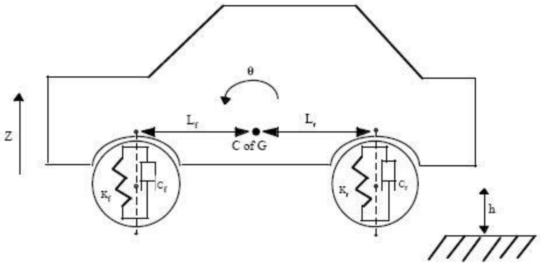
Parallel Computing and GPU Programming with MATLAB



MathWorks Seminar
University of Oldenburg
Feb 19, 2013

Michael Glaßer
Kremena Radeva

Application Engineering
Education Sales

Agenda

13:30	Welcome and Introduction
13:45	Introduction to Parallel Computing with MATLAB
	MATLAB–extensions with built-in support for Parallel Computing
14:15	Interactive development of task- and data-parallel Algorithms
15:15	*Coffee Break*
15:30	GPU programming with MATLAB
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	Q&A Session
17:00	*End of Seminar*

Your MathWorks Team Today

Kremena Radeva	*Account Manager*
Michael Glaßer	*Application Engineer*

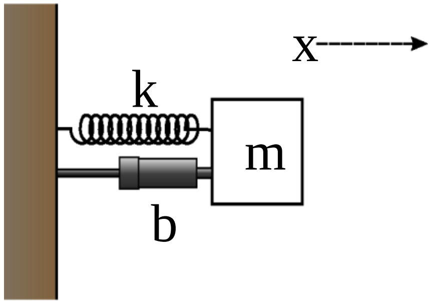
Agenda

13:30	Welcome and Introduction
13:45	Introduction to Parallel Computing with MATLAB
	MATLAB–extensions with built-in support for Parallel Computing
14:15	Interactive development of task- and data-parallel Algorithms
15:15	*Coffee Break*
15:30	GPU programming with MATLAB
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	Q&A Session
17:00	*End of Seminar*

