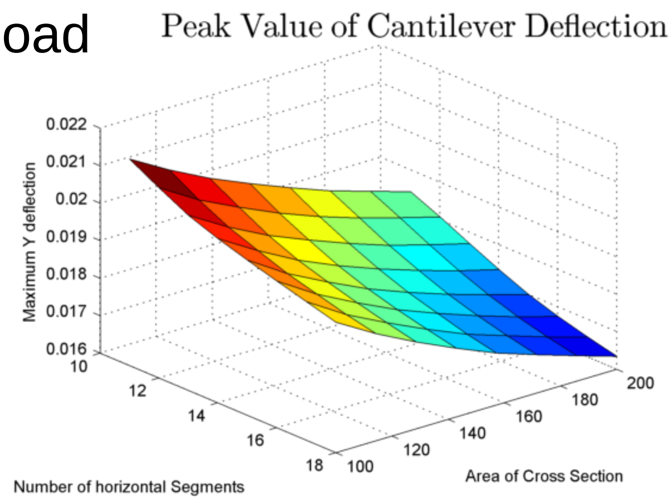


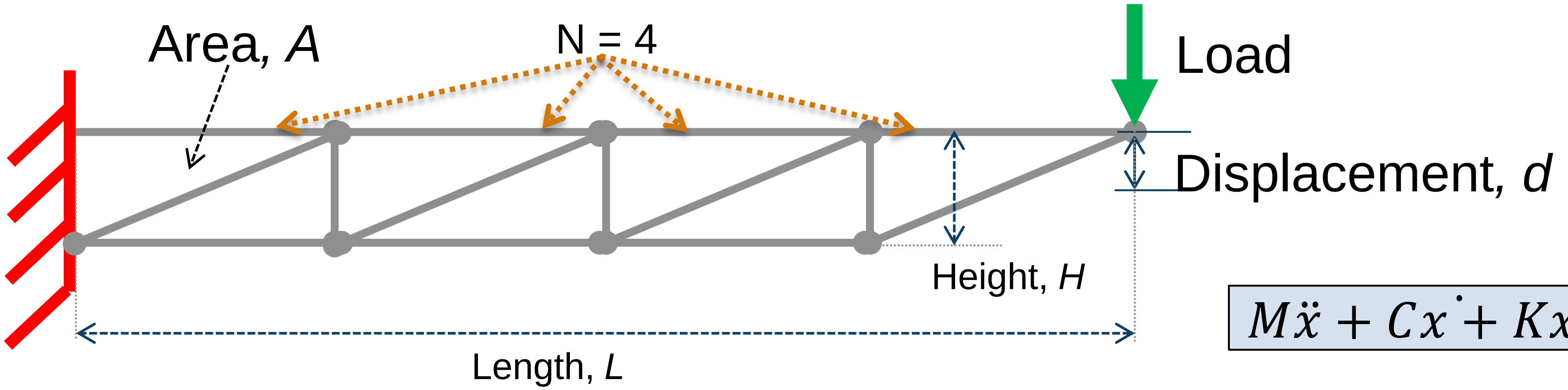
$$M\ddot{x} + C\dot{x} + Kx = F$$

Example: Parameter Sweep of ODEs

Parallel for-loops

- Parameter sweep of ODEs
  - Deflection of a truss under a dynamic load
  - Sweeping two parameters:
    - Number of truss elements
    - Cross sectional area of truss elements





