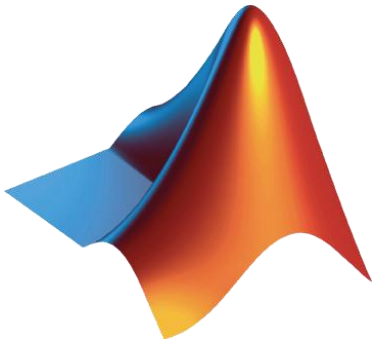
Speeding up MATLAB Applications

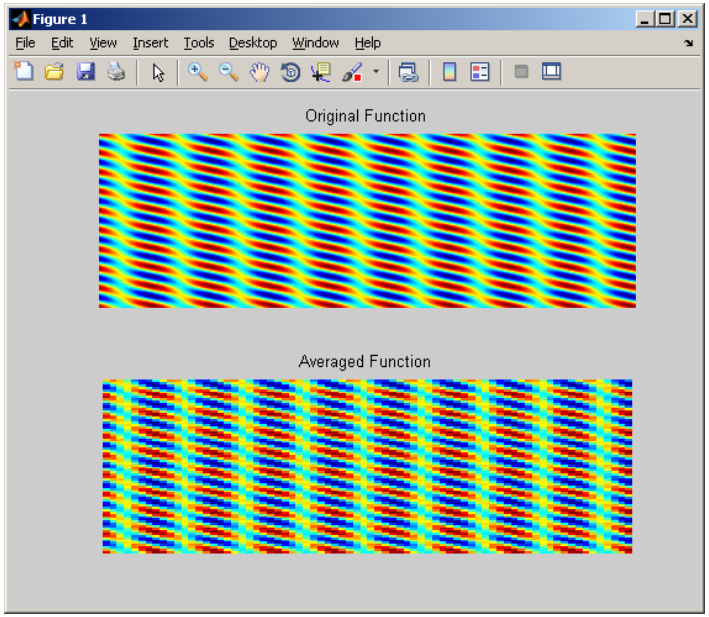


Agenda

- Leveraging the power of vector & matrix operations
- Addressing bottlenecks
- 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



Summary of Example

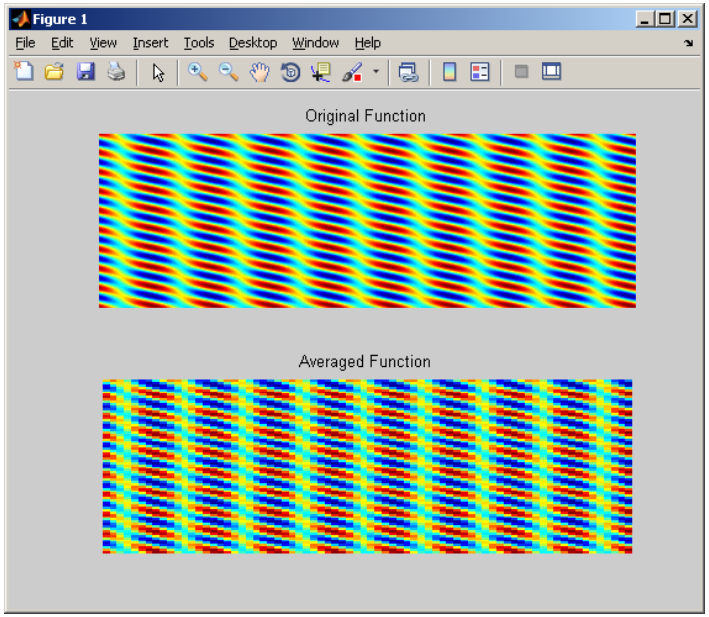
- Used built-in timing functions

```
>> tic
>> toc
```

- Used Code Analyzer to find suboptimal code

- Preallocated arrays

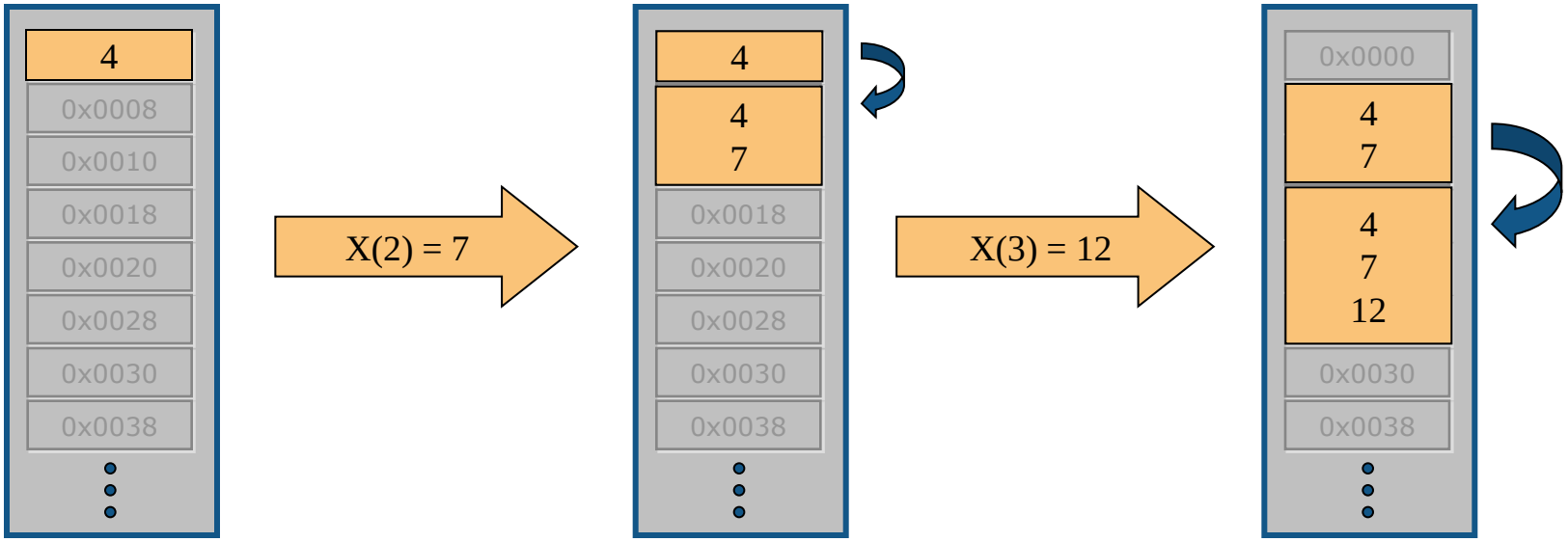
- Vectorized code



Effect of Not Preallocating Memory

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

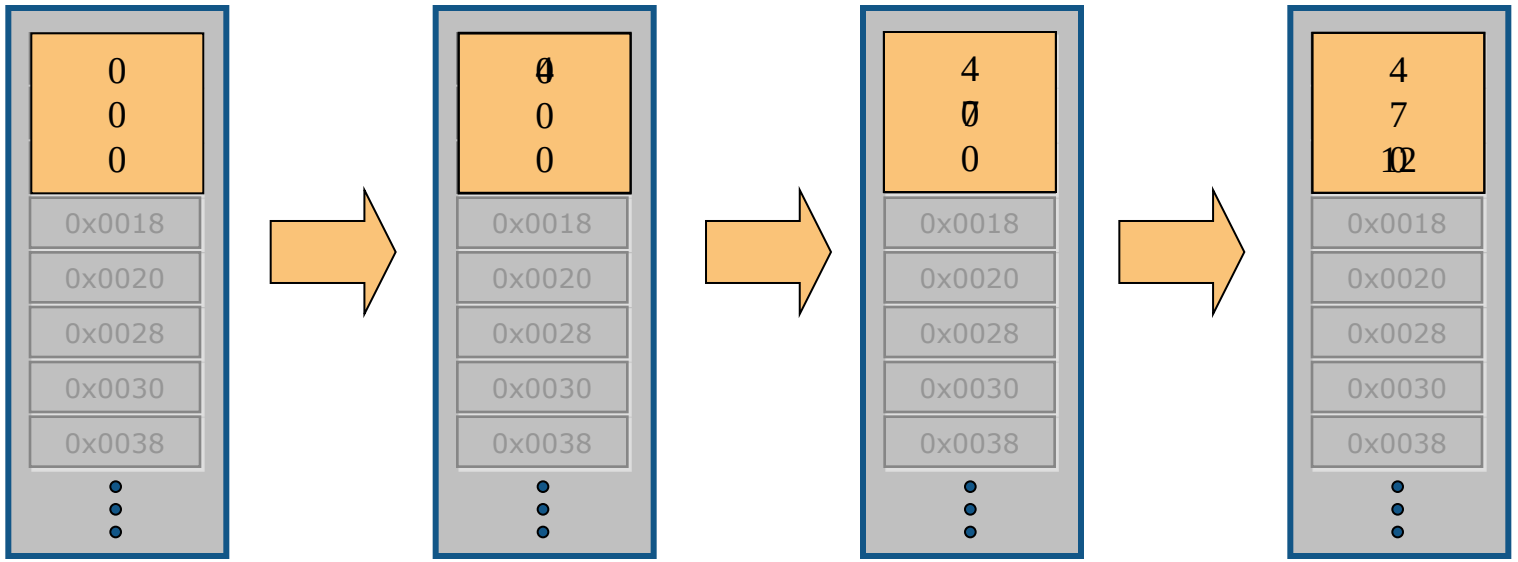
Resizing Arrays is Expensive



Benefit of Preallocation

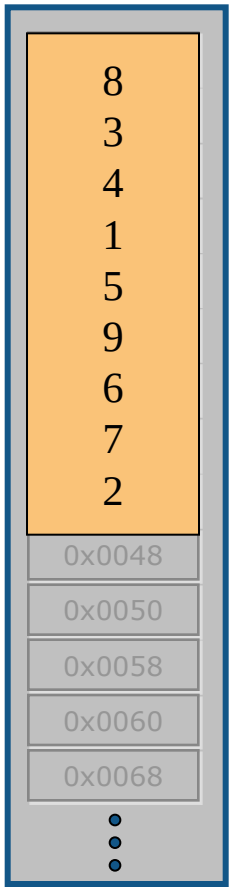
```
>> x = zeros(3,1)
>> x(1) = 4
>> x(2) = 7
>> x(3) = 12
```