Example: Parameter Sweep of ODEs

- Solve a 2nd order ODE

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

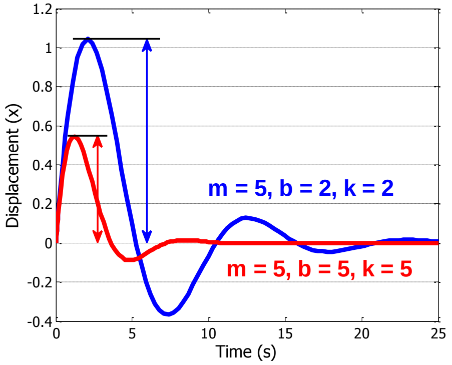


- Simulate with different values for ***b*** and ***k***

- Record peak value for each run

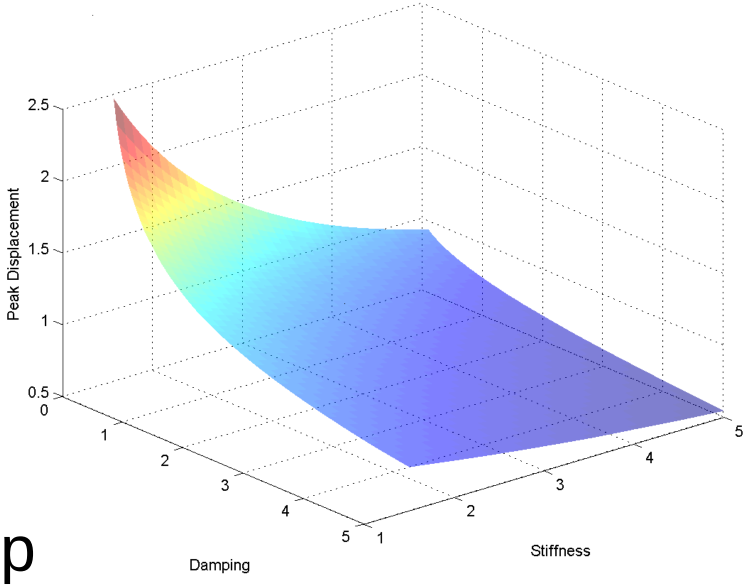
- Plot results

- Time in serial and in parallel mode



Summary of Example

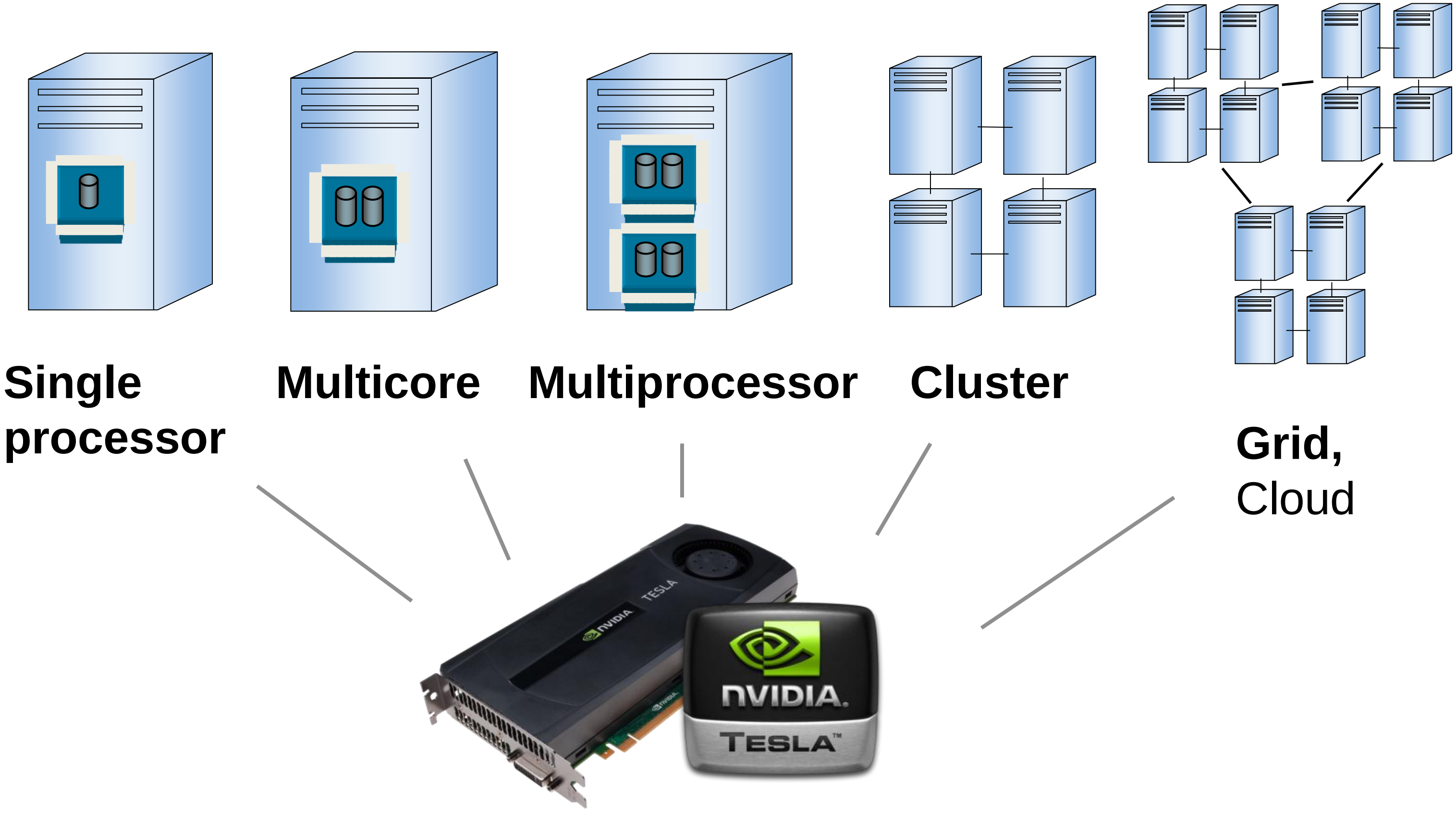
- Mixed task-parallel and serial code in the same function
- Ran loops on a pool of MATLAB resources
- Used Code Analyser to help in converting existing `for`-loop into `parfor`-loop