$$M\ddot{x} + C\dot{x} + Kx = F$$

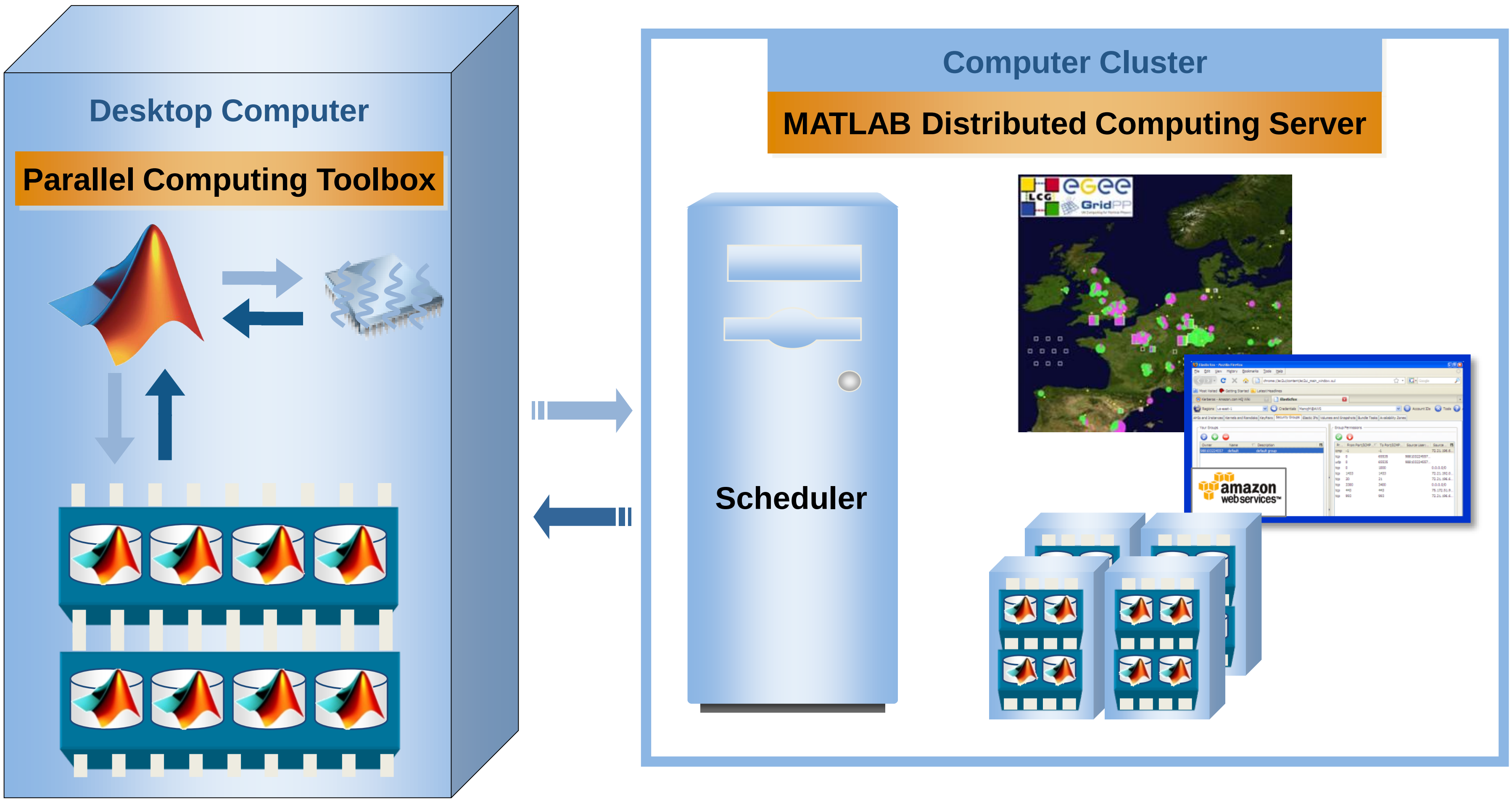
Using Additional Cores/Processors (CPUs)

