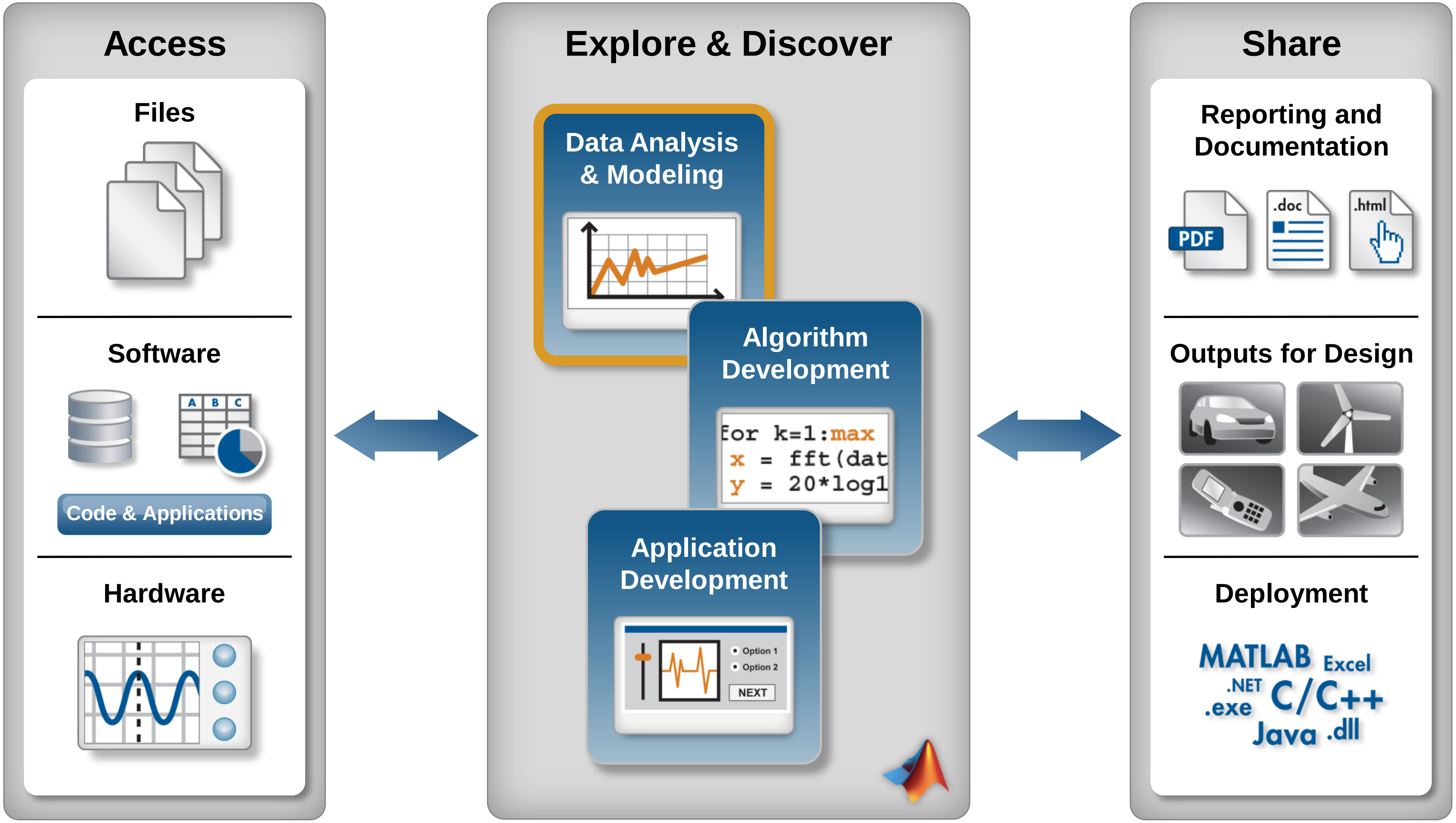
Speeding up MATLAB and Handling Large Data Sets

Mathijs Faase
Rob Heijmans
Zoetermeer - 26th April 2012

Agenda

13:30	Introduction – Scope of the seminar
13:45	Part I: Improvements with minimal effort
14:30	*Break*
14:50	Part II: High level parallel programming
15:40	*Break*
16:00	Part III: Specialized solutions
16:50	Wrap up
17:00	*Drinks*

What is MATLAB?



Why do some of our customers want to speed up their MATLAB applications?

"Our stepwise regression takes approx. 5 minutes per step for up to 100 steps and we have to make a decision after each step. This process takes one day and we need this to go down to 15 minutes!"

Why do some of our customers want to speed up their MATLAB applications?

"We need to run our credit portfolio for 2000 clients, and with 1 million simulation paths each, within 5 days! This would require 20x speed improvement!"

Time means different things to different people

Why do you want to speed up your MATLAB applications?

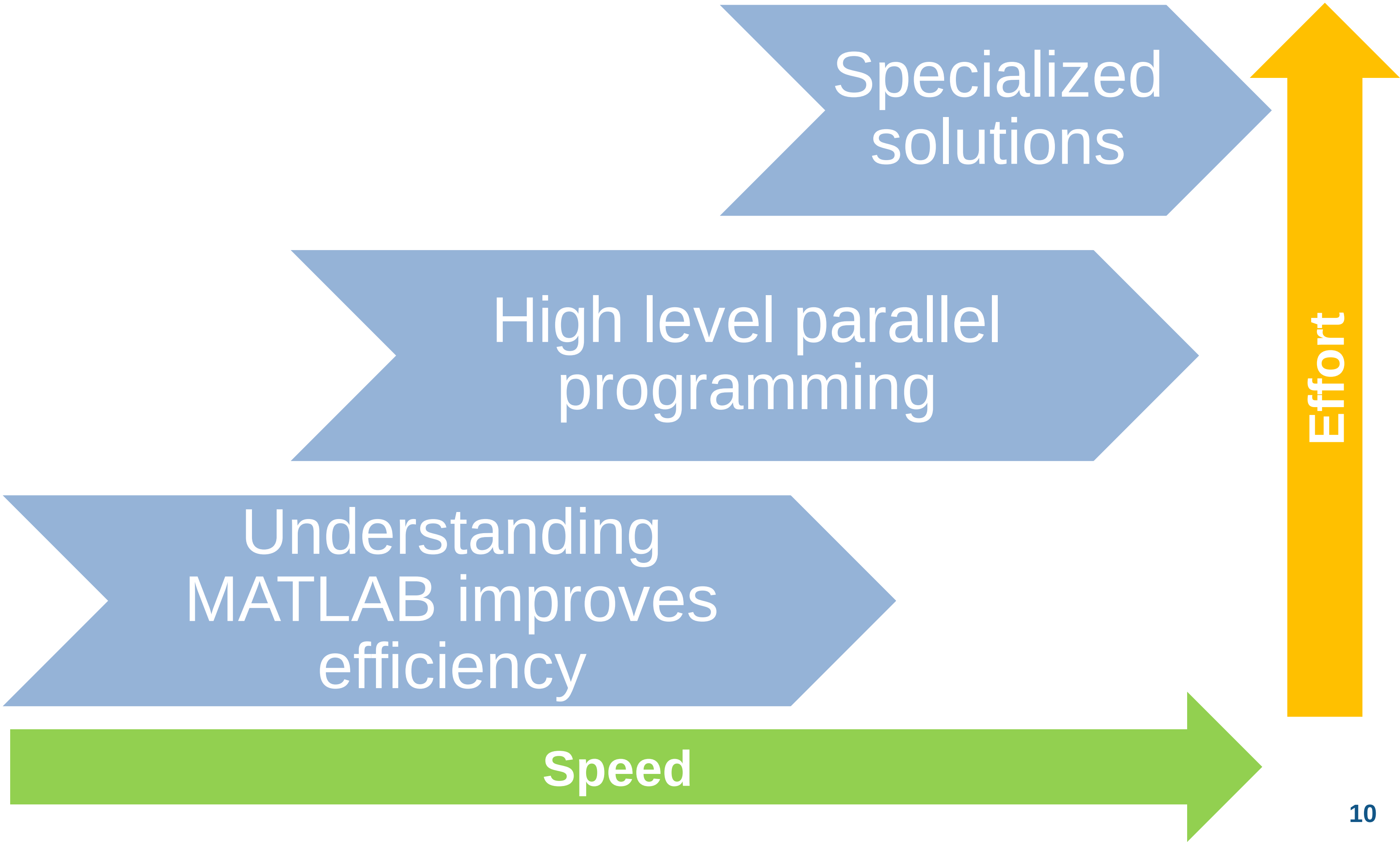
Agenda

13:30	Introduction
13:45	Part I: Improvements with minimal effort
14:30	*Break*
14:50	Part II: High level parallel programming
15:50	*Break*
16:10	Part III: Specialized solutions
16:50	Wrap up
17:00	*Drinks*

Three Key Takeaways

- 1. If you understand where to put your effort, writing fast and efficient code becomes easy
- 2. You can make the most of your hardware without becoming a programming guru
- 3. If your desktop isn’t enough, you can easily upscale to use GPUs or computer clusters

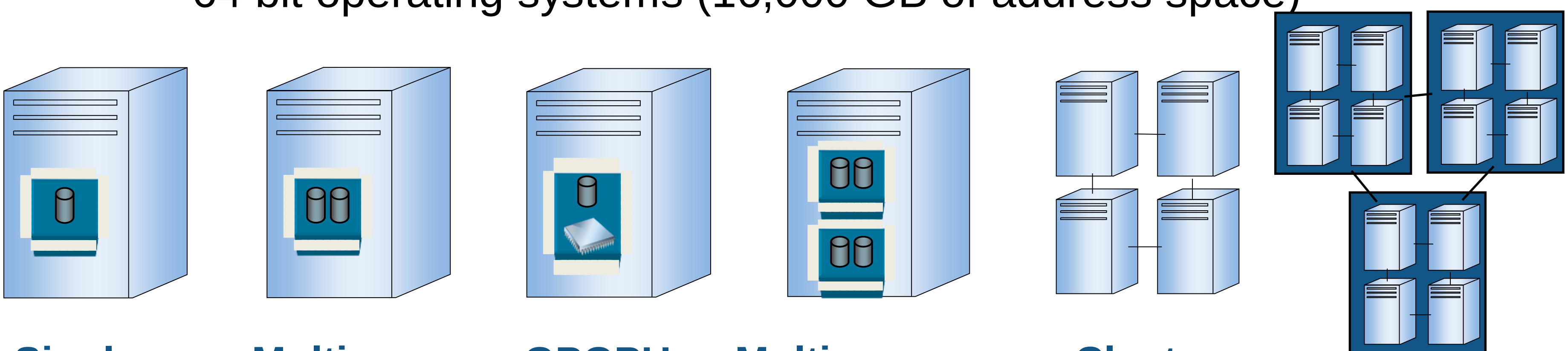
How can you speed up your MATLAB programs?



An overview of hardware solutions

- More processing power
  - Faster processors
  - Dedicated hardware
  - More processors
- More memory
  - 32 bit operating systems (4 GB of address space)
  - 64 bit operating systems (16,000 GB of address space)

4000 times the address space on a 64 bit OS

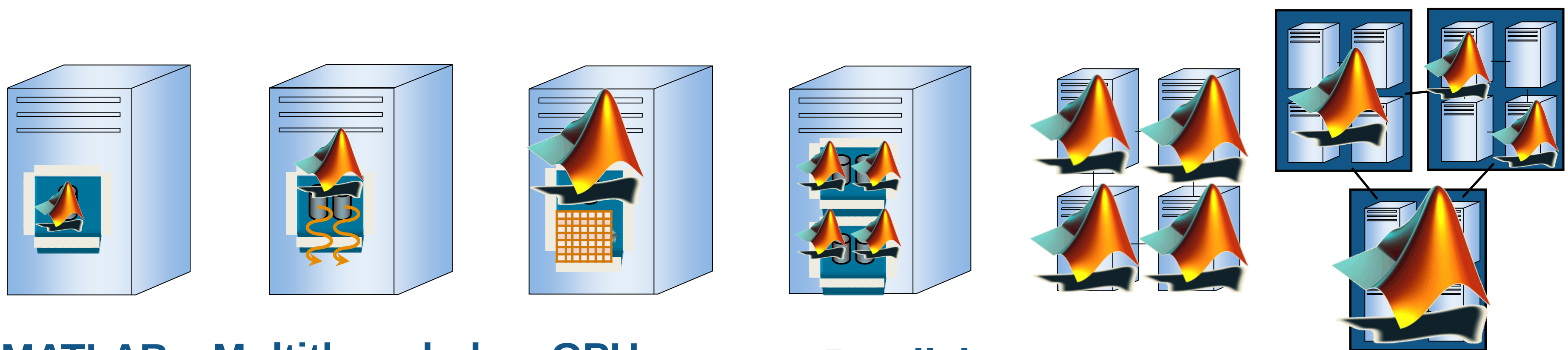


Single Processor | Multi-core | GPGPU, FPGA | Multiprocessor | Cluster | Grid, Cloud

MathWorks

Software solutions for all hardware

- MATLAB
- Parallel Computing Toolbox
- MATLAB Distributed Computing Server (MDCS)



MATLAB | Multithreaded | GPU Computing | Parallel Computing | Distributed Computing