Reduced Memory Operations



Data Storage of MATLAB Arrays

```
>> x = magic(3)
x =
     8     1     6
     3     5     7
     4     9     2
```



Column-Major Memory Storage

See the June 2007 article in "The MathWorks News and Notes":
http://www.mathworks.com/company/newsletters/news_notes/june07/patterns.html

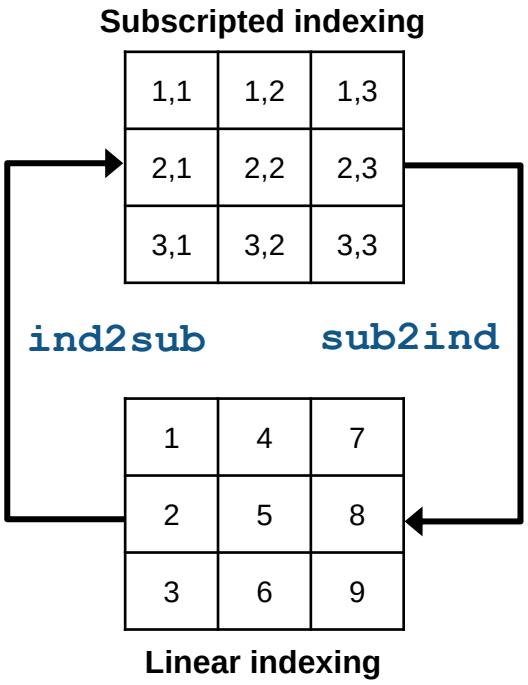
Speed and Memory Usage

- Balance vectorization and memory usage
  - Use **`bsxfun`** instead of functions such as **`repmat`**
  - Reduce size of arrays to smaller blocks for block processing

- Consider using sparse matrices
  - *Less Memory*: Store only nonzero elements and their indices
  - *Faster*: Eliminate operations on zero elements
  - Blog Post - Creating Sparse Finite Element Matrices http://blogs.mathworks.com/loren/2007/03/01/creating-sparse-finite-element-matrices-in-matlab/

Indexing into MATLAB Arrays

- Subscripted
  - Access elements by rows and columns
- Linear
  - Access elements with a single number
- Logical
  - Access elements with logical operations or mask



MATLAB Underlying Technologies

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

BLAS and LAPACK require contiguous arrays

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

Other Best Practices

- Minimize dynamically changing path

```
>> addpath(…)
>> fullfile(…)
```

instead of `cd(…)`

- Use the functional load syntax

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

instead of `load('myvars.mat')`

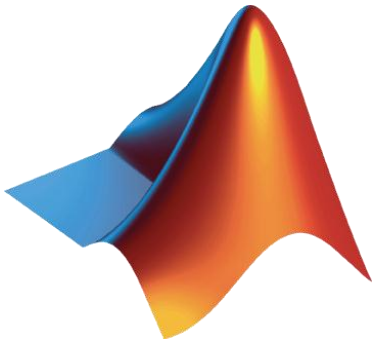
- Minimize changing variable class

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

instead of `x = 'hello';`

Summary

- Techniques for addressing performance
  - Vectorization
  - Preallocation
- Consider readability and maintainability
  - Looping vs. matrix operations
  - Subscripted vs. linear vs. logical
  - etc.

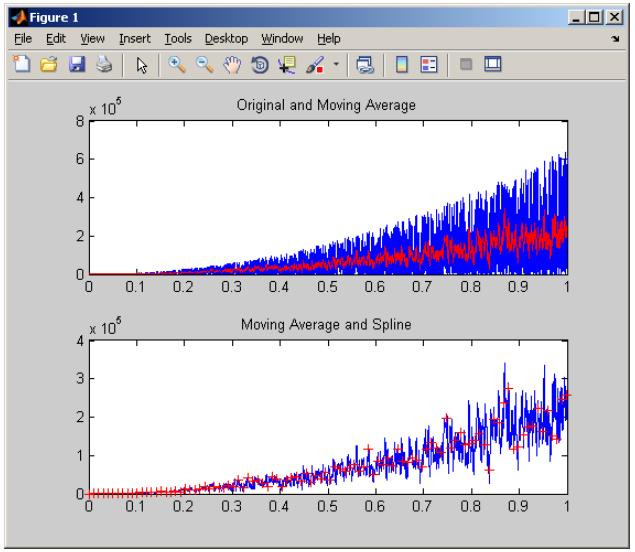


Agenda

- Leveraging the power of vector & matrix operations
- Addressing bottlenecks
- Utilizing additional processing power
- Summary

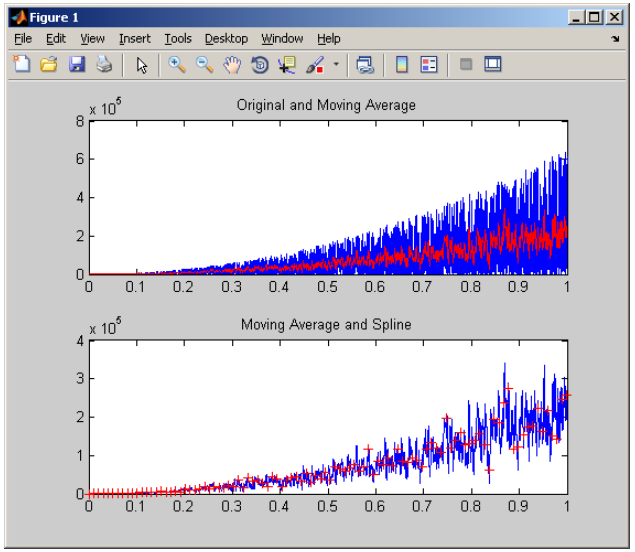
Example: Fitting Data

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



Summary of Example

- Used profiler to analyze code
- Targeted significant bottlenecks
- Reduced file I/O
- Reused figure



Interpreting Profiler Results

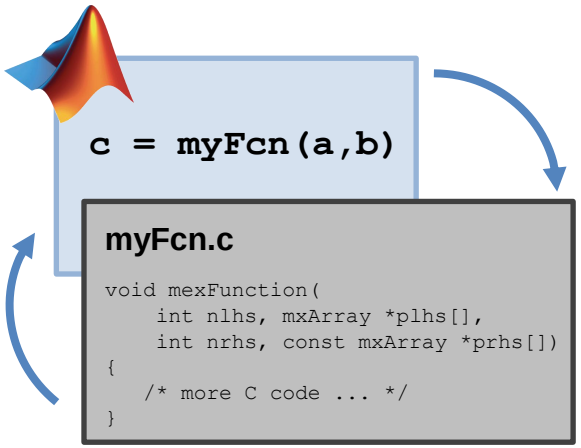
- Focus on top bottleneck
  - Total number of function calls
  - Time per function call
- Functions
  - All function calls have overhead
  - MATLAB functions often take vectors or matrices as inputs
  - Find the right function – performance may vary
    - Search MATLAB functions (e.g., `textscan` vs. `textread`)
    - Write a custom function (specific/dedicated functions may be faster)
    - Many shipping functions have viewable source code

Classes of Bottlenecks

- File I/O
  - Disk is slow compared to RAM
  - When possible, use `load` and `save` commands
- Displaying output
  - Creating new figures is expensive
  - Writing to command window is slow
- Computationally intensive
  - Use what you've learned today
  - Trade-off modularization, readability and performance
  - Integrate other languages or additional hardware
    - e.g. MEX, GGPUs, FPGAs, clusters, etc.

Acceleration using MEX (MATLAB Executable)

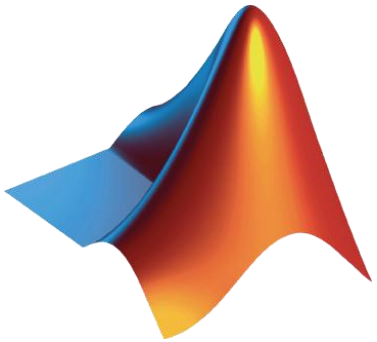
- Call C or Fortran code directly from MATLAB
  - Integrate existing code using MEX API
  - Auto-generate C-based MEX files from MATLAB code using MATLAB Coder



- Speed-up factor will vary
  - May see speedup for state-based for-loops
  - May not see a speedup when MATLAB code is
    - Using multithreaded computations
    - Using optimized libraries (BLAS, FFTW, etc.)