Ease of Use

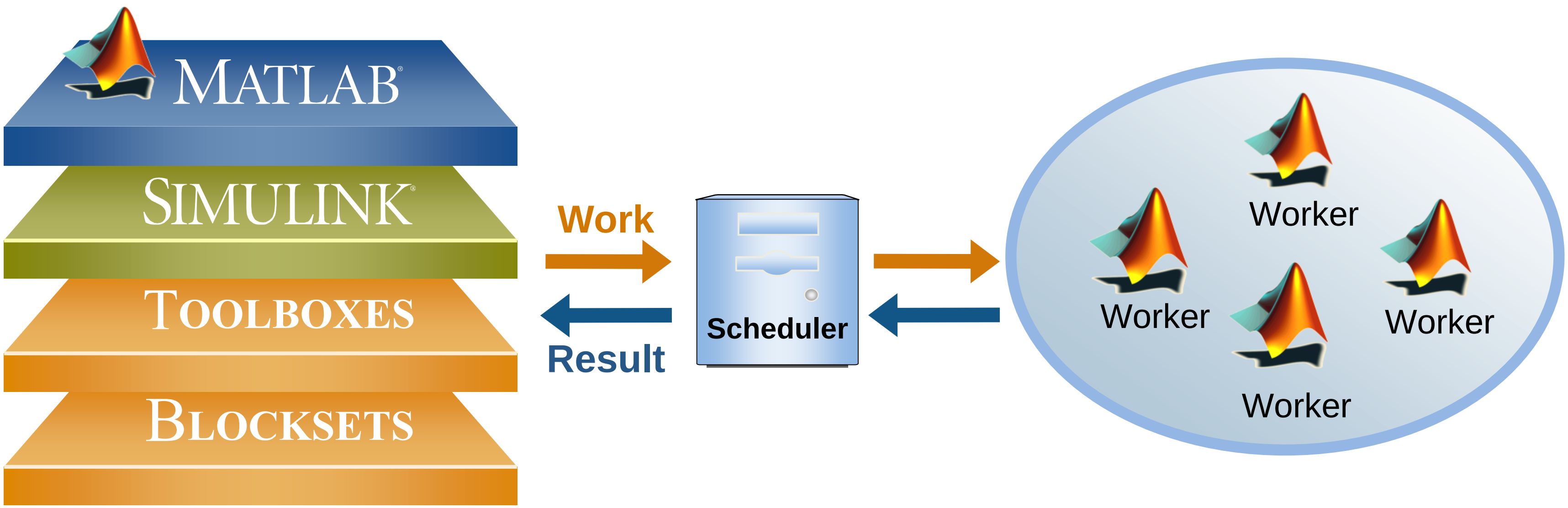
- Support built into Toolboxes
- Simple programming constructs:
  `parfor, batch, distributed`
- Full control of parallelization:
  jobs and tasks, `spmd`