Agenda

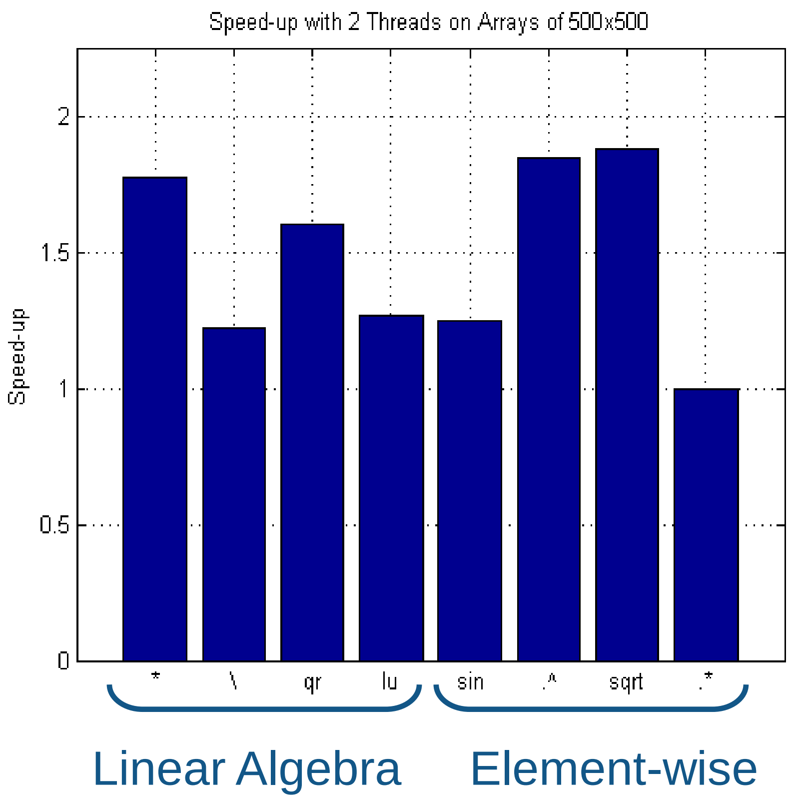
13:30	Introduction
13:45	Part I: Improvements with minimal effort
14:30	*Break*
14:50	Part II: High level parallel programming
15:50	*Break*
16:10	Part III: Specialized solutions
16:50	Wrap up
17:00	*Drinks*

Improvements with minimal effort

- **Multithreading capabilities in MATLAB**
- MATLAB data storage model
- Finding bottlenecks
- Techniques for improving performance

MATLAB Multithreaded: new releases are faster

- No code changes required
- Enabled at MATLAB start-up
- Vector math improved
  - Linear Algebra operations
  - Element-wise operations



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

Improvements with minimal effort

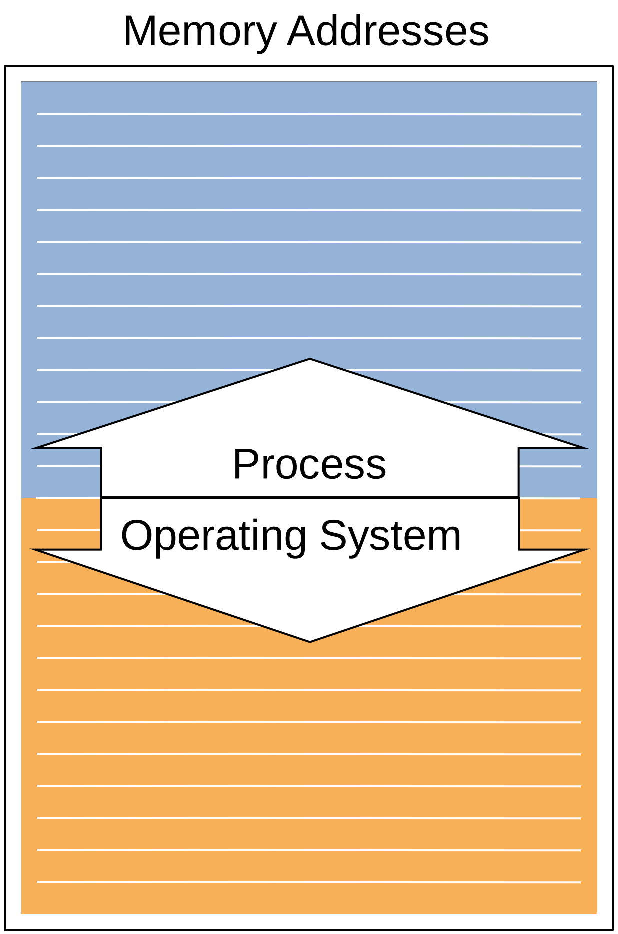
- Multithreading capabilities in MATLAB
- **MATLAB data storage model**
- Finding bottlenecks
- Techniques for improving performance

Understand how MATLAB uses memory

- Prevent slow execution/out of memory errors
  – Understand the constraints
  – Identify bottlenecks
- Find the best tradeoff between programming effort and achieving your goals

What is the largest array you can create in MATLAB on 32 bit Windows XP?

Understand the Constraints …

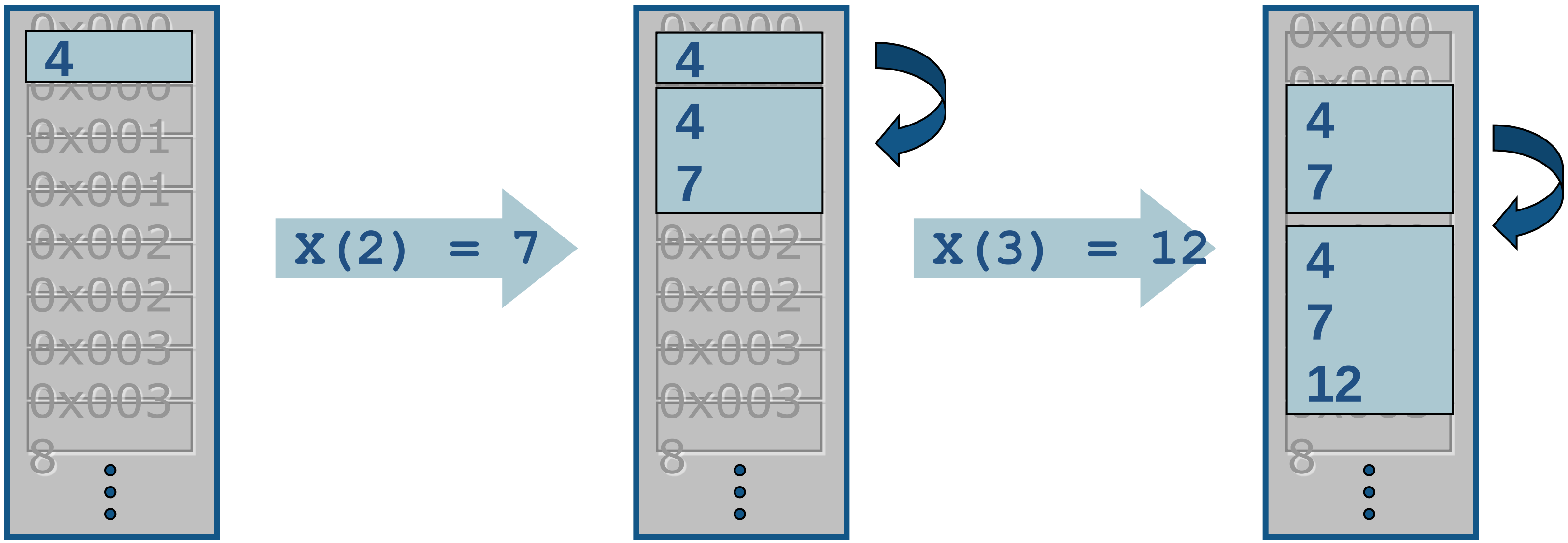


>> memory

a) 0.5 GB
b) 1.0 GB
c) 1.5 GB
d) 2.0 GB
e) 2.5 GB

Memory Allocation for Dynamic Arrays

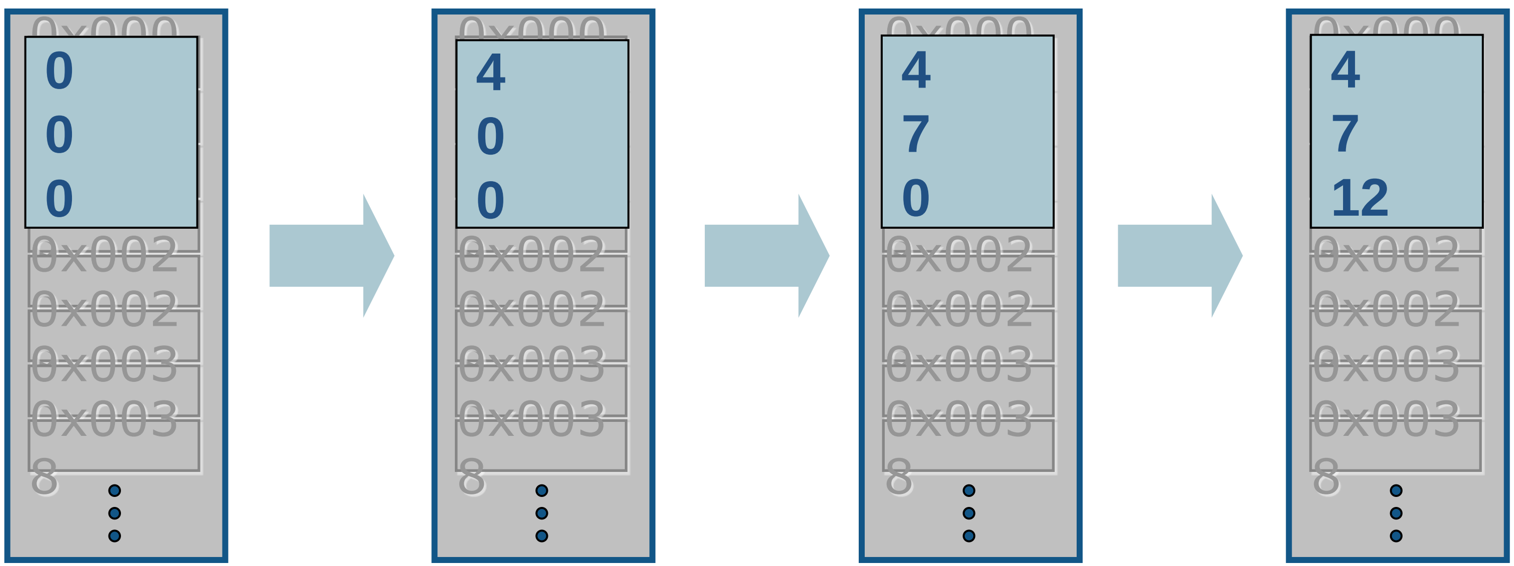
```
>> x = 4
>> x(2) = 7
>> x(3) = 12
```



Reduce Memory Operations

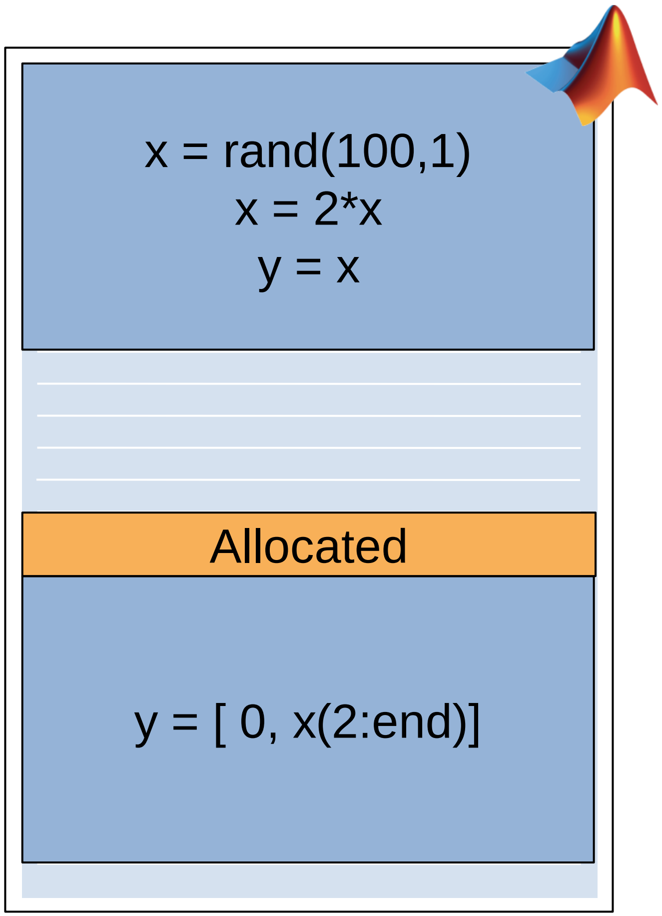
```
>> x = zeros(3,1)
>> x(1) = 4
>> x(2) = 7
>> x(3) = 12
```

BEST PRACTICE
Preallocation



Contiguous memory and copy on write

```
>> x    = rand(100,1)
>> x    = 2*x
>> y    = x
>> y(1) = 0
```



- Do not grow arrays within loops!
- MATLAB handles memory efficiently

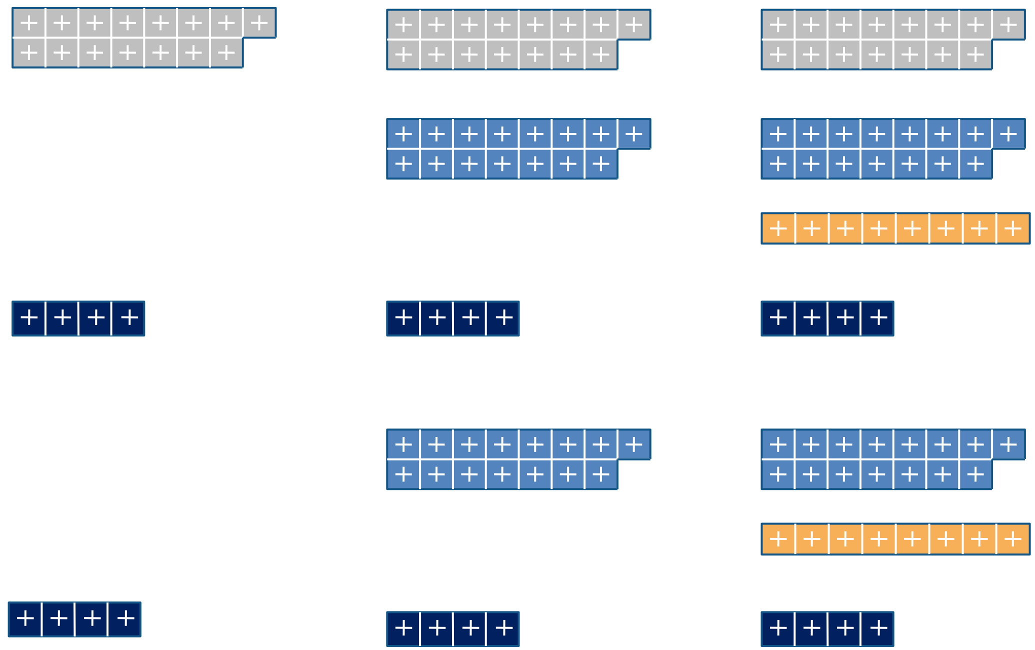
Use the best way to store your data

- Numeric arrays
  - Basic matrix; contains only numbers
  - Use sparse where appropriate
- Cell arrays
  - Can contain a mix of different datatypes
  - Very good for storing string arrays
- Structures
  - Can contain a mix of different datatypes
  - Make your code more readable