Steps for Improving Performance

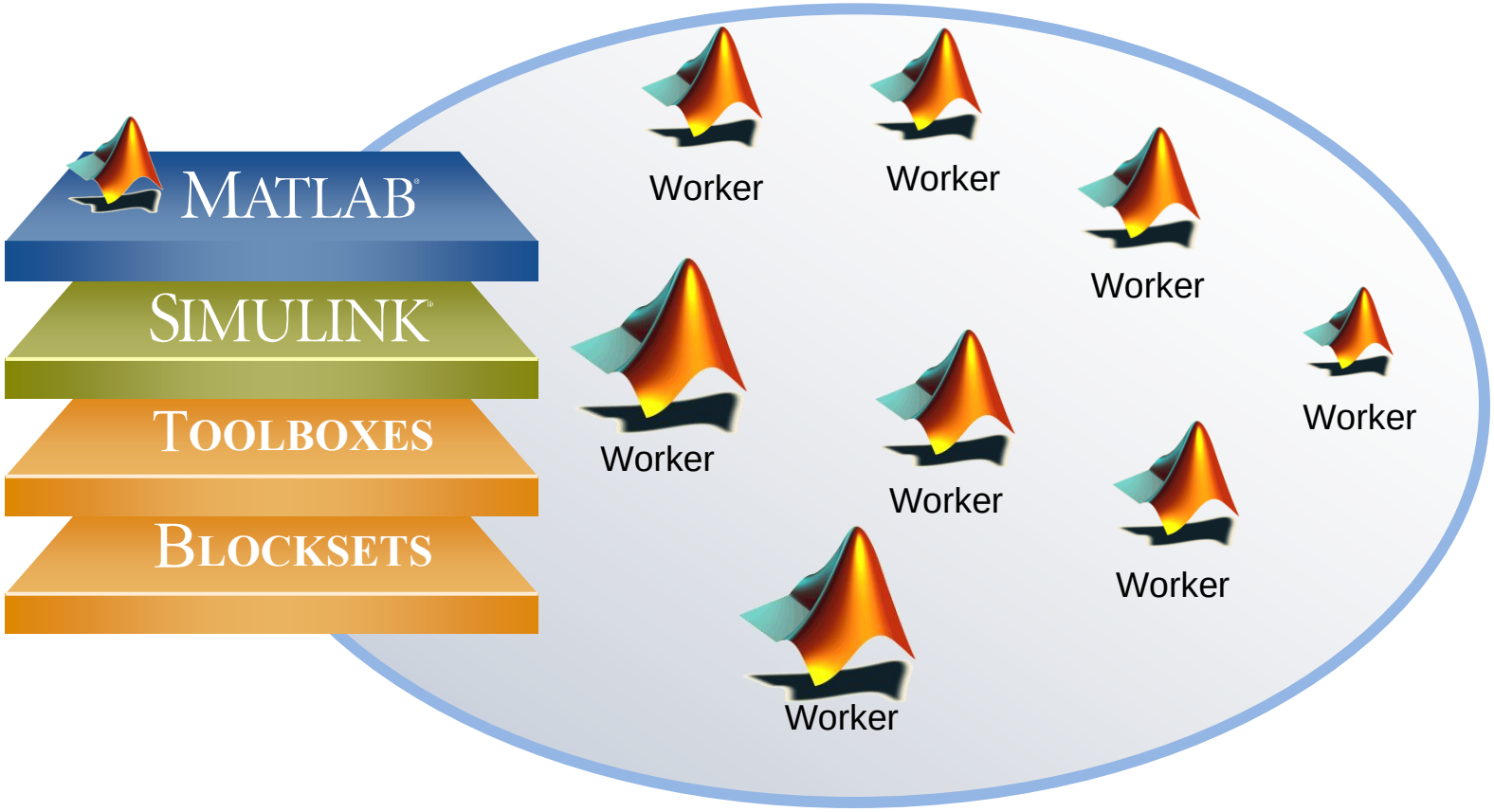
- First focus on getting your code working
- Then speed up the code within core MATLAB
- Consider additional processing power



Agenda

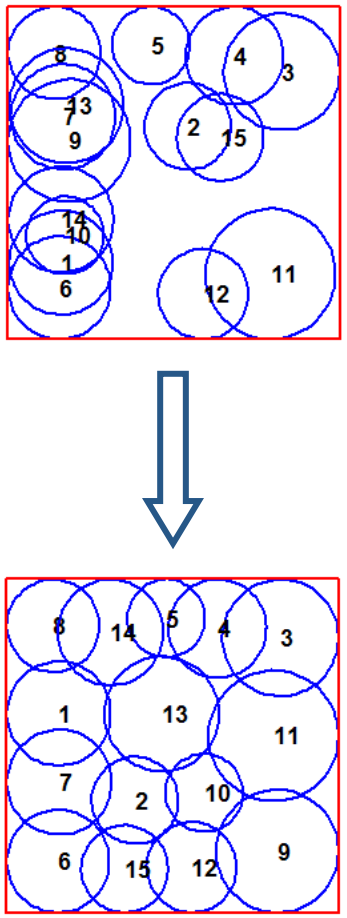
- Leveraging the power of vector & matrix operations
- Addressing bottlenecks
- Utilizing additional processing power
- Summary

Going Beyond Serial MATLAB Applications



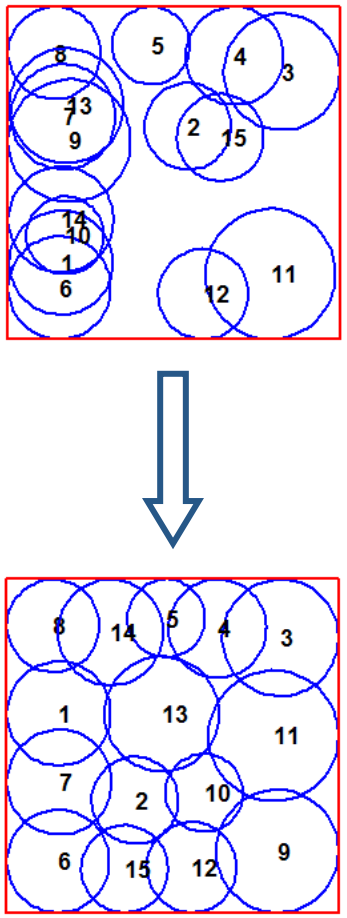
Example: Optimizing Tower Placement

- Determine location of cell towers
- Maximize coverage
- Minimize overlap



Summary of Example

- Enabled built-in support for Parallel Computing Toolbox in Optimization Toolbox
- Used a pool of MATLAB workers
- Optimized in parallel using `fmincon`



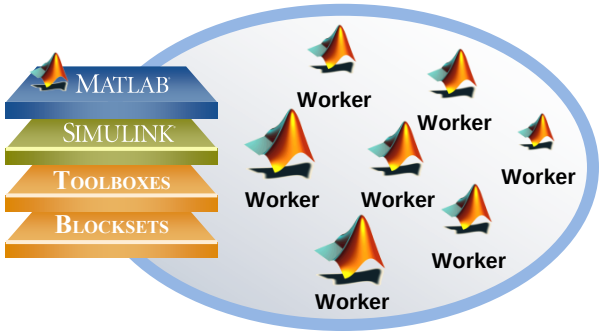
Parallel Computing Support in Optimization Toolbox

- Functions:
  - `fmincon`
    - Finds a constrained minimum of a function of several variables
  - `fminimax`
    - Finds a minimax solution of a function of several variables
  - `fgoalattain`
    - Solves the multiobjective goal attainment optimization problem

- Functions can take finite differences in parallel in order to speed the estimation of gradients

Tools Providing Parallel Computing Support

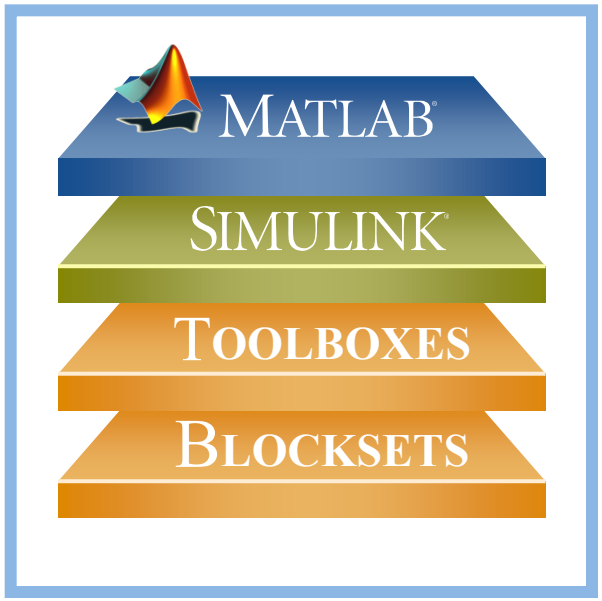
- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- Communications System Toolbox
- Simulink Design Optimization
- Bioinformatics Toolbox
- Image Processing Toolbox
- …