Greater Control

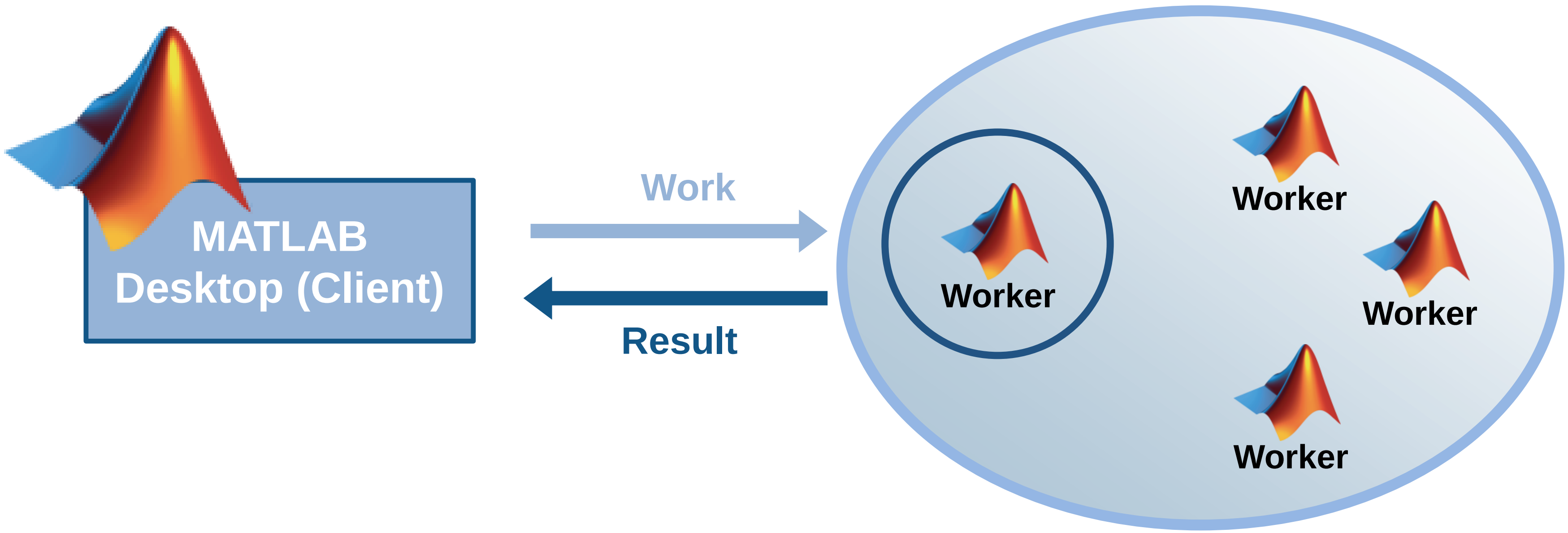
Scale Up to Clusters, Grids and Clouds



Scheduling Work

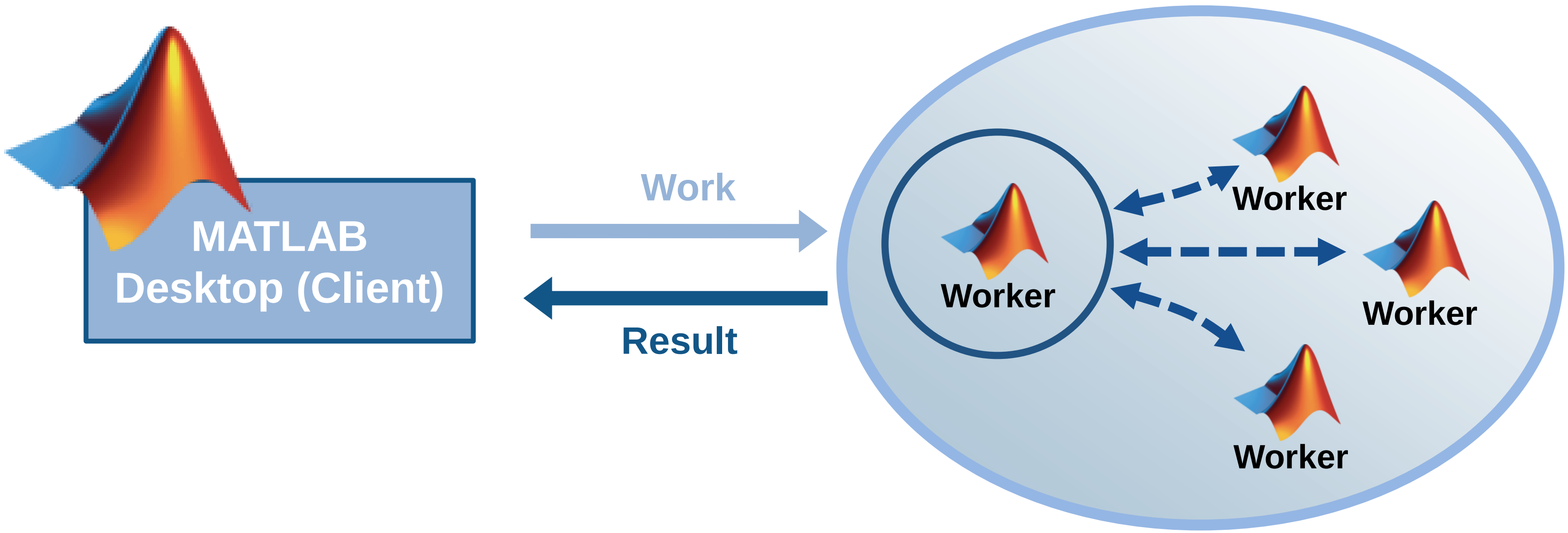


Offload Computations with `batch`



`batch(…)`