Use only the precision you need

- Numerical data types
  - Float: double and single precision (8 and 4 bytes)
  - Integer: signed and unsigned (1-4 bytes)
  - Logical: 0 or 1 only (1 byte)
- Floating point for math (e.g. linear algebra)
- Integers where appropriate (e.g. images)

Be aware of container overhead



	Numeric Array	Cell Array	Structure
Variable header: 112 bytes*			
Element header: 112 bytes*			
Field name: 64 bytes			
Data: 8 bytes			
2nd header: 112 bytes*			
Field name: 64 bytes			
Data: 8 bytes			
Total Used:	120 + 8 = 128	232 + 120 = 352	296 + 184 = 480
Total Reported:	8 + 8 = 16	120 + 120 = 240	184 + 184 = 368

*On a 64-bit system

Plotting data

- How much memory is needed to plot a 10 MB double array?

```
>> x = sin(1:125e4);
>> plot(x);
```

a) 0 MB

b) 10 MB

c) 20 MB

d) 30 MB

e) 40 MB

Plotting Increases Memory Requirements

Plot only what you need

- Every plot independently stores x and y data

```
>> x = sin(1:125e4); %10MB
>> plot(x)          ; %20MB for x and y data
```

- Integers plotted as doubles
- Strategies:
  – Downsample your data prior to plotting (e.g. every 10th element)
  – Divide your data into regular intervals and plot values of interest (e.g. open and close for stock prices, or min/max values)

Load only the data you need

- ASCII file: **`textscan`**
  - Selectively choose columns to load or ignore
  - Selectively choose rows to load (i.e. block processing)

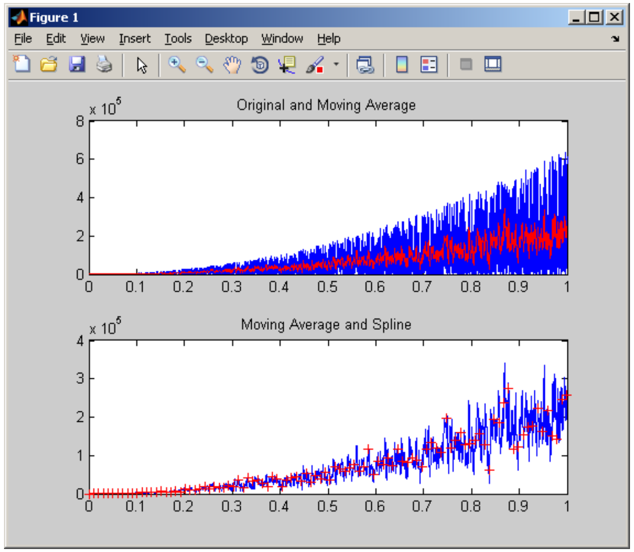
- Binary file: **`memmapfile`**
  - Read and write directly to/from file on disk
  - Overlay address space directly onto file
  - MATLAB dynamically shifts address space to handle larger files
    - e.g. >1.5 GB files on 32 bit Windows can be accessed

Improvements with minimal effort

- Multithreading capabilities in MATLAB
- MATLAB data storage model
- **Finding bottlenecks**
- Techniques for improving performance

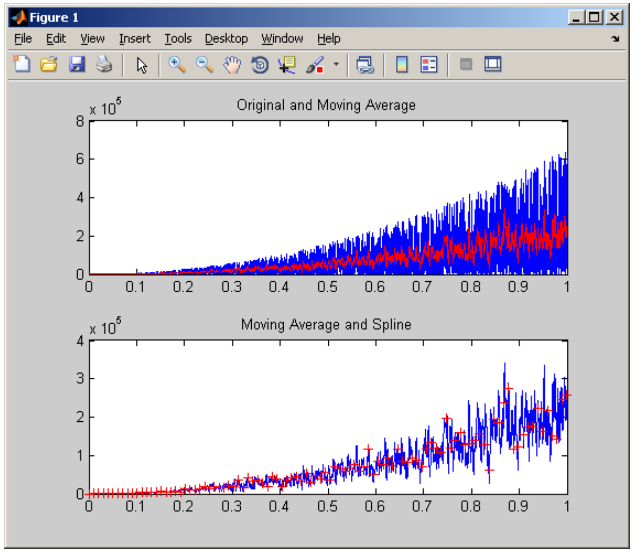
Example: fitting data

- 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



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
  - Trade-off modularization, readability and performance
  - Integrate other languages or additional hardware
    - E.g. MEX, GPU / CUDA, FPGAs…

Improvements without programming effort

- Multithreading capabilities in MATLAB
- MATLAB data storage model
- Finding bottlenecks
- **Techniques for improving performance**

Techniques for improving performance

- Vectorization
  - Take full advantage of BLAS and LAPACK
  - Brute force: cellfun, structfun, arrayfun …
- Preallocation
  - Minimize changing variable class
- Mexing (compiled code)

Improvements with minimal effort

MATLAB helps you to take advantage of your hardware

- Use the latest release to take advantage of the latest improvements in hardware
- Use the profiler to identify bottlenecks
- Write efficient code

Three Key Takeaways

- 1. If you understand where to put your effort, writing fast and efficient code becomes easy
- 2. You can make the most of your hardware without becoming a programming guru
- 3. If your desktop isn't enough, you can easily upscale to use GPUs or computer clusters

Agenda

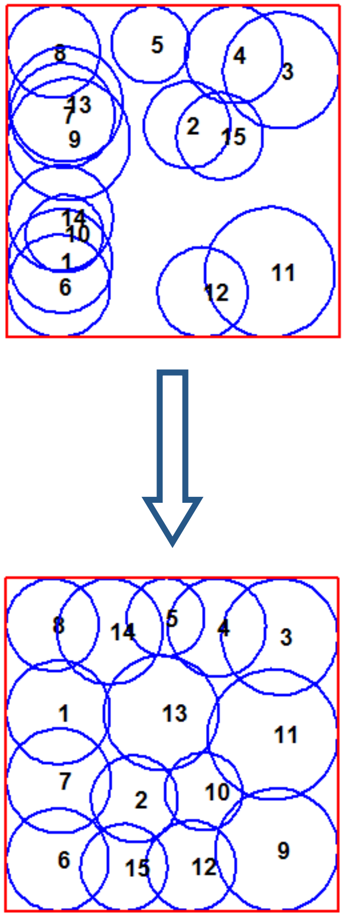
13:30	Introduction
13:45	Part I: Improvements with minimal effort
14:30	*Break*
14:50	Part II: High level parallel programming
15:50	*Break*
16:10	Part III: Specialized solutions
16:50	Wrap up
17:00	*Drinks*

High level parallel programming

- **Support of parallel computing built into toolboxes**
- Task distribution
- Data distribution
- Interactive to scheduled

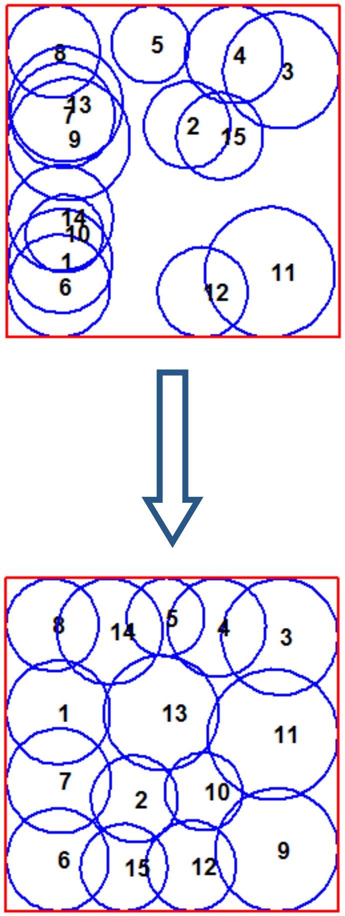
Example: optimizing tower placement

- 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`



Programming parallel applications

Level of control	Required effort
Minimal	**None**
Some	**Straightforward**
Extensive	**Involved**

Programming parallel applications

Level of control	Required effort
Minimal	Built-in into Toolboxes
Some	Straightforward
Extensive	Involved

Parallel support in Optimization Toolbox

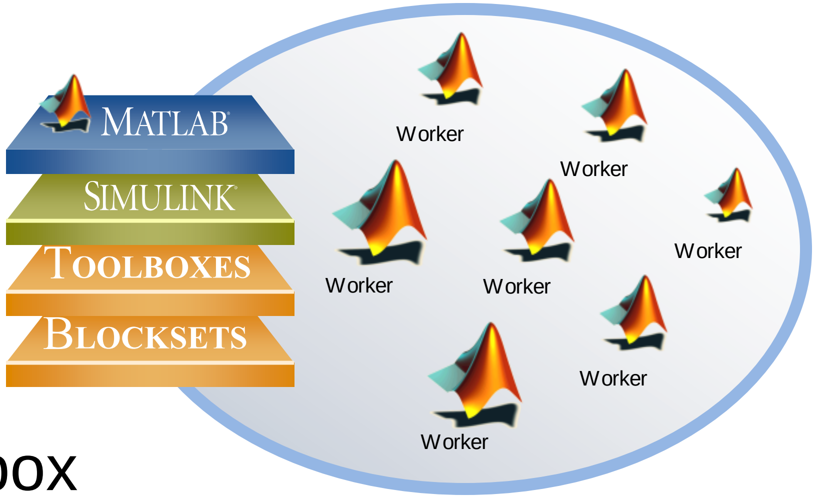
- Functions:
  - **`fmincon`**
    - Finds a constrained minimum of a function of several variables
  - **`fminimax`**
    - Finds a minimax solution of a function of several variables
  - **`fgoalattain`**
    - Solves the multiobjective goal attainment optimization problem

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

Toolboxes with built-in support

Contain functions to directly leverage the Parallel Computing Toolbox

- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- SystemTest
- Simulink Design Optimization
- Bioinformatics Toolbox
- Model-Based Calibration Toolbox
- Communications System Toolbox



High level parallel programming

- Support of parallel computing built into toolboxes
- **Task distribution**
- Data distribution
- Interactive to scheduled

Programming parallel applications

Level of control	Required effort
Minimal	None
Some	Straightforward
Extensive	Involved

Programming parallel applications

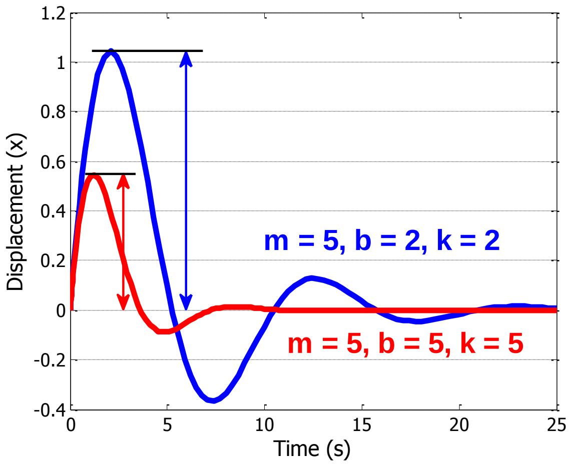
Level of control	Required effort
Minimal	Built-in into Toolboxes
Some	High Level Programming (parfor)
Extensive	Involved

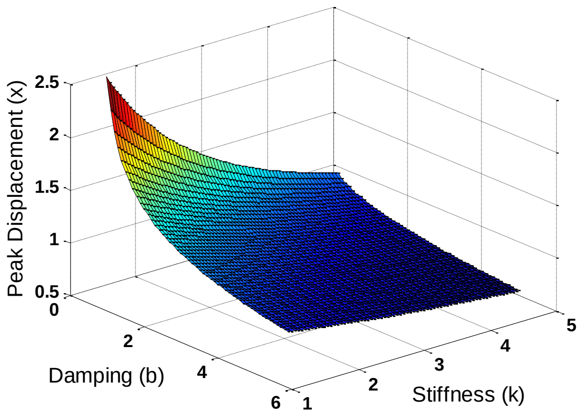
Example: parameter sweep of ODEs

- Solve a 2nd order ODE

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

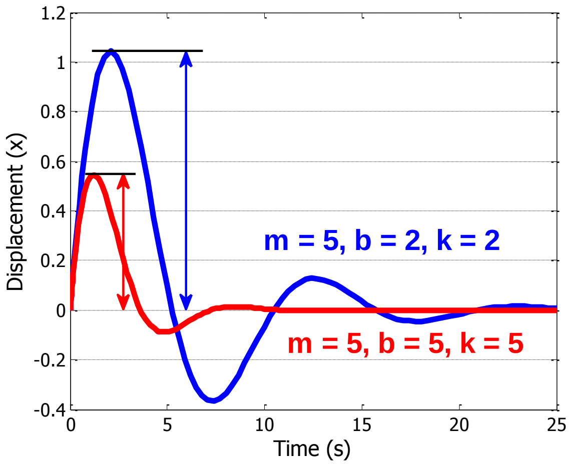
- Simulate with different values for ***b*** and ***k***
- Record peak value for each run
- Plot results