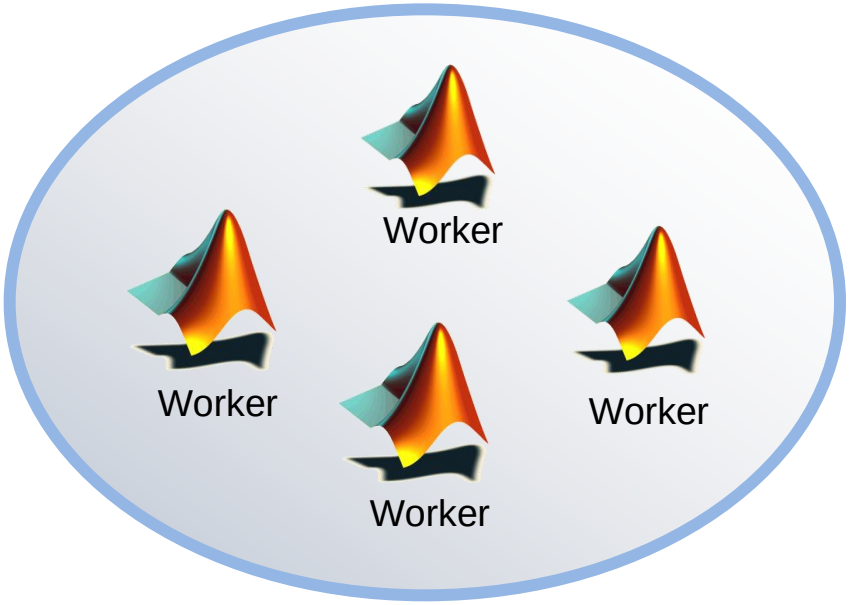


Directly leverage functions in Parallel Computing Toolbox

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

Task Parallel Applications

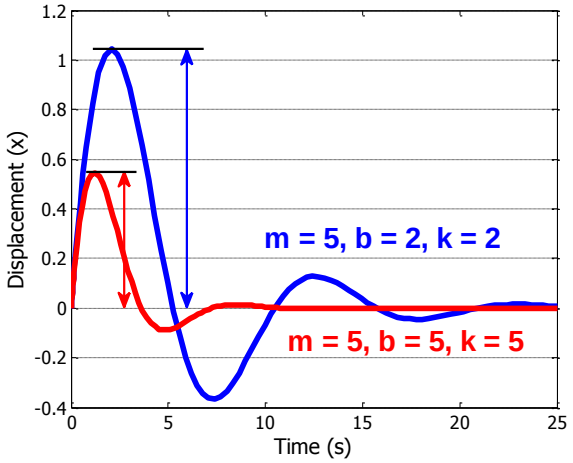




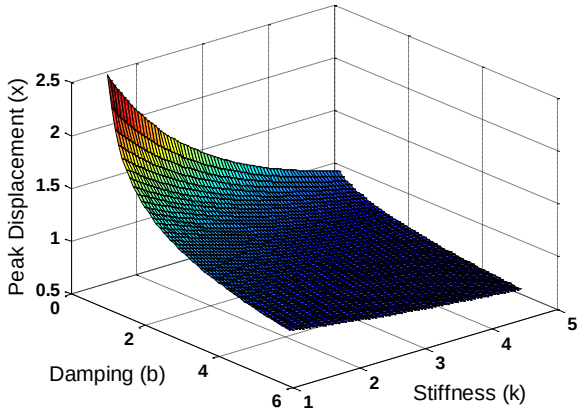
Example: Parameter Sweep of ODEs

- Solve a 2nd order ODE

$$\overbrace{m}^{5}\ddot{x} + \underbrace{b}_{1,2,\ldots}\dot{x} + \underbrace{k}_{1,2,\ldots}x = 0$$

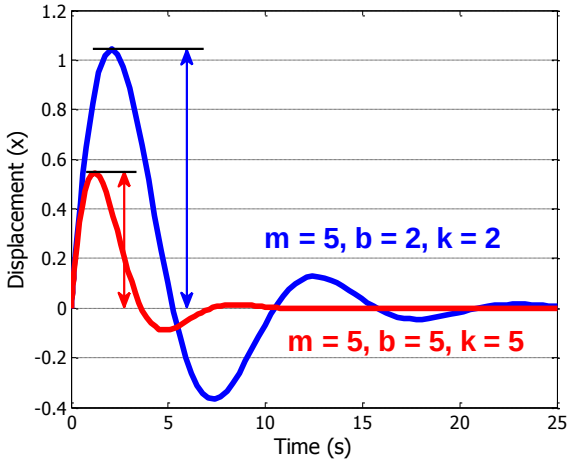


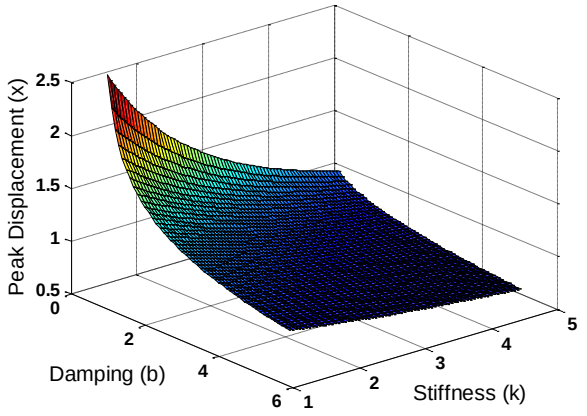
- Simulate with different values for ***b*** and ***k***
- Record peak value for each run
- Plot results



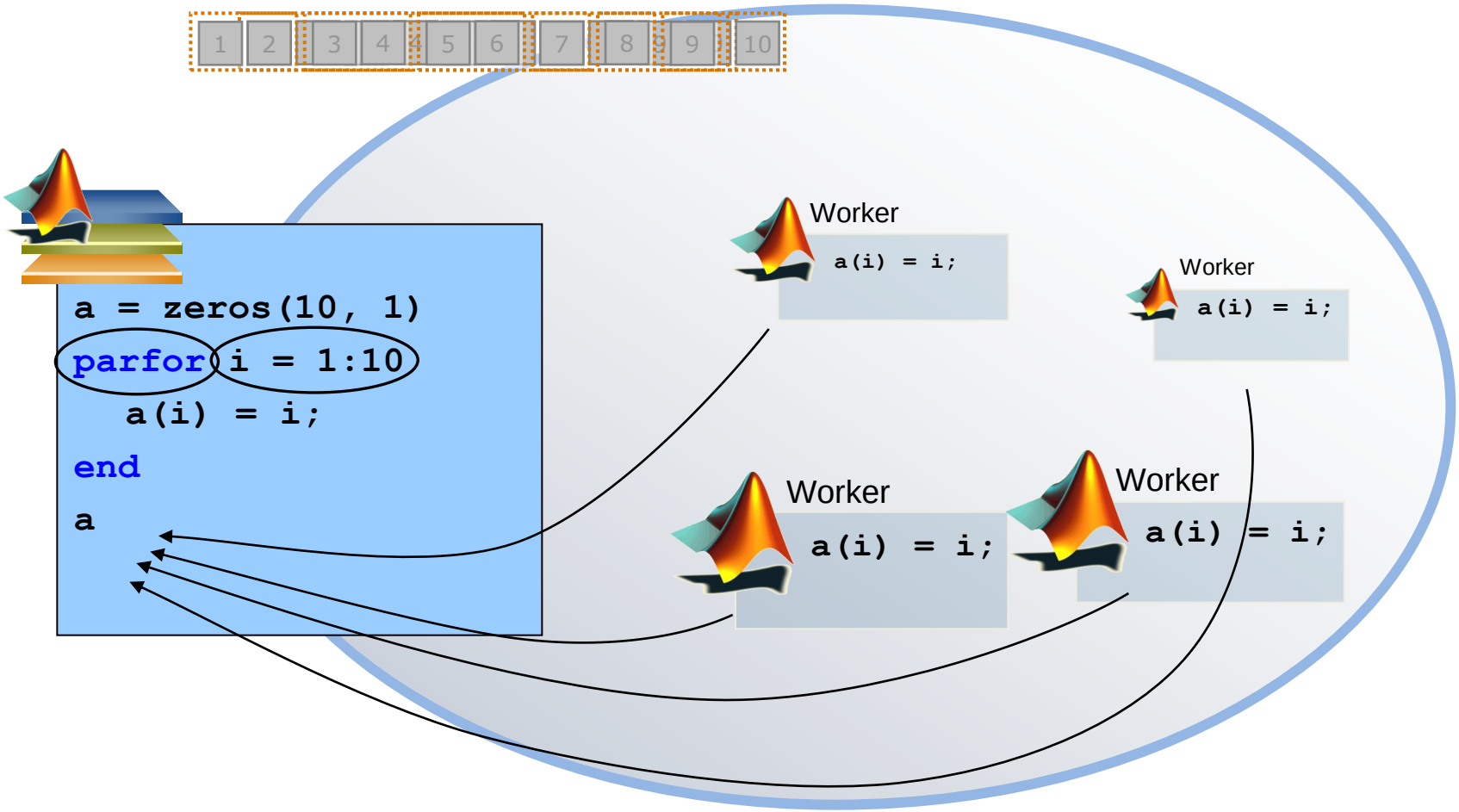
Summary of Example

- Mixed task-parallel and serial code in the same function
- Ran loops on a pool of MATLAB resources
- Used M-Lint analysis to help in converting existing `for`-loop into `parfor`-loop