Offload and Scale Computations with `batch`

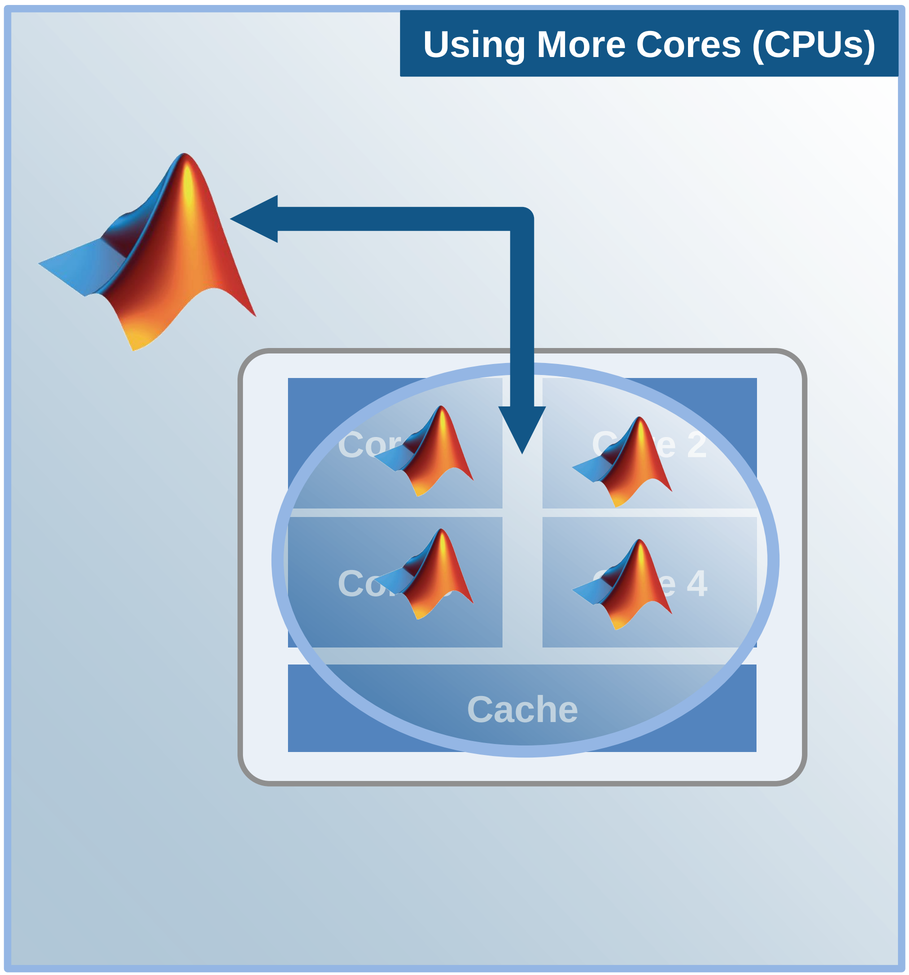


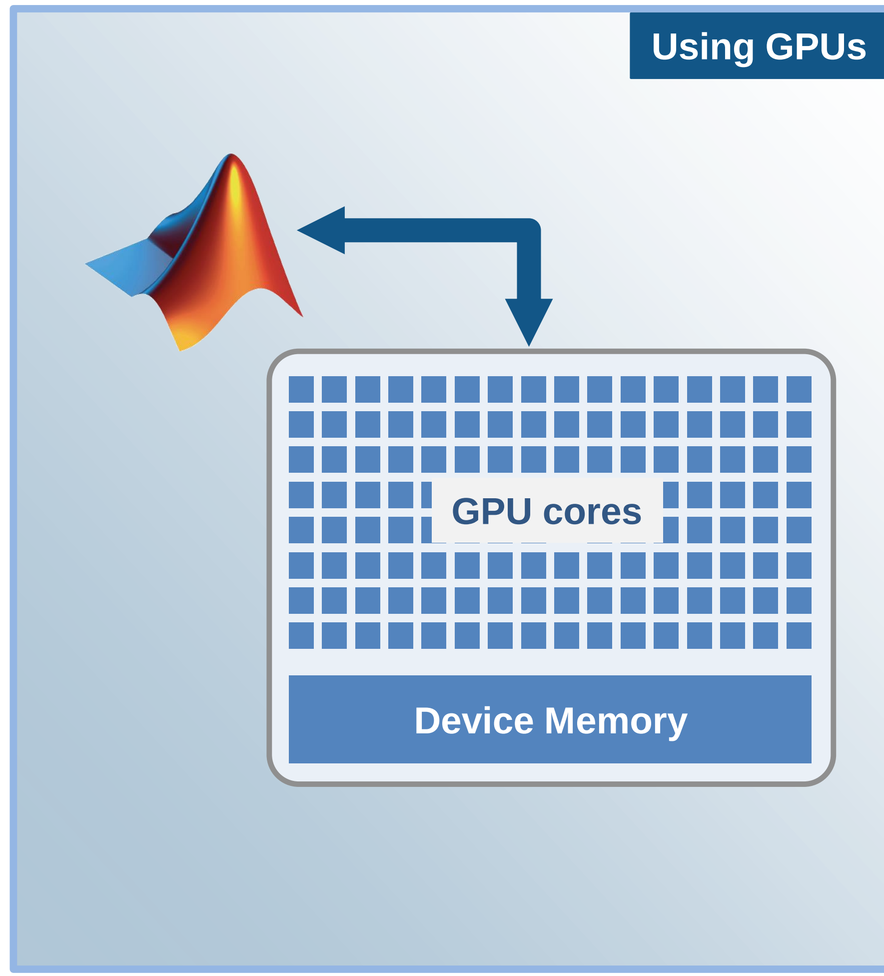
`batch(…,'Pool',…)`

What is a Graphics Processing Unit (GPU)

- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory

Performance Gain with More Hardware





GPU Requirements

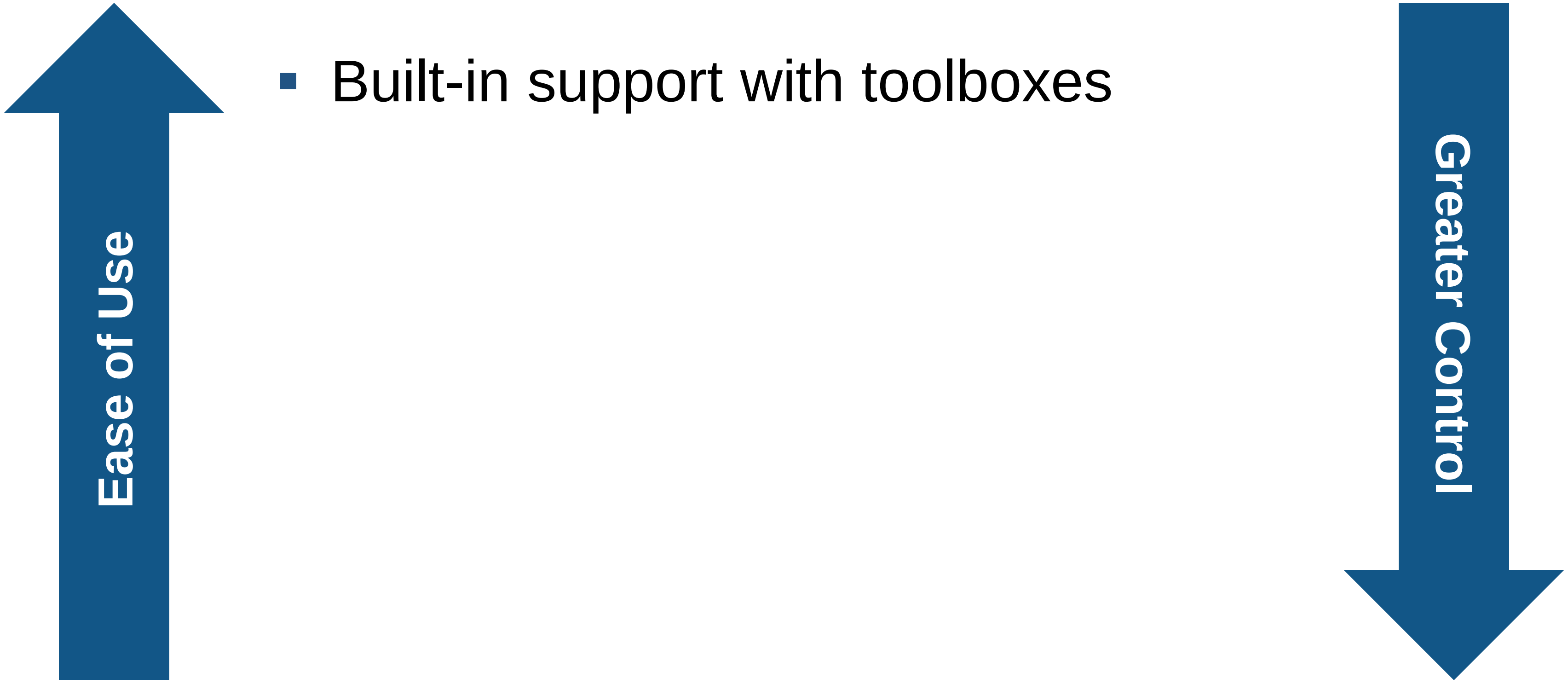
- Parallel Computing Toolbox requires NVIDIA GPUs