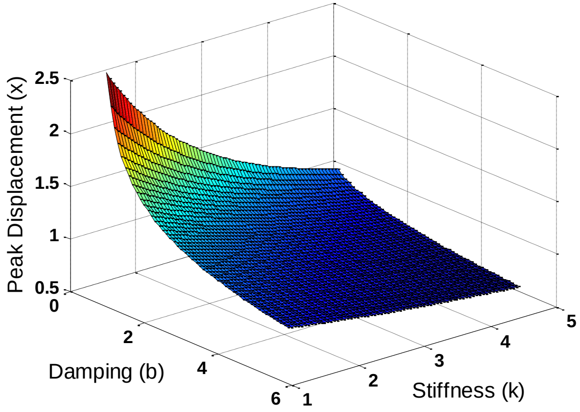




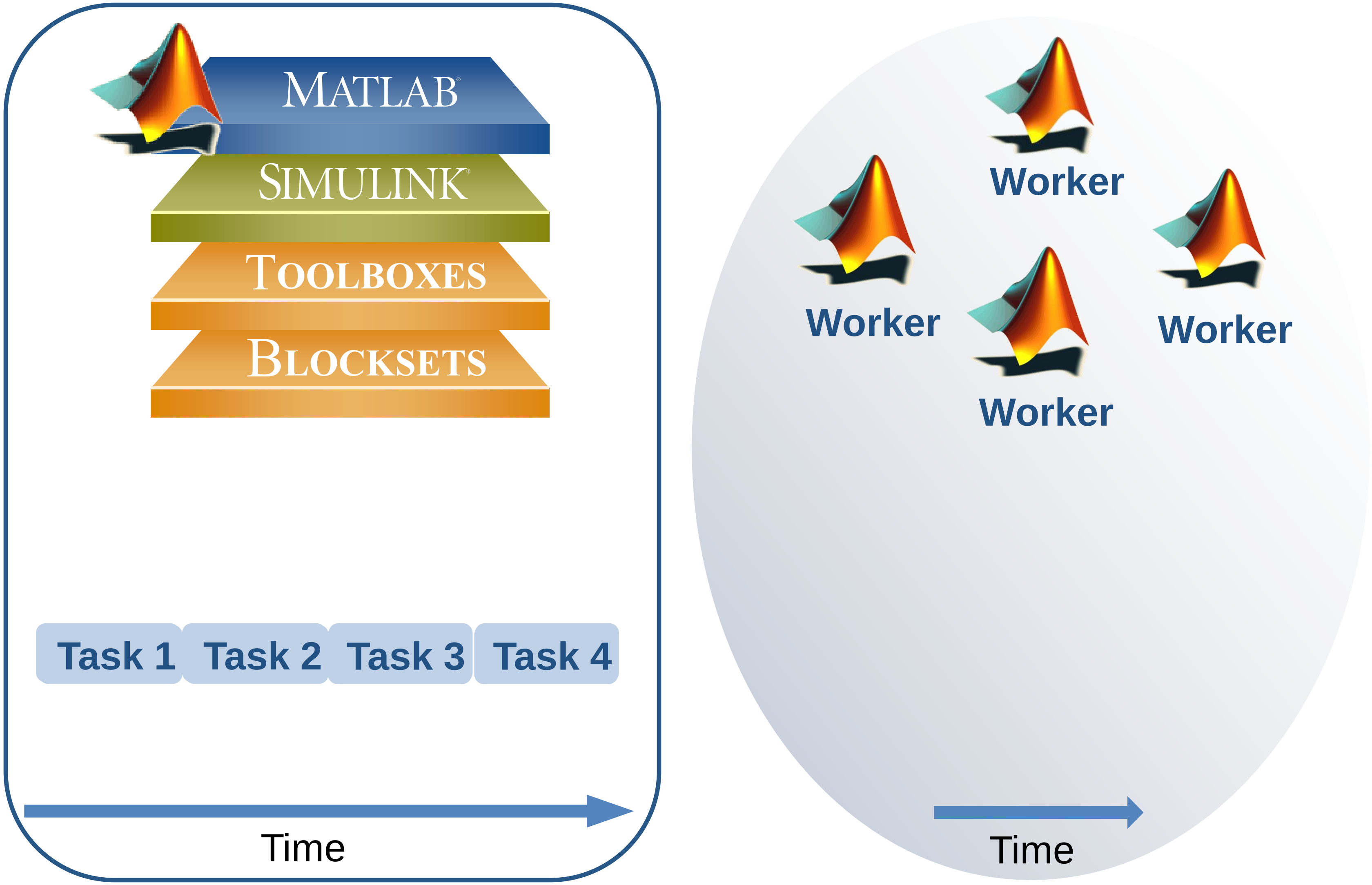
Summary of example

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

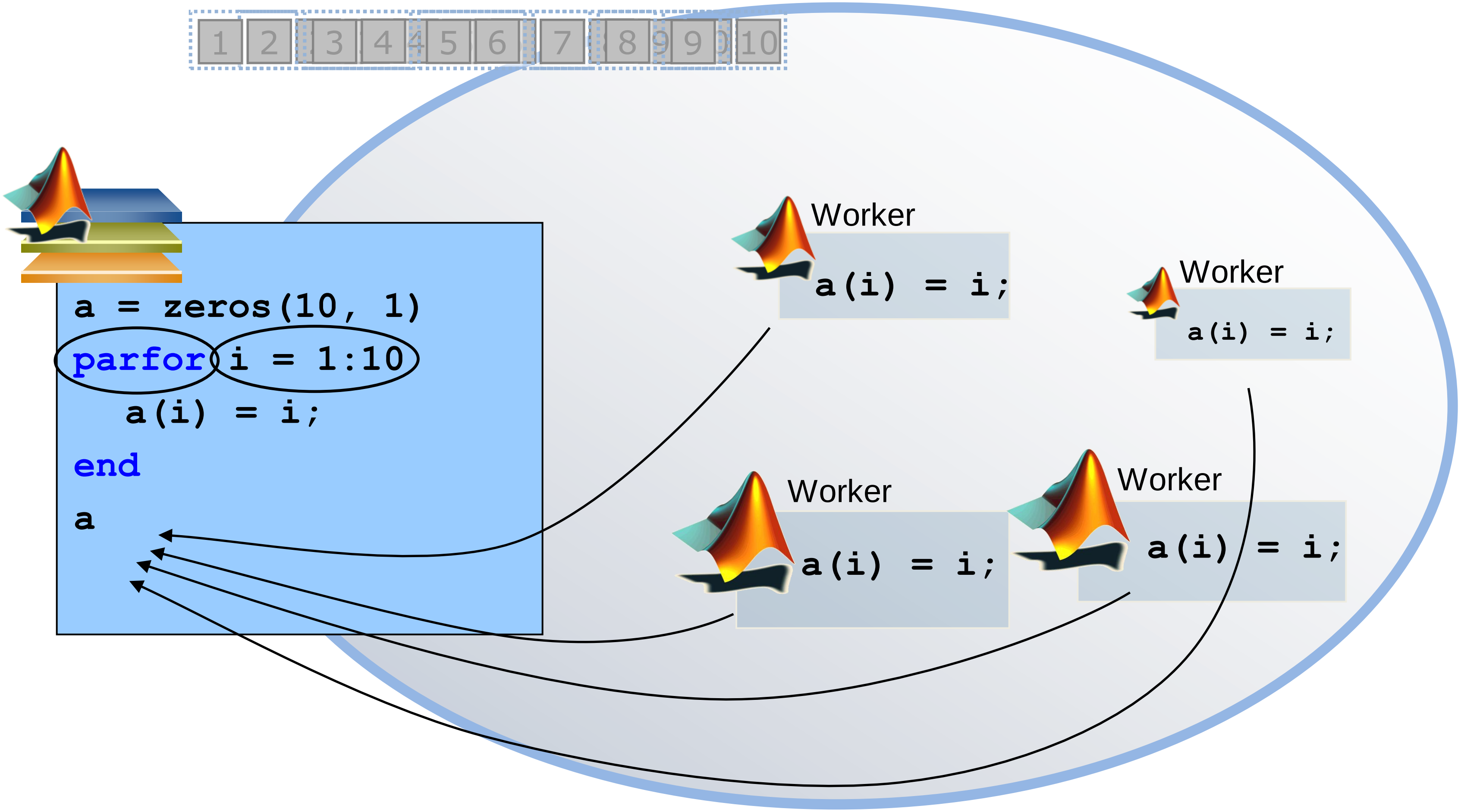




Parallel tasks



The mechanics of `parfor` loops



Pool of MATLAB Workers

Converting `for` to `parfor`

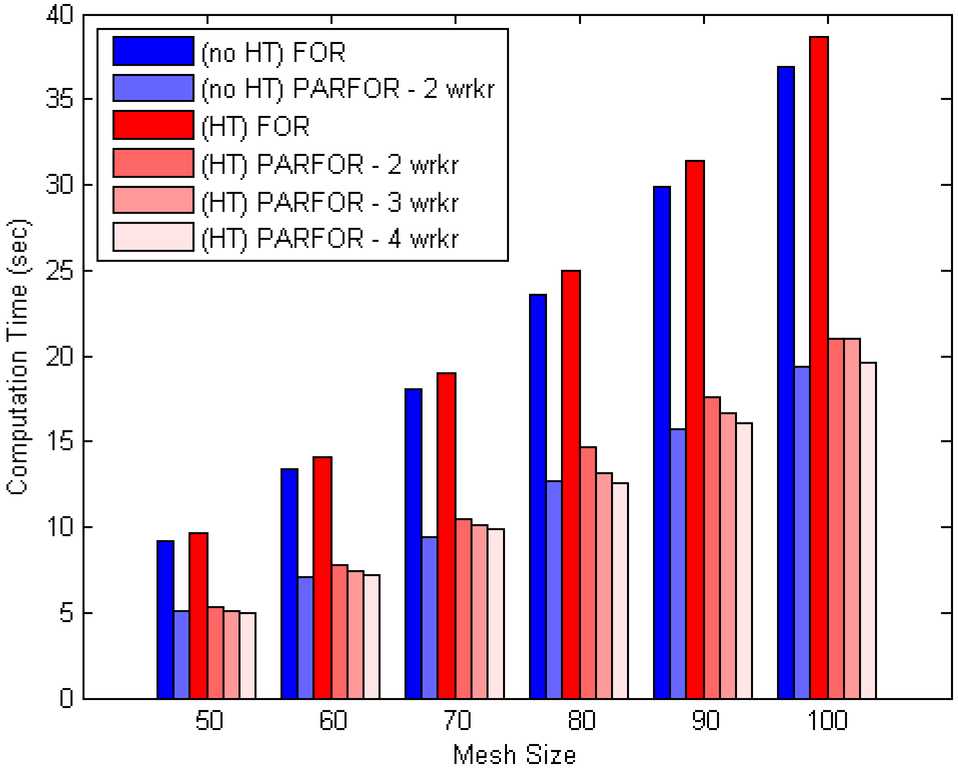
- Requirement for `parfor` loops
  - Order independent
- Constraints on the loop body
  - Cannot “introduce” variables (e.g. `eval`, `load`, `global`, etc.)
  - Cannot contain `break` or `return` statements
  - Cannot contain another `parfor` loop
  - Use MATLAB Code Analyzer to resolve issues
- `parfor` will run serially if no workers are available

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
- Classification (slicing) of variables
- http://blogs.mathworks.com/loren/2009/10/02/using-parfor-loops-getting-up-and-running/

Some notes on Simultaneous Multi-Threading (hyperthreading)

- SMT provides your operating system with 2 logical cores for each physical core
- Computational capability is <u>not</u> increased



High level parallel programming

- Support of parallel computing built into toolboxes
- Task distribution
- **Data distribution**
- Interactive to scheduled

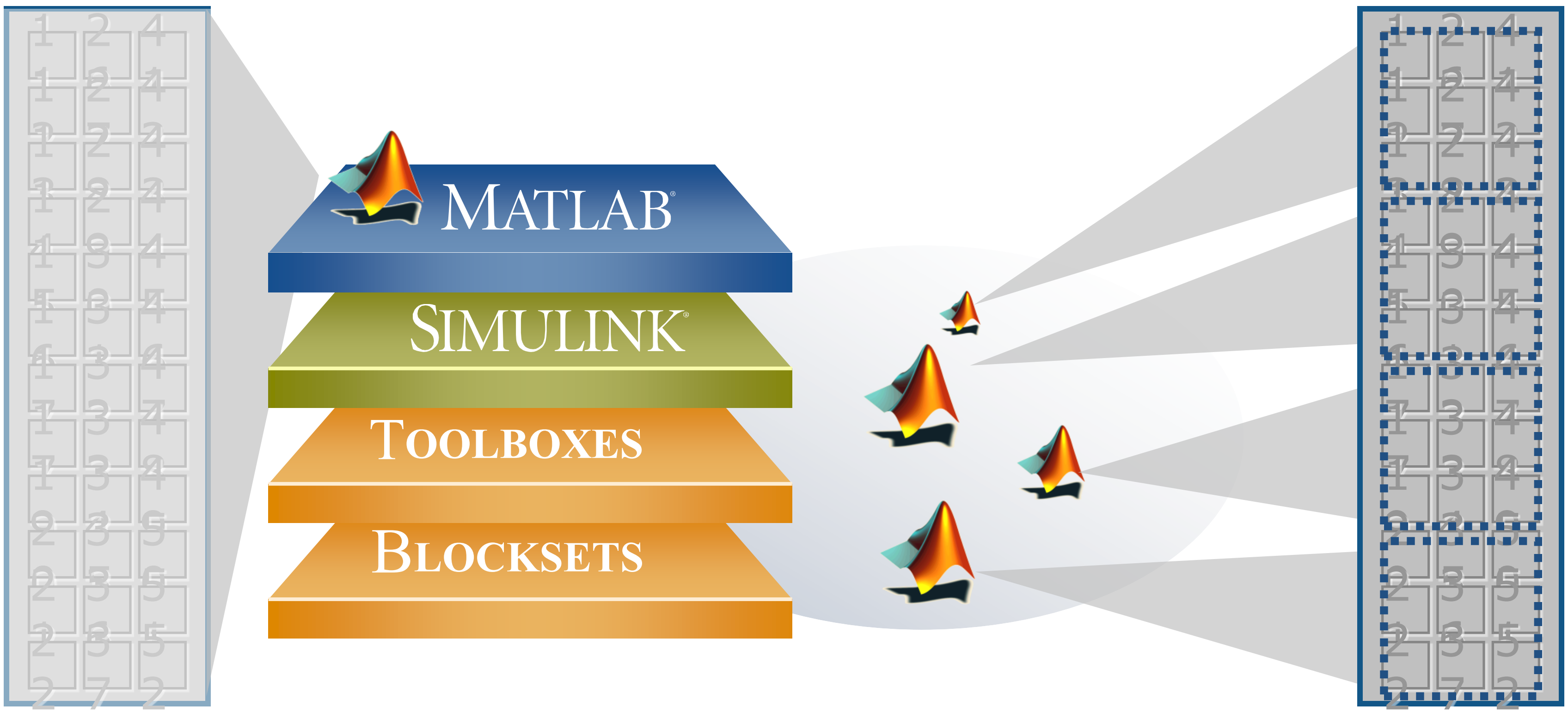
Programming parallel applications

Level of control	Required effort
Minimal	None
Some	Straightforward
Extensive	Involved