The Mechanics of `parfor` Loops



Pool of MATLAB Workers

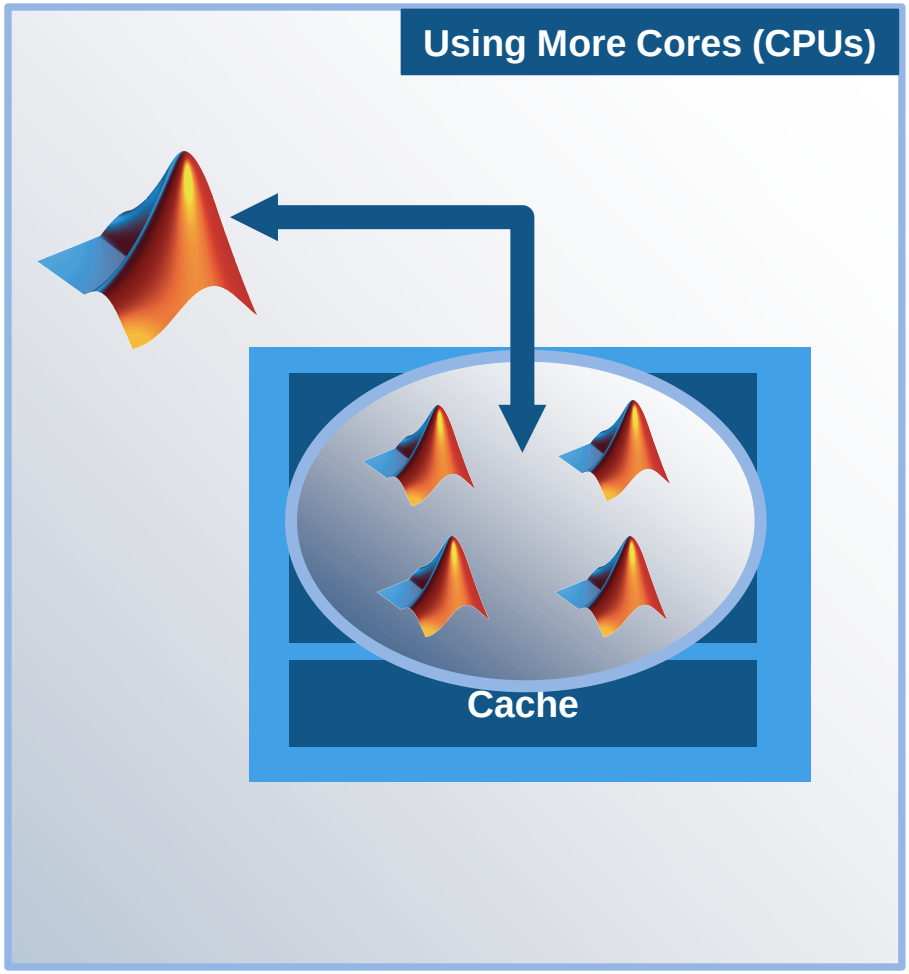
Converting `for` to `parfor`

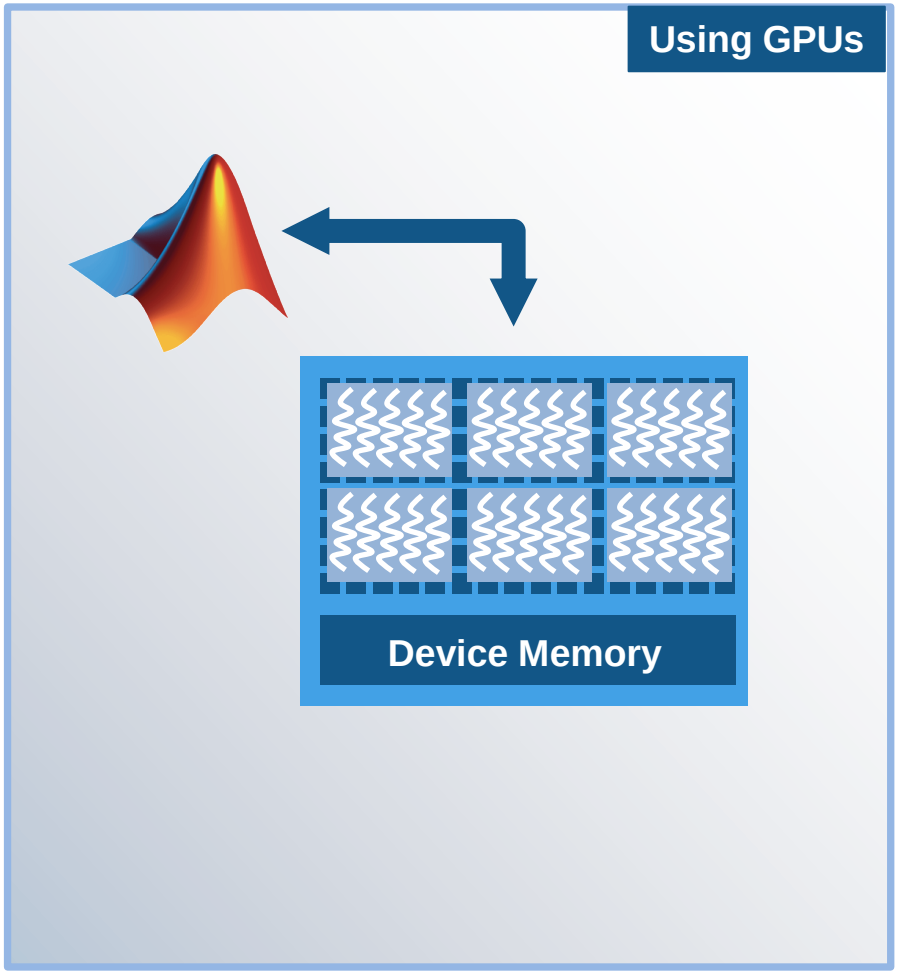
- Requirements for `parfor` loops
  - Task independent
  - Order independent
- Constraints on the loop body
  - Cannot “introduce” variables (e.g. `load`, etc.)
  - Cannot contain `break` or `return` statements
  - Cannot contain another `parfor` loop

Advice for Converting `for` to `parfor`

- Use Code Analyzer to diagnose `parfor` issues
- If your `for` loop cannot be converted to a `parfor`, consider wrapping a subset of the body to a function
- Read the section in the documentation on classification of variables

Performance Gain with More Hardware





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

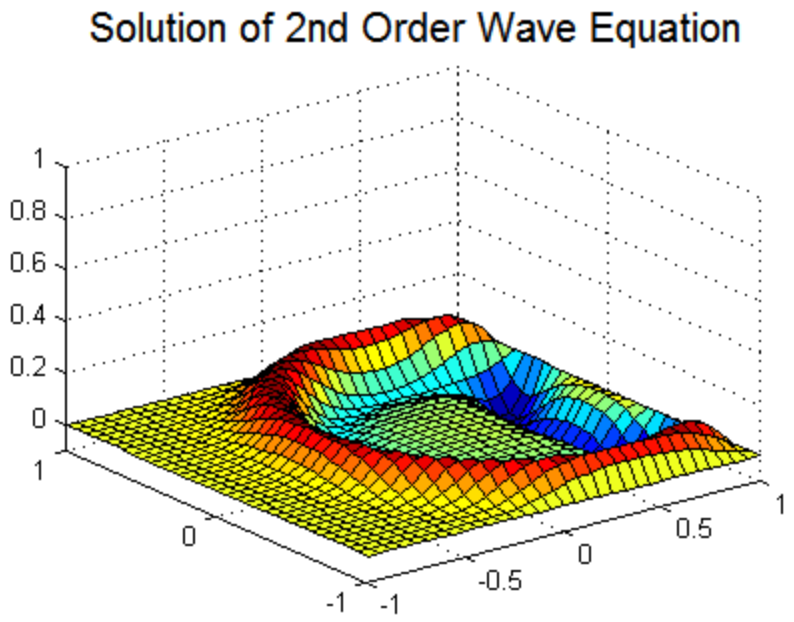
* Parallel Computing Toolbox requires NVIDIA GPUs with Compute Capability 1.3 or greater, including NVIDIA Tesla 10-series and 20-series products. See http://www.nvidia.com/object/cuda_gpus.html for a complete listing

Example: Solving 2D Wave Equation

- Solve 2nd order wave equation using spectral methods:

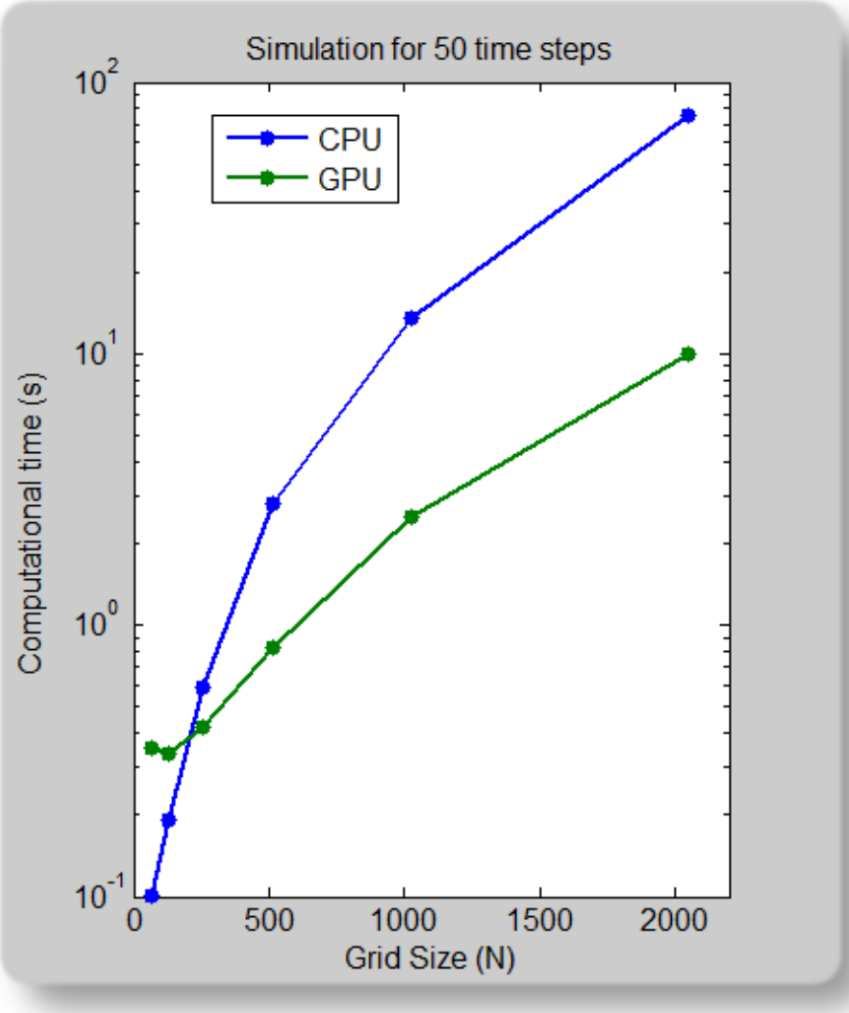
$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