- This includes the Tesla 20-series products

MATLAB Release	Required Compute Capability
MATLAB R2014b	2.0 or greater
MATLAB R2014a and earlier releases	1.3 or greater

See a complete listing at www.nvidia.com/object/cuda_gpus.html

Programming Parallel Applications (GPU)



- Built-in support with toolboxes

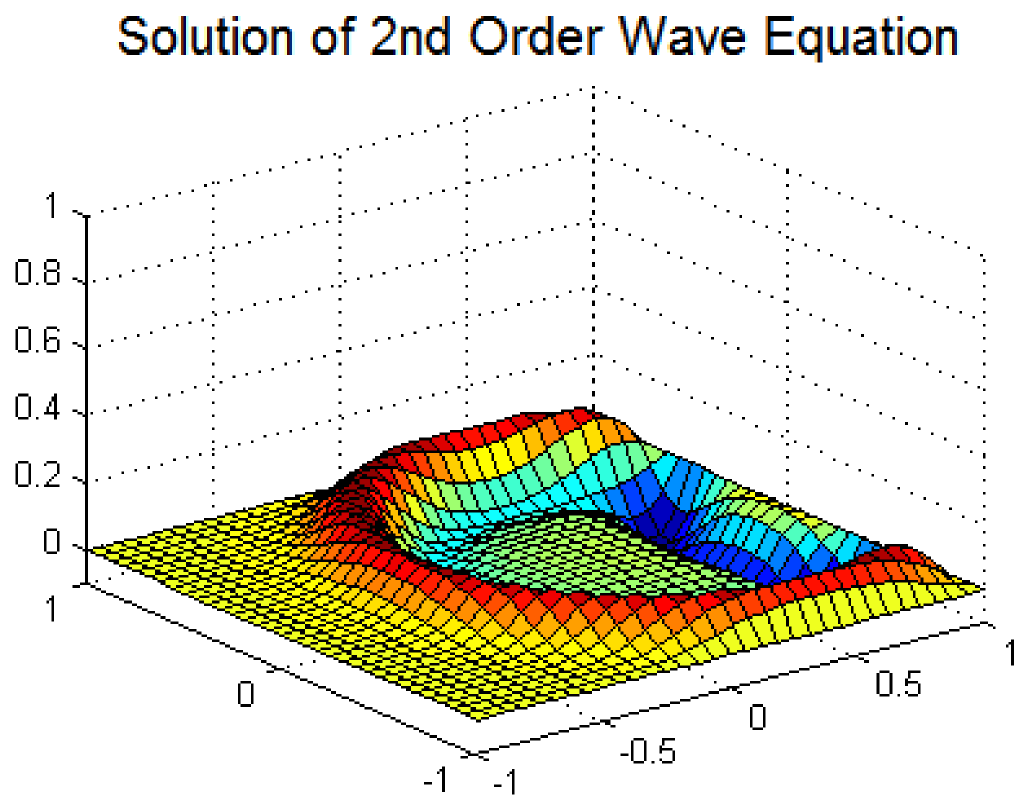
Programming Parallel Applications (GPU)

Ease of Use

- Built-in support with toolboxes
- Simple programming constructs: **`gpuArray, gather`**