Programming parallel applications

Level of control	Required effort
Minimal	Built-in into Toolboxes
Some	High Level Programming (distributed, spmd)
Extensive	Involved

Parallel data distribution



Advice for distributing data

- Prototype locally before going to a cluster
  - But the data transfer might be not significant
- Use a scalable approach
  - Do not hard code the number of workers
- Take care of path and data dependencies

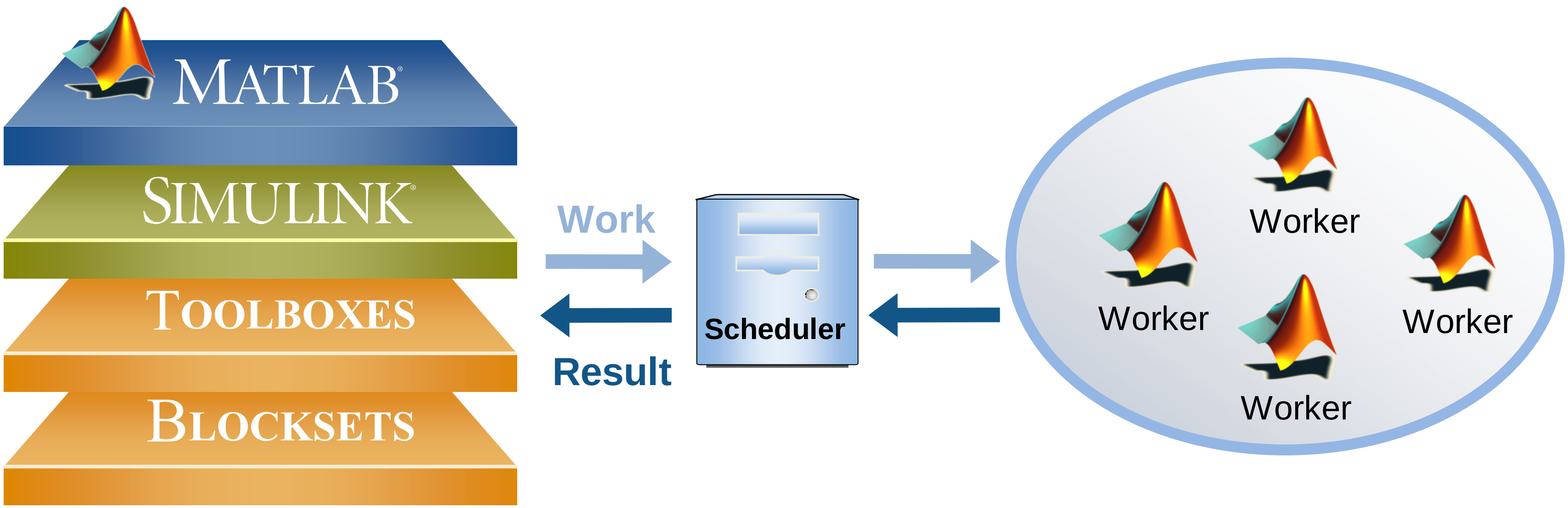
High level parallel programming

- Support of parallel computing built into toolboxes
- Task distribution
- Data distribution
- **Interactive to scheduled**

Interactive to scheduled

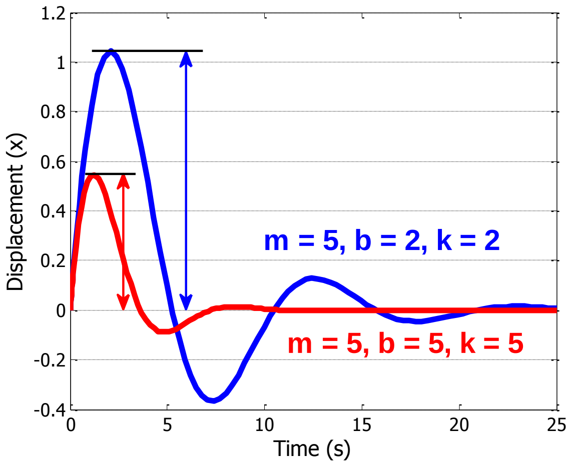
- Interactive
  – Great for prototyping
  – Immediate access to MATLAB workers
- Scheduled
  – Offloads work to other MATLAB workers (local or on a cluster)
  – Frees up local MATLAB session

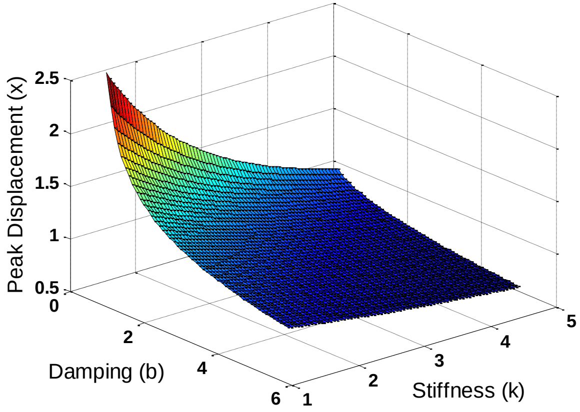
Scheduling work



Example: schedule processing

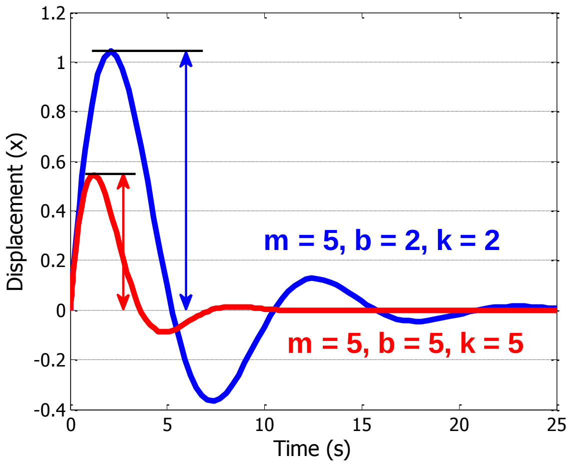
- Offload parameter sweep to local workers
- Get peak value results when processing is complete
- Plot results in local MATLAB

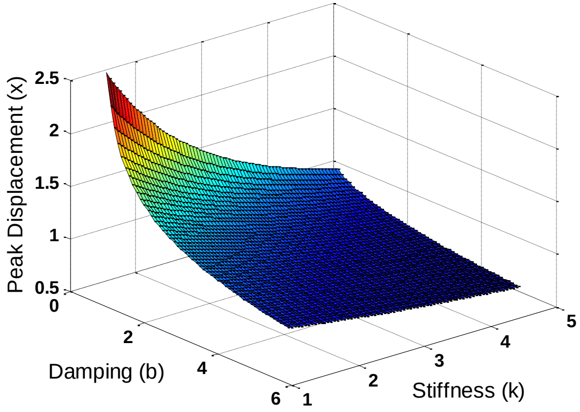




Summary of example

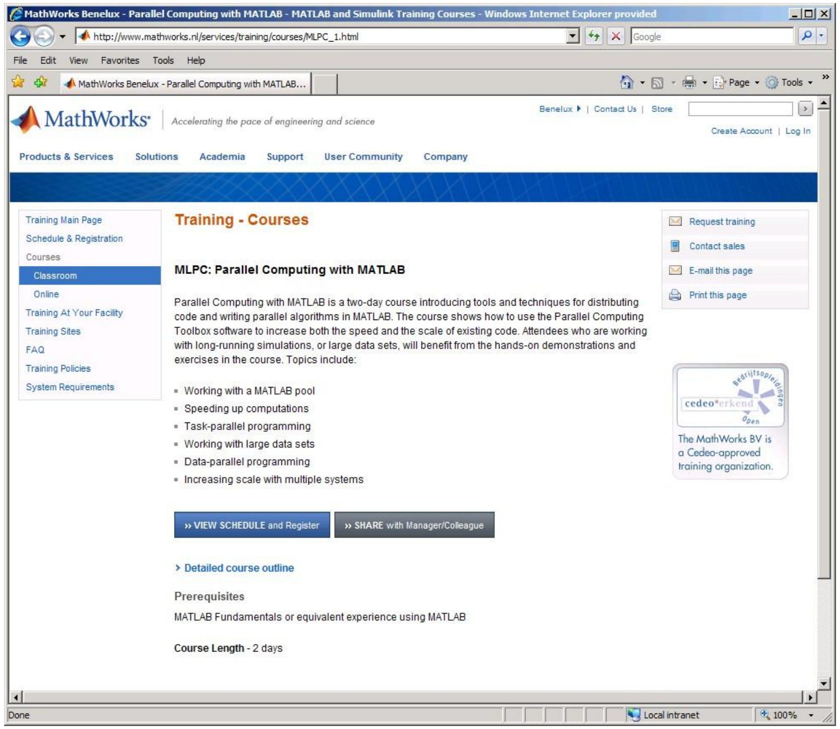
- Used `batch` for off-loading work
- Used `matlabpool` option to off-load and run in parallel
- Used `load` to retrieve worker's workspace





High level parallel programming: takeaways

- Support for parallel computing is built into toolboxes
- Exploit data & task parallelism
- Offload your computations by scheduling



Three Key Takeaways

- 1. If you understand where to put your effort, writing fast and efficient code becomes easy

- 2. You can make the most of your hardware without becoming a programming guru

- 3. If your desktop isn’t enough, you can easily upscale to use GPUs or computer clusters

Agenda

13:30	Introduction
13:45	Part I: Improvements without programming effort
14:30	*Break*
14:50	Part II: High level parallel programming
15:50	*Break*
16:10	Part III: Specialized solutions
16:50	Wrap up
17:00	*Drinks*

Specialized solutions

- **Programming distributed jobs**
- Up-scaling to a cluster
- GPU Computing
- Handling arbitrarily large datasets

Programming parallel applications

Level of control	Required effort
Minimal	None
Some	Straightforward
Extensive	Involved

Programming parallel applications

Level of control	Required effort
Minimal	Built-in into Toolboxes
Some	High Level Programming (parfor, spmd)
Extensive	Low-Level Programming Constructs: (e.g. Jobs/Tasks, MPI-based)

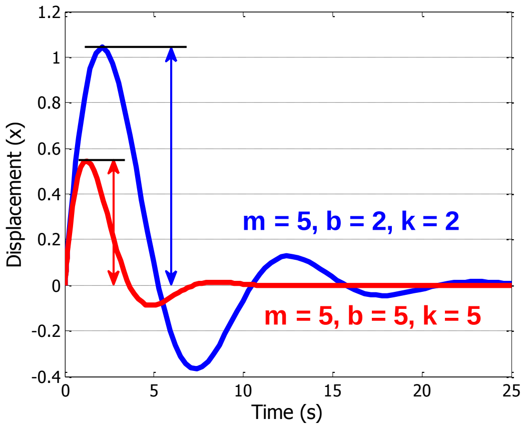
Task-parallel workflows

- `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

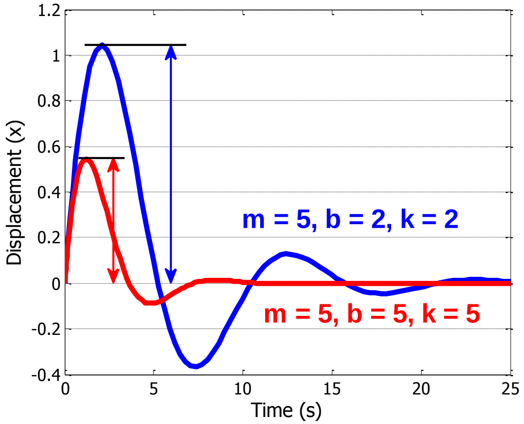
Example: scheduling independent simulations

- 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 scheduler
- 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