- Run for 50 time steps on both CPU and GPU
- Using **`gpuArray`** and overloaded functions



Benchmark: Solving 2D Wave Equation

CPU vs GPU



Grid Size	CPU (s)	GPU (s)	Speedup
64 x 64	0.1004	0.3553	0.28
128 x 128	0.1931	0.3368	0.57
256 x 256	0.5888	0.4217	1.4
512 x 512	2.8163	0.8243	3.4
1024 x 1024	13.4797	2.4979	5.4
2048 x 2048	74.9904	9.9567	7.5

Intel Xeon Processor X5650, NVIDIA Tesla C2050 GPU

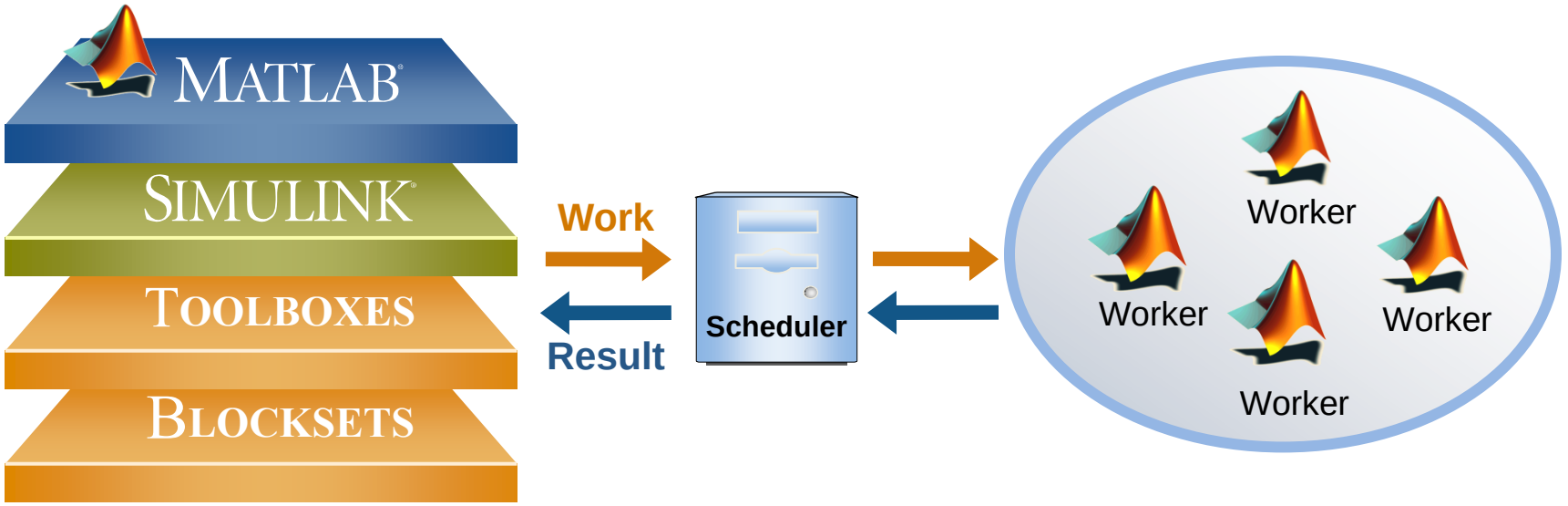
Summary of Options for Targeting GPUs

- Use GPU array interface with MATLAB built-in functions
- Execute custom functions on elements of the GPU array
- Create kernels from existing CUDA code and PTX files

Interactive to Scheduling

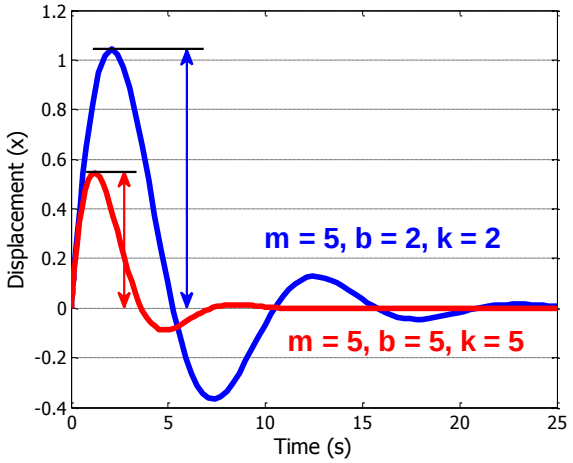
- Interactive
  - Great for prototyping
  - Immediate access to MATLAB workers
- Scheduling
  - Offloads work to other MATLAB workers (local or on a cluster)
  - Access to more computing resources for improved performance
  - Frees up local MATLAB session

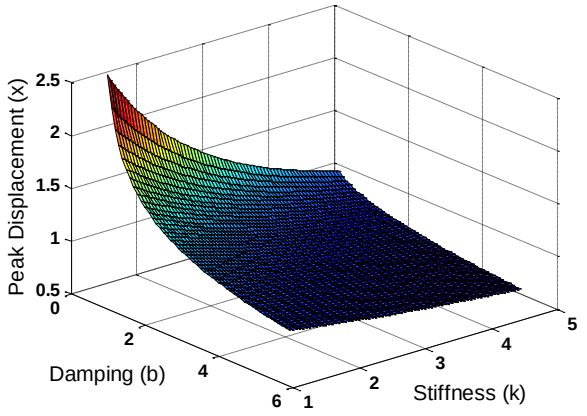
Scheduling Work



Example: Schedule Processing

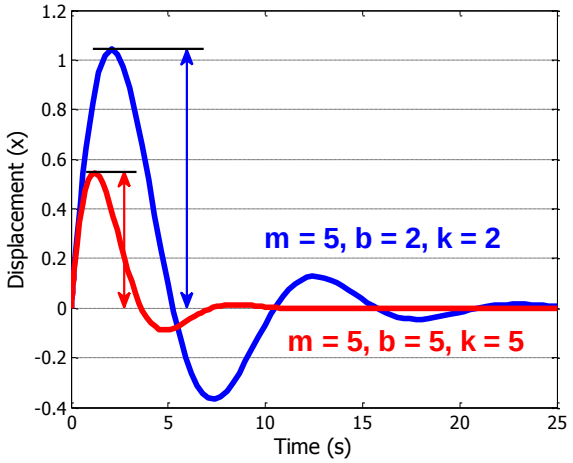
- Offload parameter sweep to local workers
- Get peak value results when processing is complete
- Plot results in local MATLAB

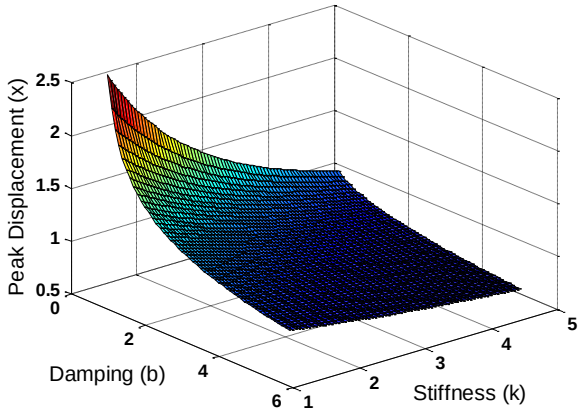




Summary of Example

- Used `batch` for off-loading work
- Used `matlabpool` option to off-load and run in parallel
- Used `load` to retrieve worker's workspace