Greater Control

Example: Solving 2D Wave Equation

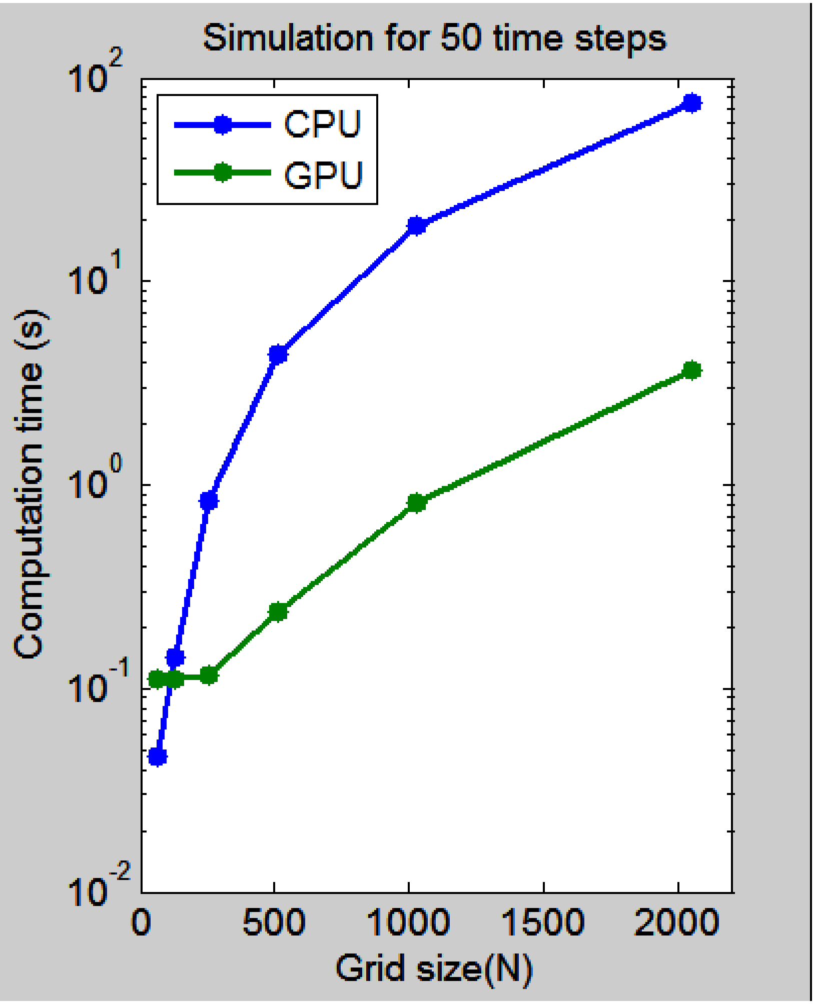


- Solve 2nd order wave equation using spectral methods:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Benchmark: Solving 2D Wave Equation

CPU v. GPU



Intel Xeon Processor W3550 (3.07GHz), NVIDIA Tesla K20c GPU

Programming Parallel Applications (GPU)

Ease of Use

- Built-in support with toolboxes
- Simple programming constructs:
  `gpuArray, gather`
- Advanced programming constructs:
  `arrayfun, spmd`
- Interface for experts:
  `CUDAKernel, MEX support`

Greater Control

www.mathworks.com/help/distcomp/run-cuda-or-ptx-code-on-gpu

www.mathworks.com/help/distcomp/run-mex-functions-containing-cuda-code

Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary

Key Takeaways

- Consider performance benefit of vector and matrix operations in MATLAB
- Analyze your code for bottlenecks and address most critical items
- Leverage MATLAB Coder to speed up applications through generated C/C++ code
- Leverage parallel computing tools to take advantage of additional computing resources

Sample of Other Performance Resources

- MATLAB documentation

  MATLAB → Advanced Software Development → Performance and Memory

- Accelerating MATLAB Algorithms and Applications

  http://www.mathworks.com/company/newsletters/articles/accelerating-matlab-algorithms-and-applications.html

- The Art of MATLAB, Loren Shure's blog

  blogs.mathworks.com/loren/

- MATLAB Answers

  http://www.mathworks.com/matlabcentral/answers/