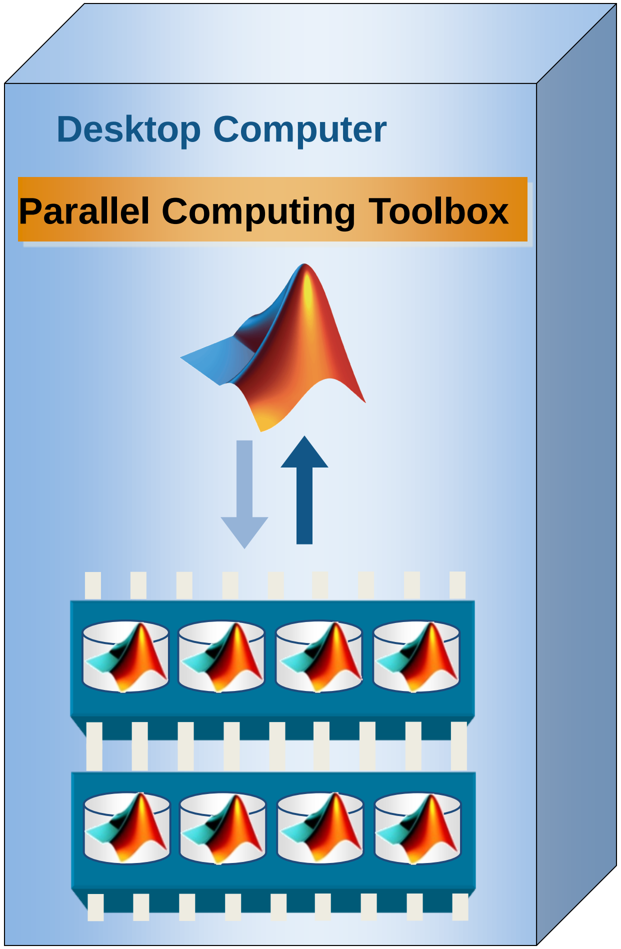
MPI-Based functions for higher control

- High-level abstractions of MPI functions
  - Send, receive, and broadcast any data type in MATLAB: `labSendReceive`, `labBroadcast`, and others
- Automatic bookkeeping
  - Setup: communication, ranks, etc.
  - Error detection: deadlocks and miscommunications
- Pluggable
  - Use any MPI implementation that is binary-compatible with MPICH2

Specialized solutions

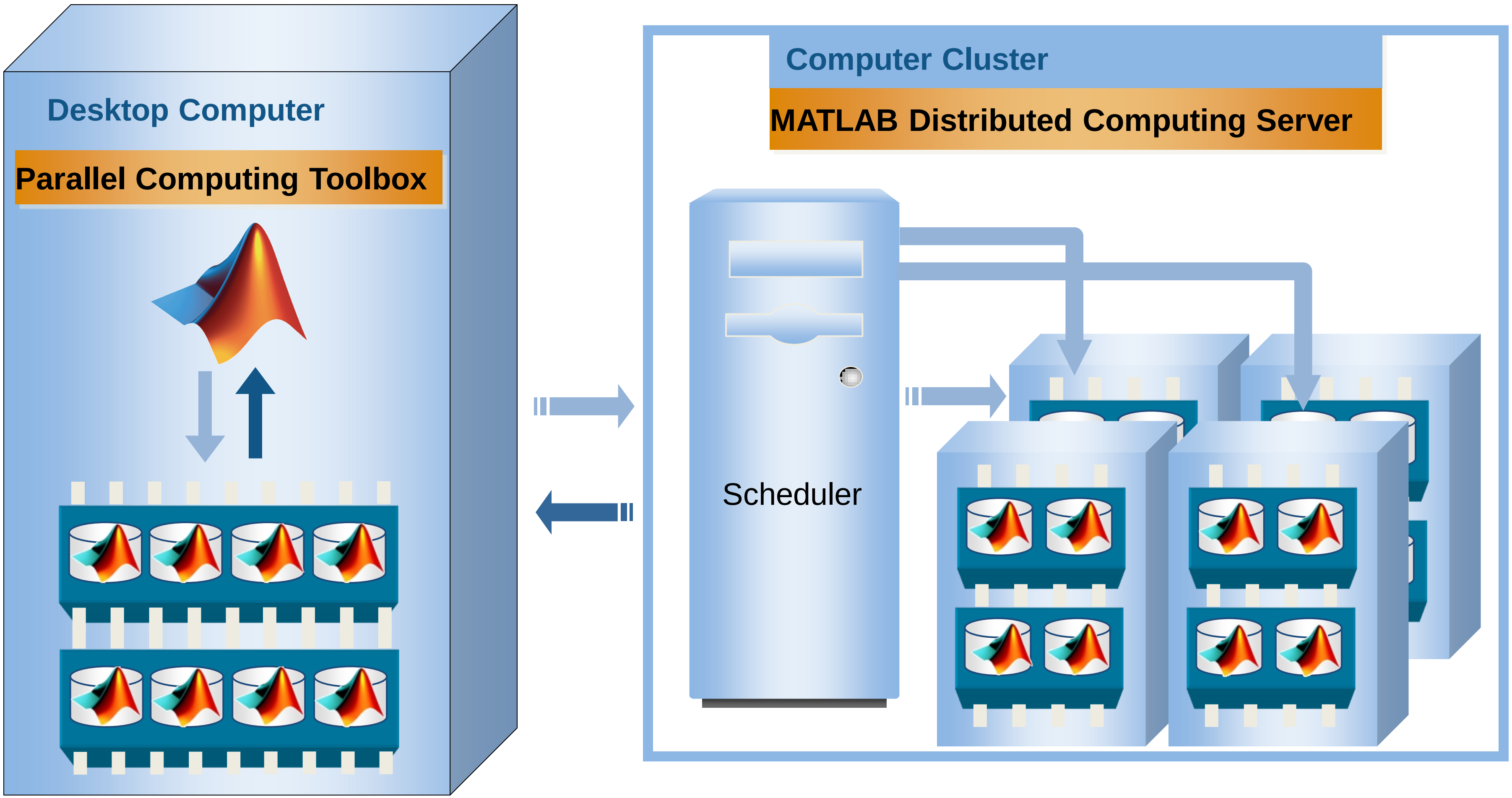
- Programming distributed jobs
- **Up-scaling to a cluster**
- GPU Computing
- Handling arbitrarily large datasets

Run 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



Setting up your cluster client

- Get a profile file from your cluster admin
- Use the menu or parallel.importprofile(‘myclusterprofile’) to get the profile installed on your computer

Support for schedulers

Direct Support

Platform

Microsoft Windows Compute Cluster Server 2003

Windows HPC Server 2008

TORQUE

Open API for others

Grid Engine

UNIVA UD

Specialized solutions

- Programming distributed jobs
- Up-scaling to a cluster
- **GPU Computing**
- Handling arbitrarily large datasets

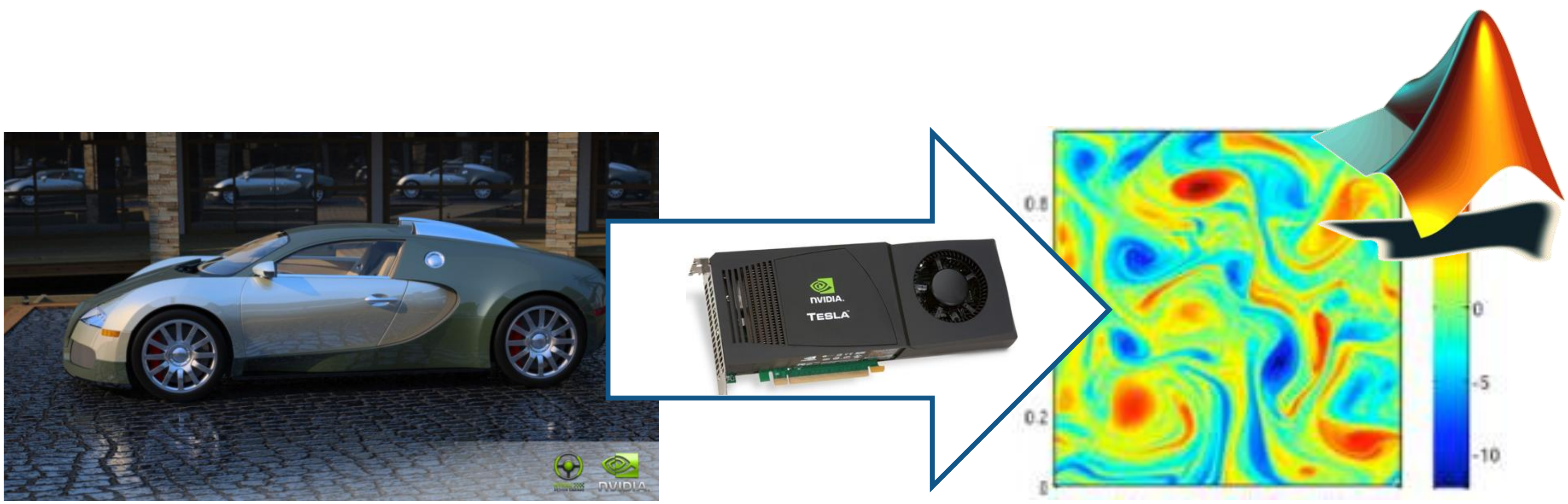
Using MATLAB with dedicated hardware

- Use DSPs for real-time execution
  - ✓ **On target automatic C code generation**
- Deploy on FPGAs
  - ✓ **Synthesizable automatic HDL code generation**
- Use multi-core and multiprocessor computers
  - ✓ **Multithreaded parallel computing**
- Distribute on a cluster
  - ✓ **Distributed computing**
- Use GP-GPUs ◀

Why GPUs now?

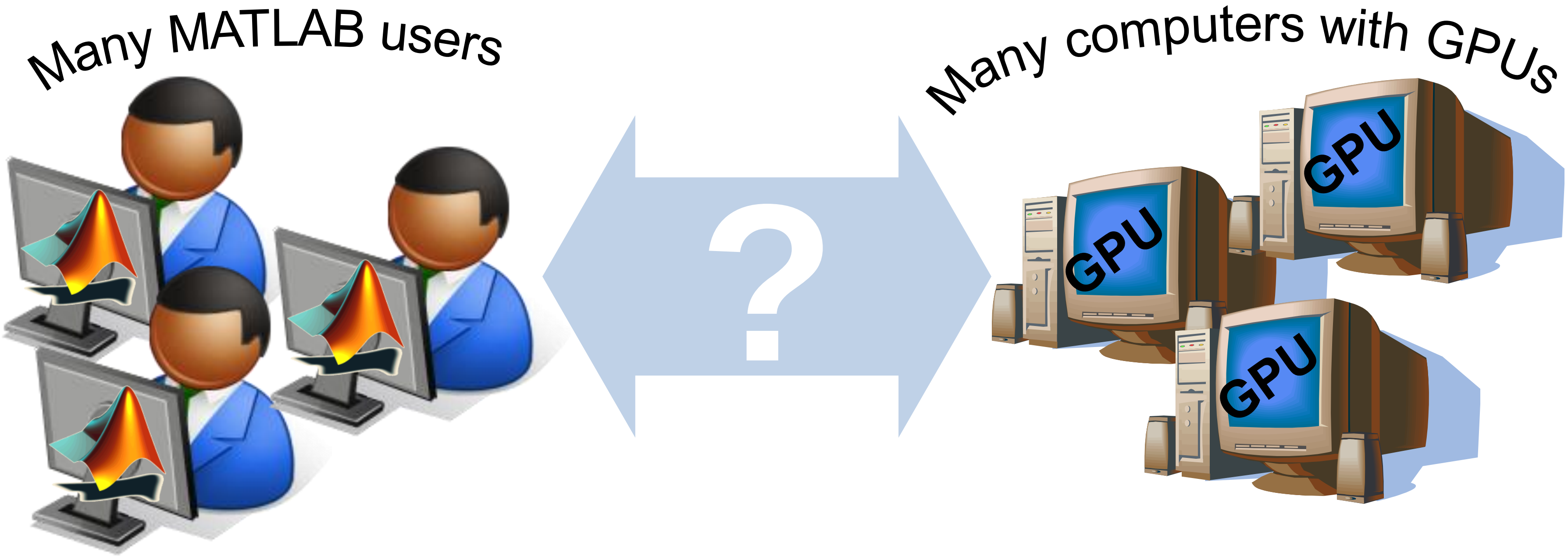
- GPUs are a commodity
- Massively parallel architecture that can speed up intensive computations
- GPU are more generically programmable

From 3D gaming to scientific computing



Speeding up MATLAB using GPUs

- MATLAB users can easily benefit from GPUs
- Support all users from beginners to experts



Programming GPU applications

Level of control	Required effort
Minimal	Minimal
Some	Straightforward
Extensive	Involved

Programming GPU applications

Level of control	Required effort
Minimal	Built-in functions
Some	Straightforward
Extensive	Involved

Invoke built-in MATLAB functions on the GPU

- Accelerate standard (highly parallel) functions
  - More than 100 MATLAB functions are already supported
- Out of the box:
  - No additional effort for programming the GPU
- No accuracy for speed trade-off
  - Double precision floating-point computations

Invoke built-in MATLAB functions on the GPU

(1) Minimal effort, minimal level of control

- Without the GPU: Define an array

```
A = rand(1000,1);
B = rand(1000,1);
```

- Execute a built-in MATLAB function:

```
Y = B\A;
```

Invoke built-in MATLAB functions on the GPU

(1) Minimal effort, minimal level of control

- Define an array on the GPU

```
A = rand(1000,1);
B = rand(1000,1);
A_gpu = gpuArray(A);
B_gpu = gpuArray(B);
```