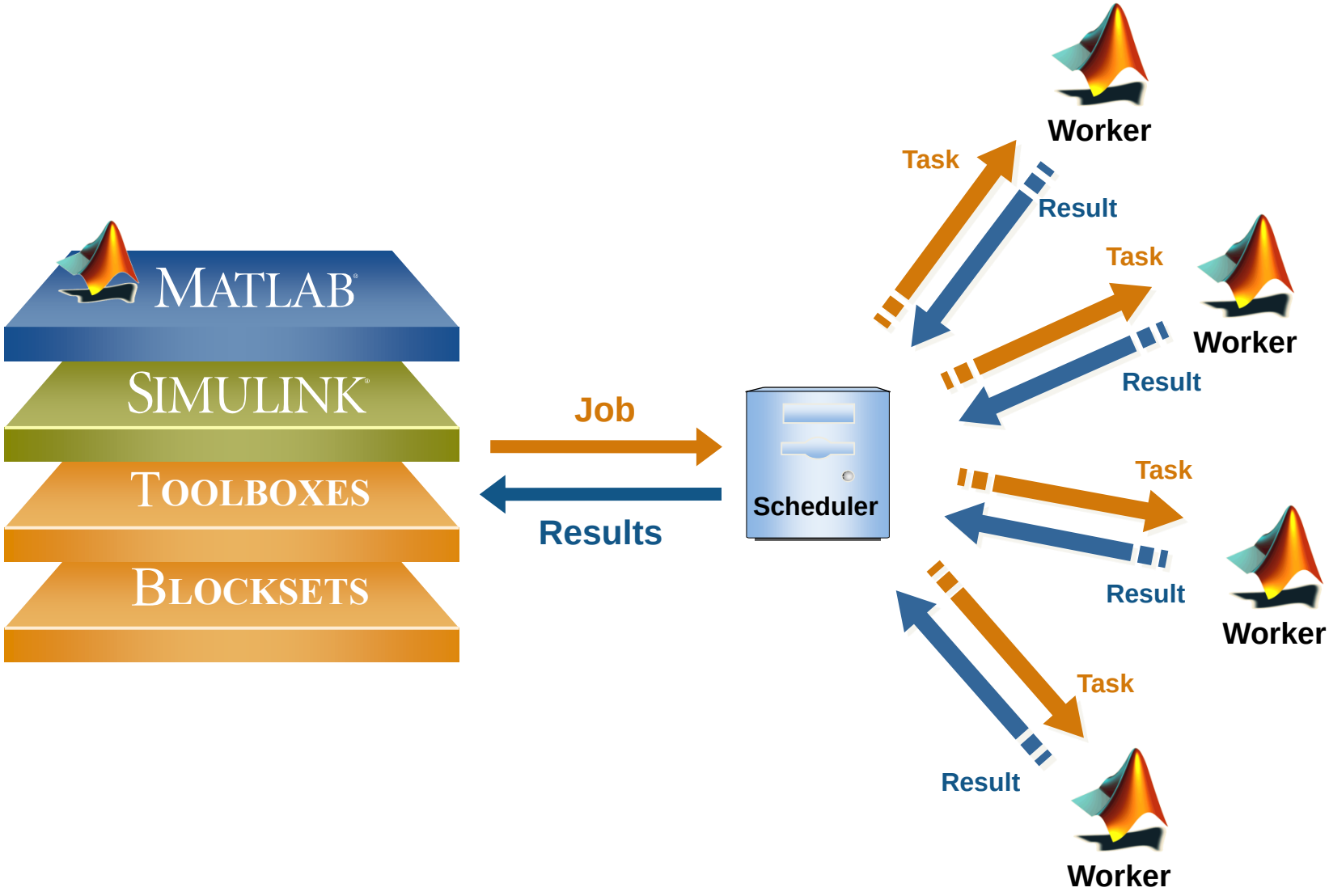


Task-Parallel Workflows

- `parfor`
  - Multiple independent iterations
  - Easy to combine serial and parallel code
  - Workflow
    - Interactive using `matlabpool`
    - Scheduled using `batch`

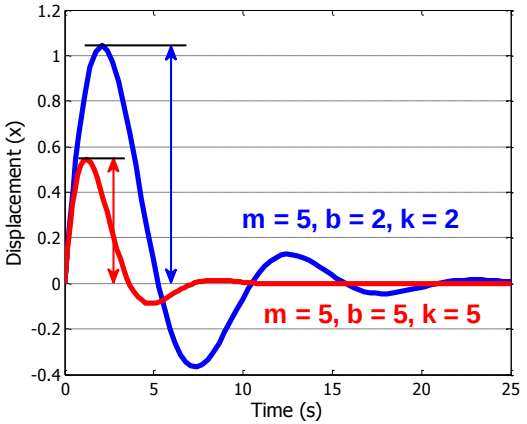
- `jobs/tasks`
  - Series of independent tasks; not necessarily iterations
  - Workflow ➔ Always scheduled

Scheduling Jobs and Tasks



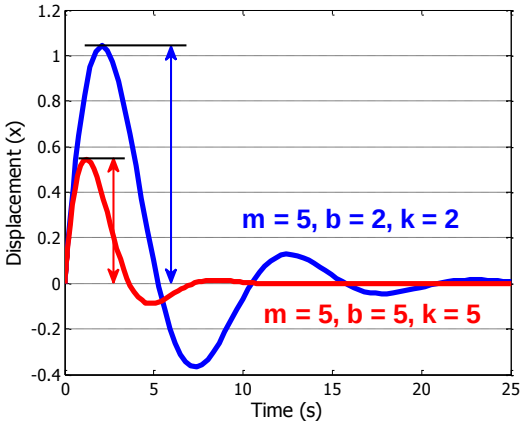
Example: Scheduling Independent Simulations

- Offload three independent approaches to solving our previous ODE example
- Retrieve simulated displacement as a function of time for each simulation
- Plot comparison of results in local MATLAB



Summary of Example

- Used `parcluster` to find resource
- Used `createJob` and `createTask` to set up the problem
- Used `submit` to off-load and run in parallel
- Used `fetchOutputs` to retrieve all task outputs



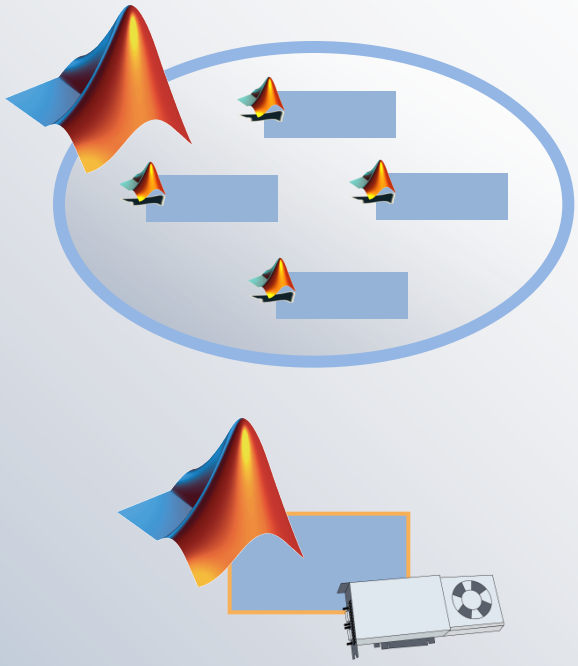
Factors to Consider for Scheduling

- There is always an overhead to distribution
  - Combine small repetitive function calls
- Share code and data with workers efficiently
  - Set job properties (`AttachedFiles`,`AdditionalPaths`)
- Minimize I/O
  - Enable `Workspace` option for `batch`
- Capture command window output
  - Enable `CaptureDiary` option for `batch`