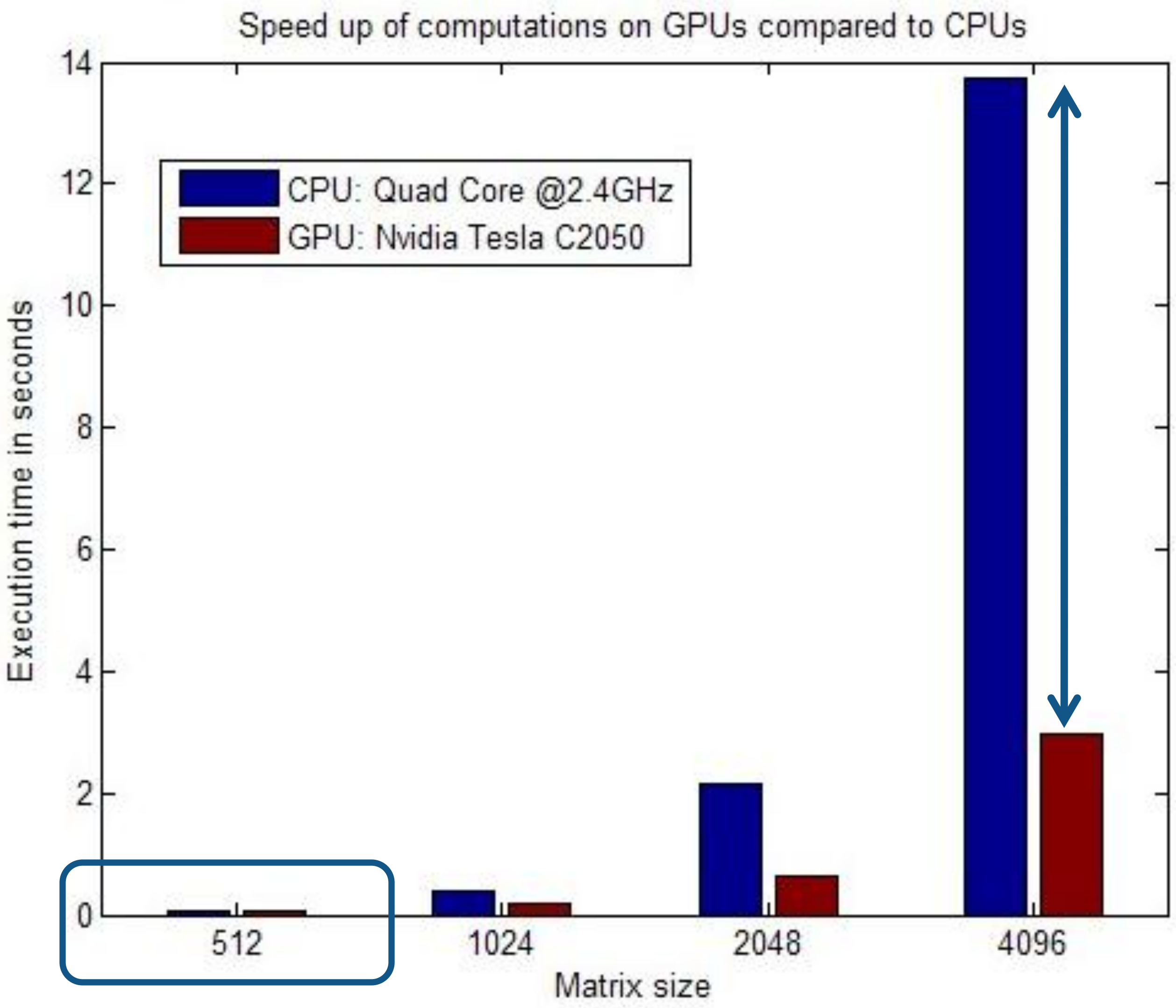
- Execute a built-in MATLAB function:

```
Y_gpu = B_gpu\A_gpu;
```

- Retrieve data from the GPU

```
Y = gather(Y_gpu);
```

Benchmarking A\b on the GPU



Programming GPU applications

Level of control	Required effort
Minimal	Built-in functions
Some	Scalar functions on array data
Extensive	Involved

Run MATLAB scalar functions on the GPU

- Accelerate scalar operations on large arrays
  - Take full advantage on data parallelism
- Out of the box:
  - No additional effort for programming the GPU
- No accuracy for speed trade-off
  - Double precision floating-point computations

Run MATLAB scalar functions on the GPU

(2) Straightforward effort, regular level of control

- MATLAB function that perform element-wise arithmetic

```
function y = TaylorFun(x)
    y = 1 + x.*(1 + x.*(1 + x.*(1 + ...
    x.*(1 + x.*(1 + x.*(1 + x.*(1 + ...
    x.*(1 + ./9)./8)./7)./6)./5)./4)./3)./2);
```

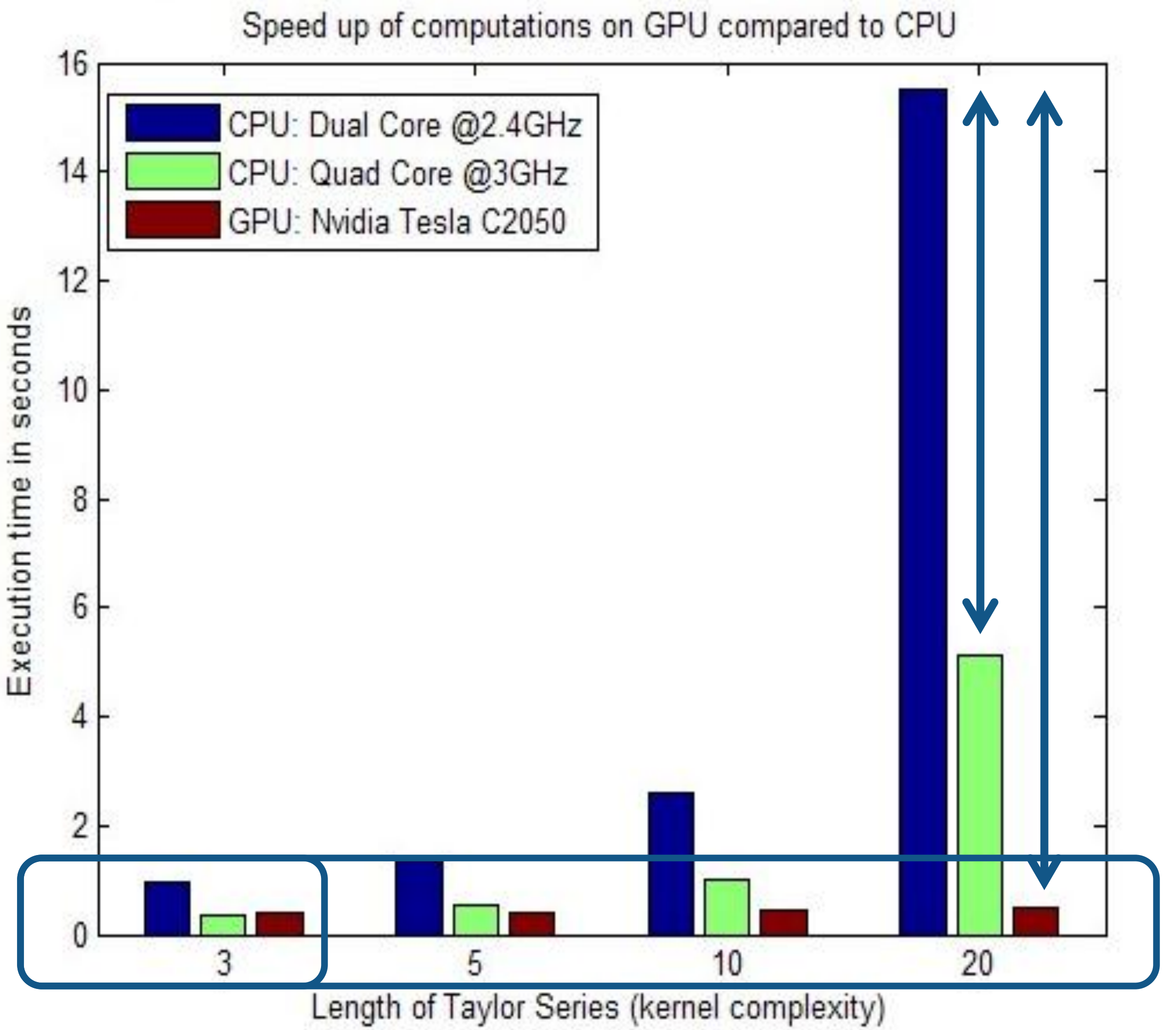
- Load data on the GPU

```
A = rand(1000,1);
A_gpu = gpuArray(A);
```

- Execute the function as GPU kernels

```
result = arrayfun(@TaylorFun, A_gpu);
```

Benchmarking scalar operations on the GPU



Programming GPU applications

Level of control	Required effort
Minimal	Built-in functions
Some	Scalar functions on array data
Extensive	Directly invoke CUDA code

Directly invoke CUDA code from MATLAB

- Benefit from legacy code highly optimized for speed
  - Achieve all the speed improvement that CUDA can deliver
- Use MATLAB as a test environment
  - Generation of test signal
  - Post-processing of results
- Suitable also for non-experts

Invoke CUDA code from MATLAB

(3) Involved effort, extensive level of control

- Compile CUDA (or PTX) code on the GPU

```
nvcc -ptx myconv.cu
```

- Construct the kernel

```
k = parallel.gpu.CUDAKernel('myconv.ptx',
    'myconv.cu');
k.GridSize        = [512 512];
k.ThreadBlockSize = [32 32];
```

- Run the kernel using the MATLAB workspace

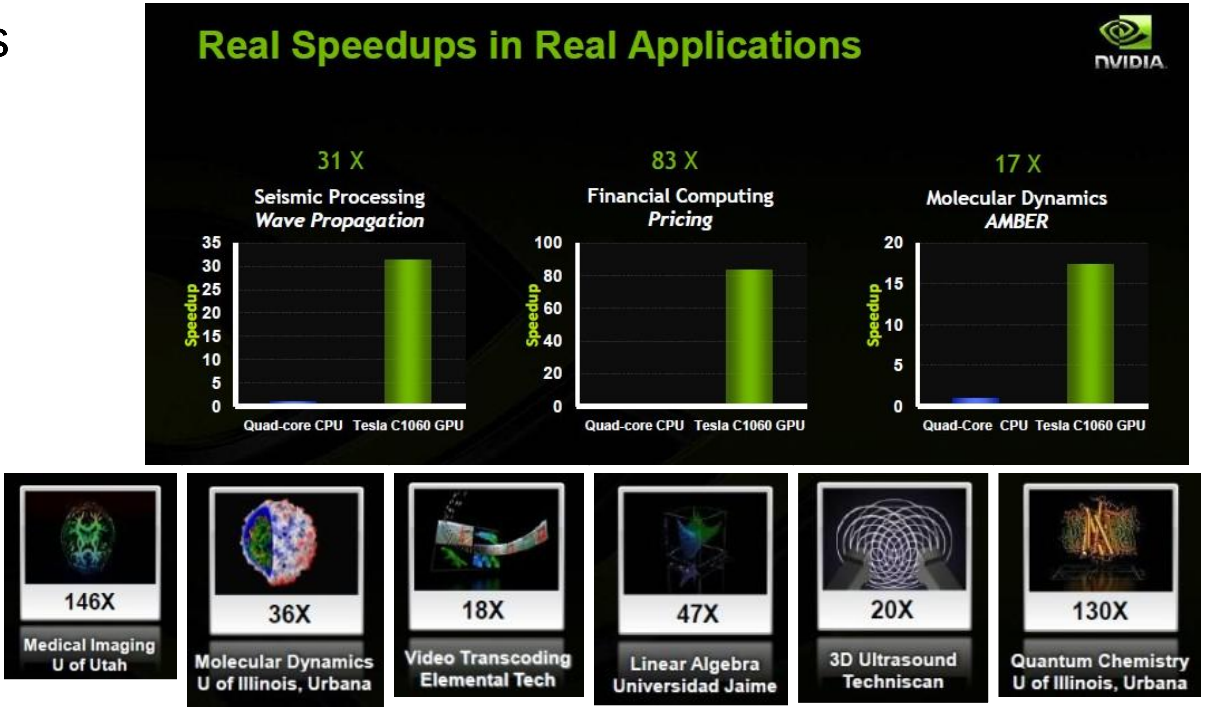
```
o = feval(k, rand(100, 1), rand(100, 1));
```

  or gpu data

```
i1gpu = gpuArray(rand(100, 1, 'single'));
i2gpu = gpuArray(rand(100, 1, 'single'));
ogpu = feval(k, i1gpu, i2gpu);
```

Speeding up applications with GPUs

- Different achievable speedups depending on:
  - Hardware
  - Type of application
  - Programmer skills



Source: NVIDIA

Applications that will run faster on GPUs

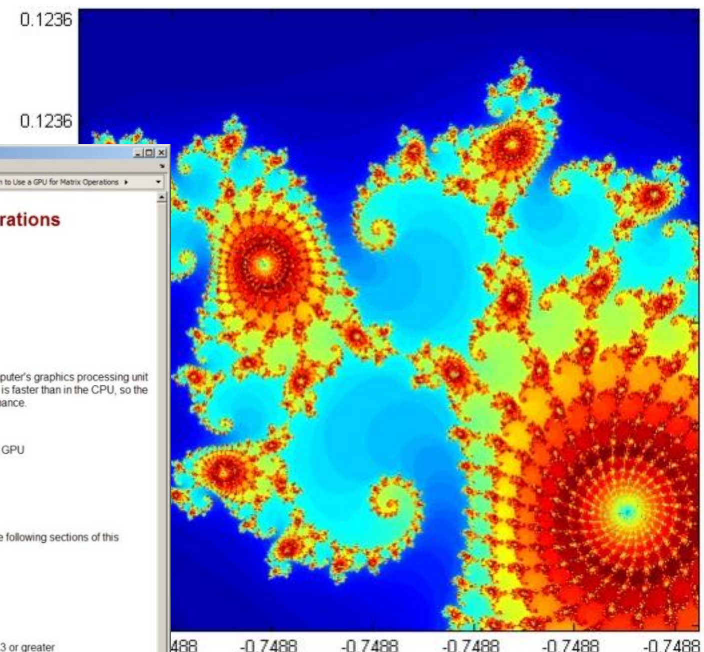
- Massively parallel tasks
- Computationally intensive tasks
- Tasks that have limited kernel size
- Tasks that do not necessarily require double accuracy

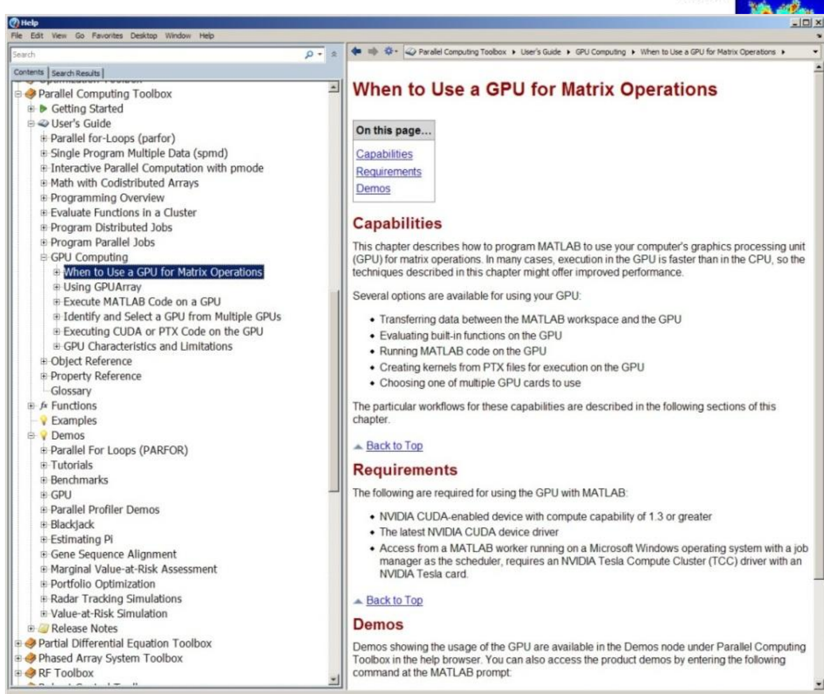
&

All these requirements need to be satisfied!

Can I use it now?

- Use the latest release!
- GPU support is part of Parallel Computing Toolbox
- NVIDIA CUDA capable GPUs
  - With CUDA version 1.3 or greater





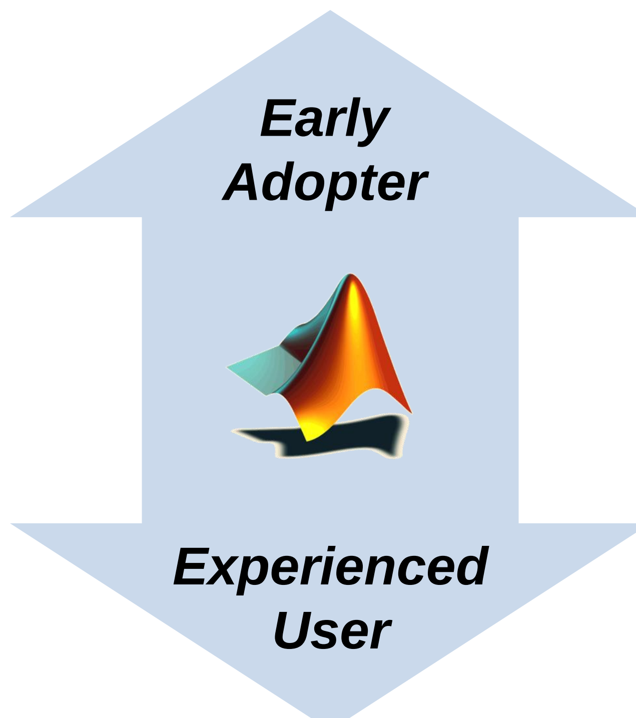
Speeding up MATLAB using GPUs
A stepping stone to accommodate a technology trend

- MATLAB users can easily benefit from GPUs
- Support all users from beginners to experts:

  1. Built-in functions

  2. Define your kernel

  3. Directly invoke CUDA



Specialized solutions

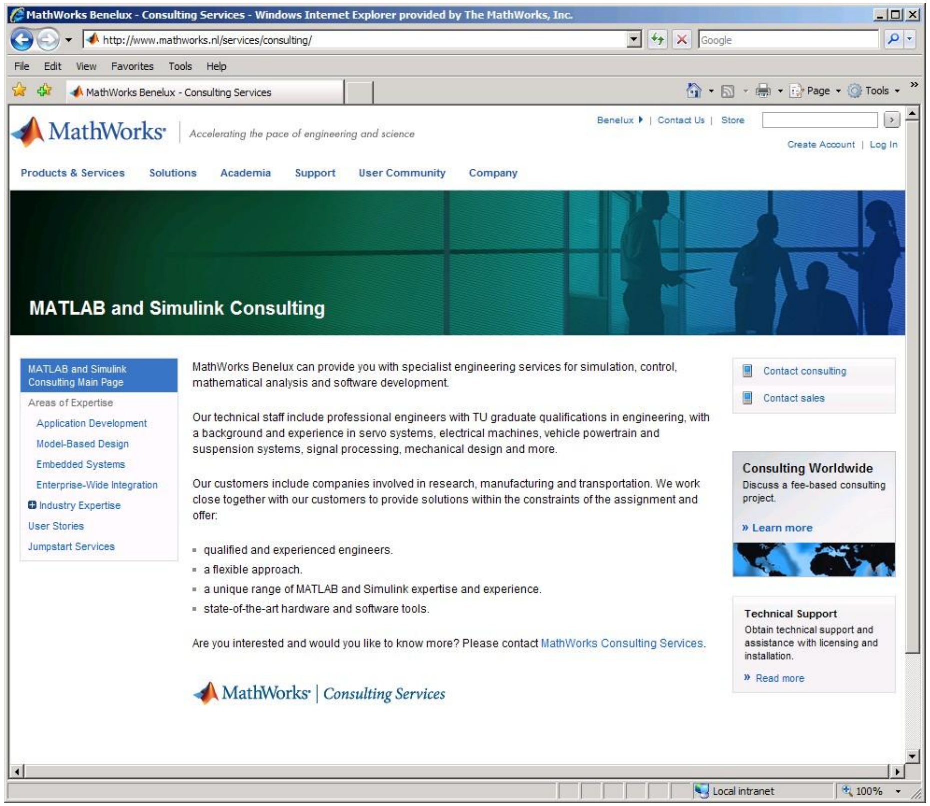
- Programming distributed jobs
- Up-scaling to a cluster
- GPU Computing
- **Handling arbitrarily large datasets**

Handling arbitrarily large datasets

- Large data (GigaBytes)
  - Use 64 bit operating system
  - Install extra RAM
  - Use SPMD to work on a cluster
- Arbitrarily large data (TeraBytes)
  - Use System Objects for streaming data
  - Use Database Toolbox, NetCDF and OpenDAP support to connect directly to databases
  - Develop a database component using our Compiler and Builder JA products

Specialized Solutions: takeaways

- Low level parallel programming for achieving total control on the execution
- Exploit the full capacity of a cluster, without code changes
- GPU computing also for non-experts
- Handle any amount of data



Three Key Takeaways

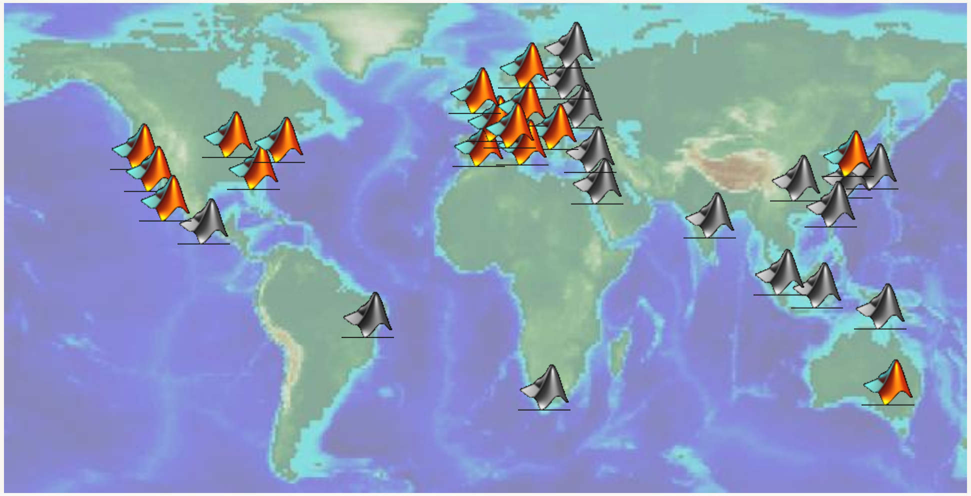
- 1. If you understand where to put your effort, writing fast and efficient code becomes easy
- 2. You can make the most of your hardware without becoming a programming guru
- 3. If your desktop isn't enough, you can easily upscale to use GPUs or computer clusters

Conclusions

The MathWorks Mission

Accelerating the pace of engineering and science

- 2400 employees
- 900+ software developers
- Benelux-office in Eindhoven
- >1 mio users worldwide



Earth's topography on an equidistant cylindrical projection, using the MATLAB Mapping Toolbox

Key Industries

- Aerospace and defense
- Automotive
- Biotech and pharmaceutical
- Communications
- Education
- Electronics and semiconductors
- Energy production
- Financial services
- Industrial automation and machinery

Deeply Rooted in Education

- 3500+ universities around the world
- 1200+ MATLAB and Simulink based books
- Academic support for research, fellowships, student competitions, and curriculum development

> "Everyone that comes in as a new hire already knows MATLAB, because they all had it in college. The learning curve is significantly lessened as a result."
>
> **Jeff Corn,**
> **Chief of Engineering Projects Section,**
> **U.S. Air Force**

Benefits for Industry:

- Every year, tens of thousands of engineers enter the workforce with MathWorks product skills and experience.
- Students learn theory and techniques while using MATLAB and Simulink.

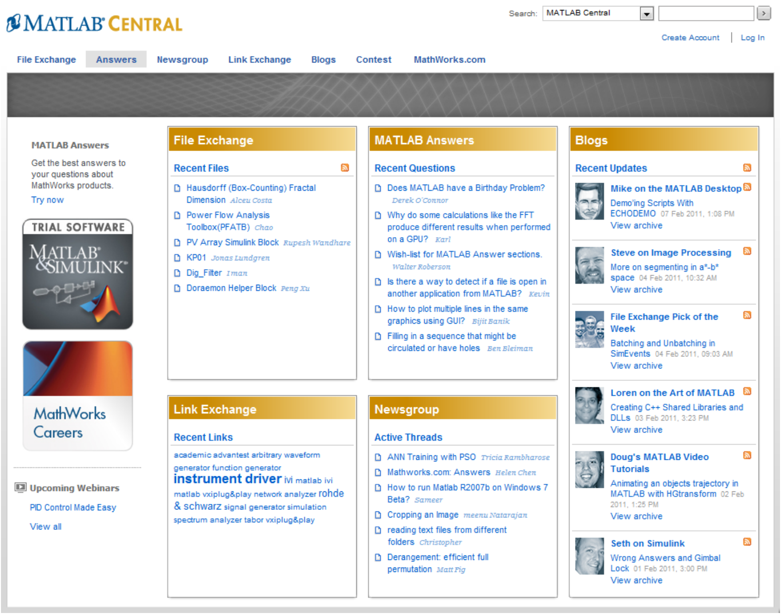



MathWorks® | *Training Services*

- **MATLAB Fundamentals (3 days)** **May 1-3**
- **Parallel Computing with MATLAB** **Nov 22-23**
- **MATLAB Programming Techniques** **June 5**

MATLAB Central

- Community for MATLAB and Simulink users
- Over 1 million visits per month
- File Exchange
  - Upload/download access to free files including MATLAB code, Simulink models, and documents
  - Ability to rate files, comment, and ask questions
  - More than 12,500 contributed files, 300 submissions per month, 50,000 downloads per month
- Newsgroup
  - Web forum for technical discussions about MathWorks products
  - More than 300 posts per day
- Blogs
  - Commentary from engineers who design, build, and support MathWorks products
  - Open conversation at blogs.mathworks.com



Based on February 2011 data

Free on-demand Webinars: a useful resource

www.mathworks.nl/webinars

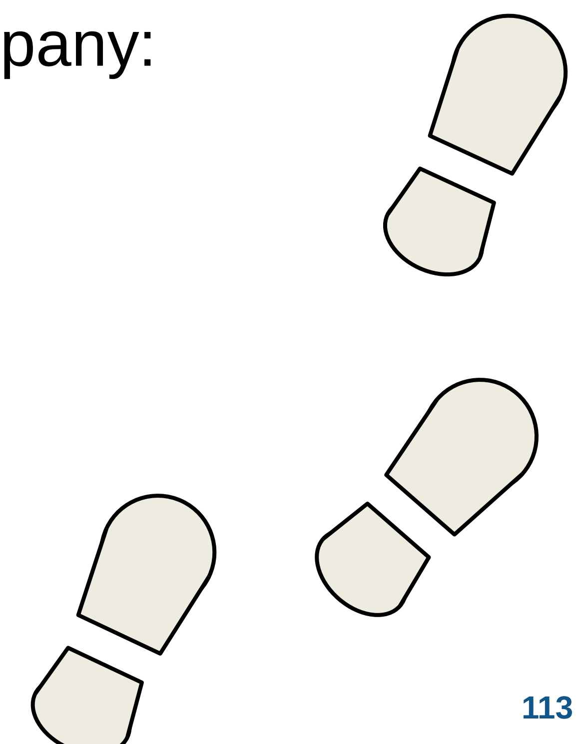
Parallel Computing with MATLAB on Multicore Desktops and GPUs 45:12

Speeding Up MATLAB Applications 53:38

Speeding Up Optimization Problems Using Parallel Computing 55:41

Next Steps…

- Questions: Contact any of us after the seminar
- For more information: look at www.mathworks.nl
- To start evaluating MATLAB in your company: Rob.Heijmans@mathworks.nl



Thank you for joining Speeding up MATLAB and Handling Large Data Sets

Mathijs Faase
Rob Heijmans
Zoetermeer - 26th April 2012

Please fill out your evaluation form.