Parallel Computing enables you to …

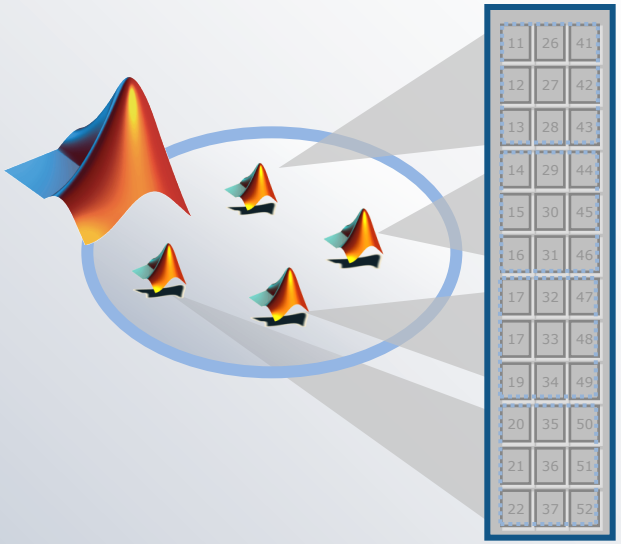
Larger Compute Pool

Speed up Computations



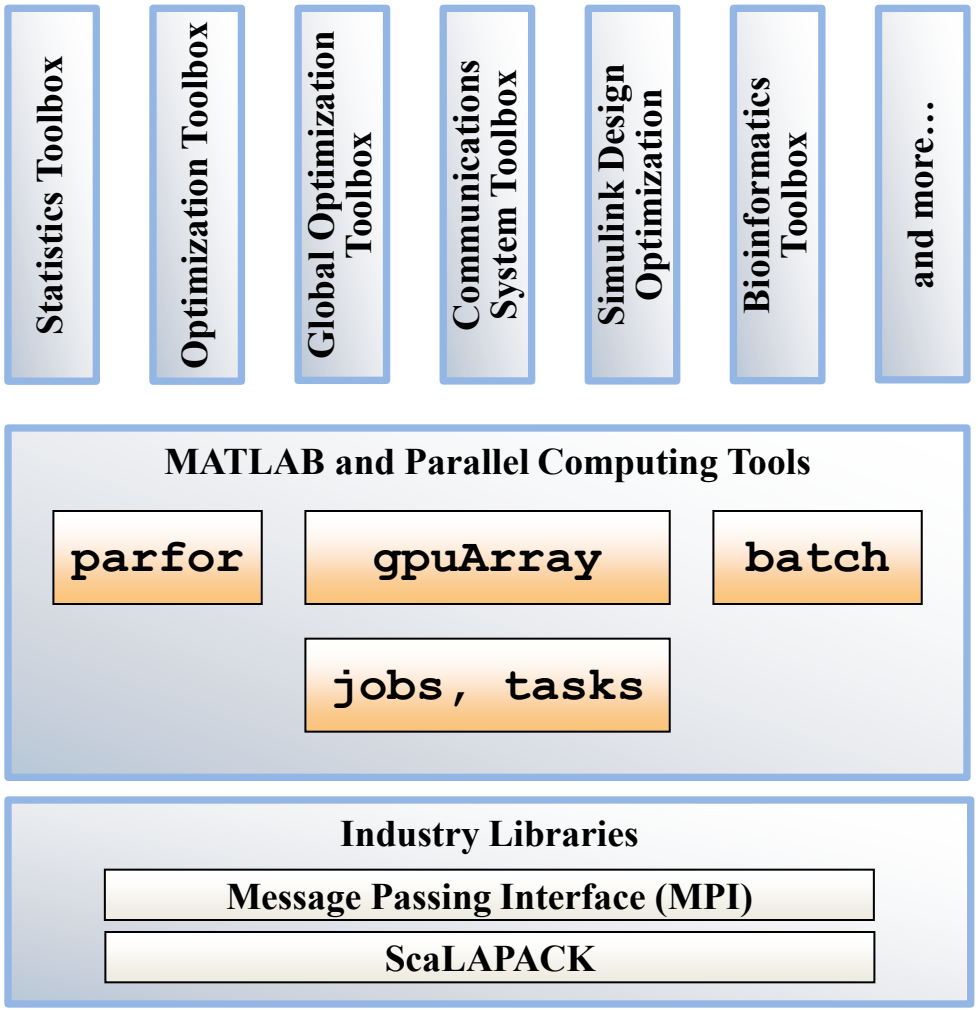
Larger Memory Pool

Work with Large Data

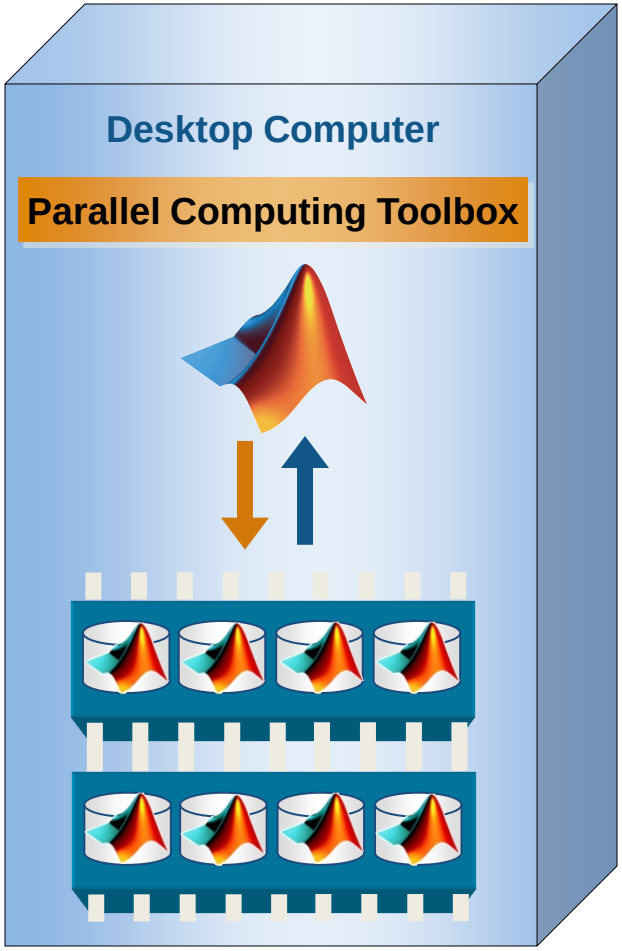


Parallel Computing with MATLAB

- Built in parallel functionality within specific toolboxes (also requires Parallel Computing Toolbox)
- High level parallel functions
- Low level parallel functions
- Built on industry standard libraries

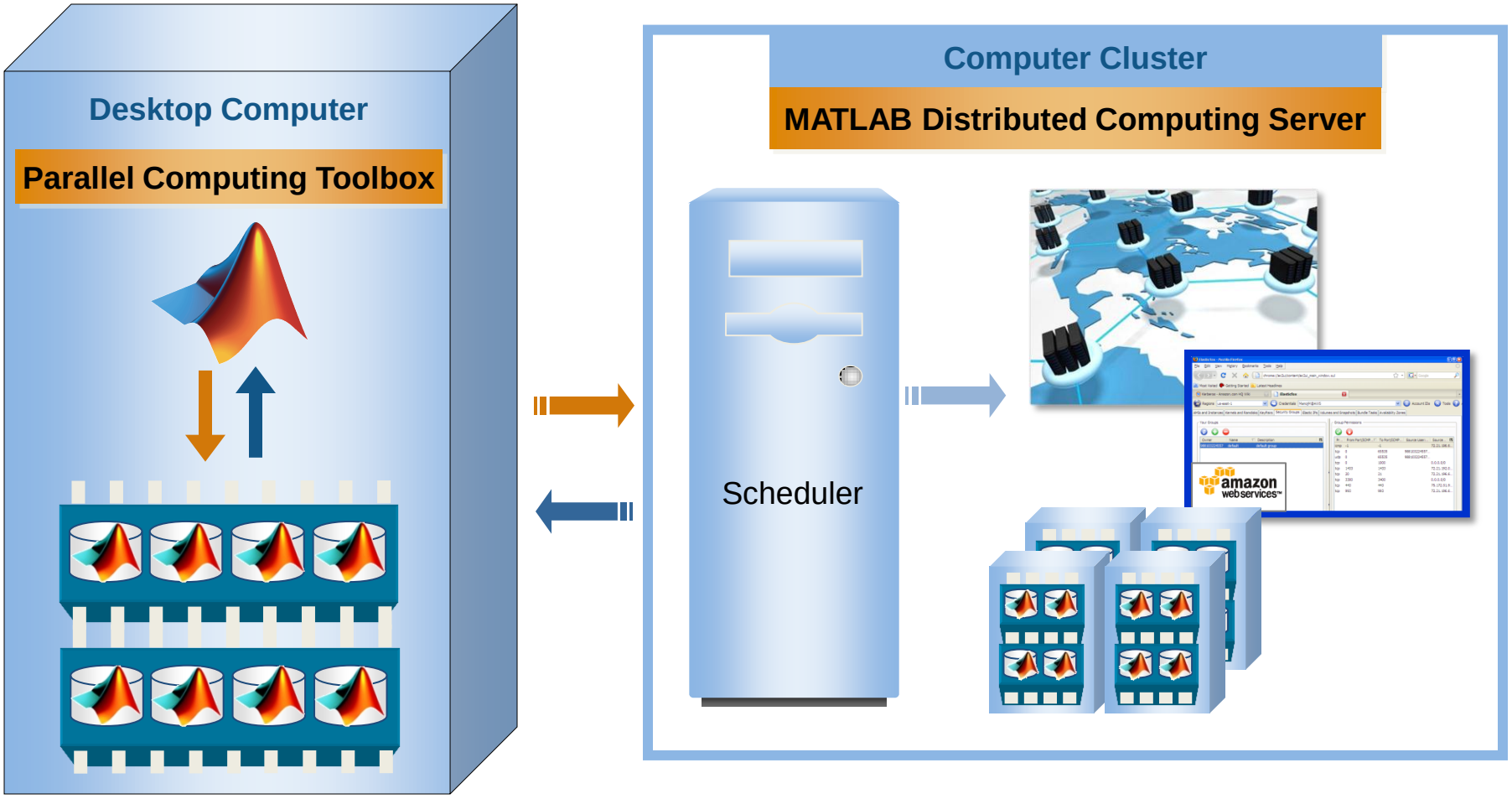


Run up to 12 Local Workers on Desktop



- Rapidly develop parallel applications on local computer
- Take full advantage of desktop power
- Separate computer cluster not required

Scale Up to Clusters, Grids and Clouds

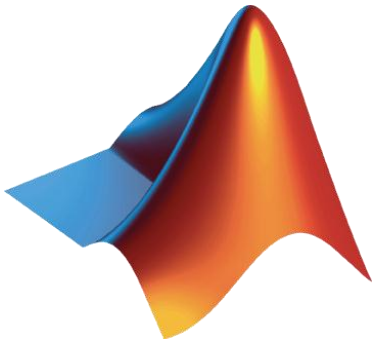


Agenda

- Leveraging the power of vector & matrix operations
- Addressing bottlenecks
- 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 parallel computing tools to take advantage of additional computing resources



Sample of Other Performance Resources

- MATLAB documentation

  Programming Fundamentals → Software Development → Performance

- Memory Management Guide

  www.mathworks.com/support/tech-notes/1100/1106.html?BB=1

- The Art of MATLAB, Loren Shure's blog

  blogs.mathworks.com/loren/

- MATLAB Answers

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

Support and Community

MathWorks | Book Program

MathWorks | Connections Program

MATLAB CENTRAL

MathWorks | Consulting Services

MathWorks | Training Services

Training Services

Exploit the full potential of MathWorks products

Flexible delivery options:

- Public training available worldwide
- Onsite training with standard or customized courses
- Web-based training with live, interactive instructor-led courses
- Self-paced interactive online training

More than 30 course offerings:

- Introductory and intermediate training on MATLAB, Simulink, Stateflow, code generation, and Polyspace products
- Specialized courses in control design, signal processing, parallel computing, code generation, communications, financial analysis, and other areas

Schedule

Courses	Days	Level	Mar.	Apr.	May	Jun.	Jul.	Aug.
MATLAB Fundamentals	3	기본	~~11~13~~	1~3 22~24	20~22	10~12	1~3 22~24	5~7 19~21
Simulink for System and Algorithm Modeling	2	기본	~~14~15~~	4~5 25~26	23~24	13~14	4~5 25~26	12~13
Signal Processing with Simulink	3	기본	19~21					26~28
Parallel Computing with MATLAB	2	중급						
MATLAB Based Optimization Techniques	1	중급	~~8~8~~				11~11	
Image Processing with MATLAB	2	중급	~~6~7~~				9~10	
MATLAB for Data Processing and Visualization	1	중급	~~4~4~~					
Statistical Methods in MATLAB	2	중급			14~15			22~23

Note. 상기 일정은 등록 수에 따라 조기 마감이 될 수 있으며 또는 취소가 될 수 있습니다.

Contact

- Account Manager 이장원 차장
  johney.lee@mathworks.com
  02-6006-5128

- Application Engineer 엄준상
  joseph.eom@mathworks.com
  02-6006-5136

- Technical Support
  www.mathworks.com/support