Solving Big Technical Problems

Challenges	You could…		Solutions
Long running Computationally intensive	Wait	⇒	Larger Compute Pool (e.g. More Processors)
Large data set	Reduce size of problem	⇒	Larger Memory Pool (e.g. More Machines)

High-Performance Hardware is Available



Single processor

Multicore

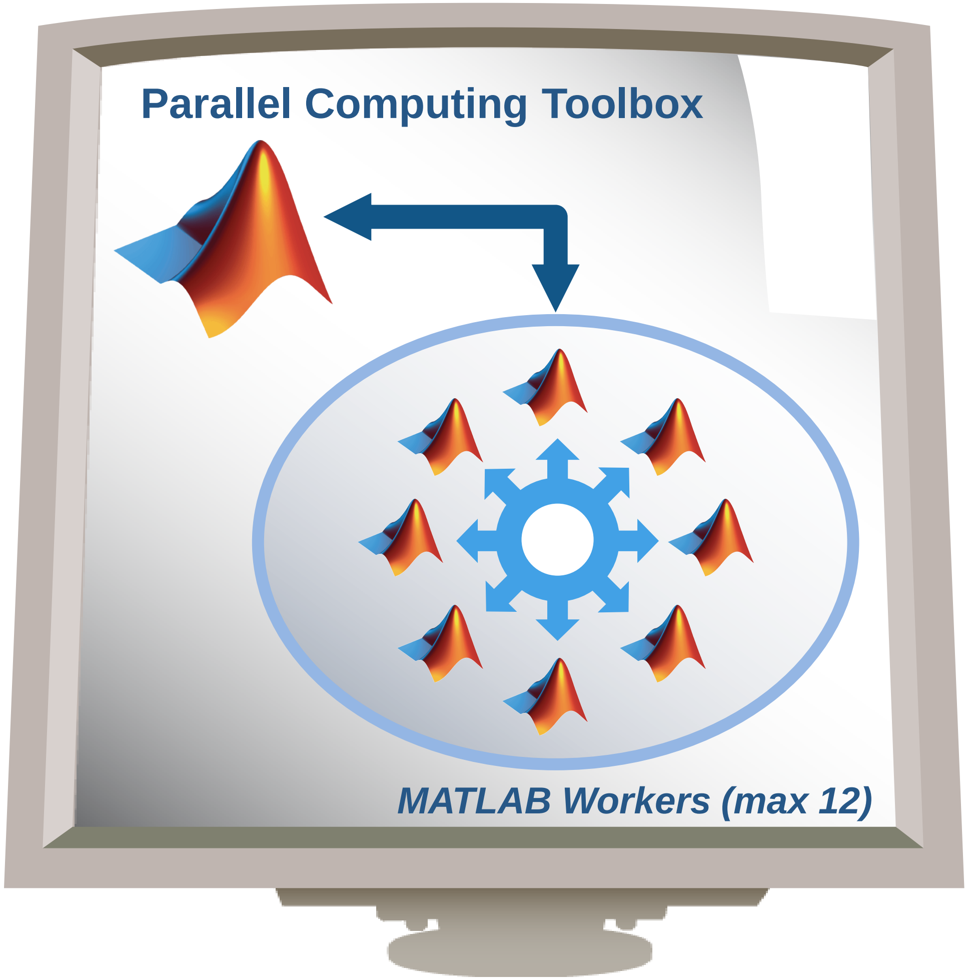
Multiprocessor

Cluster

Grid,
Cloud

***GP*GPU**

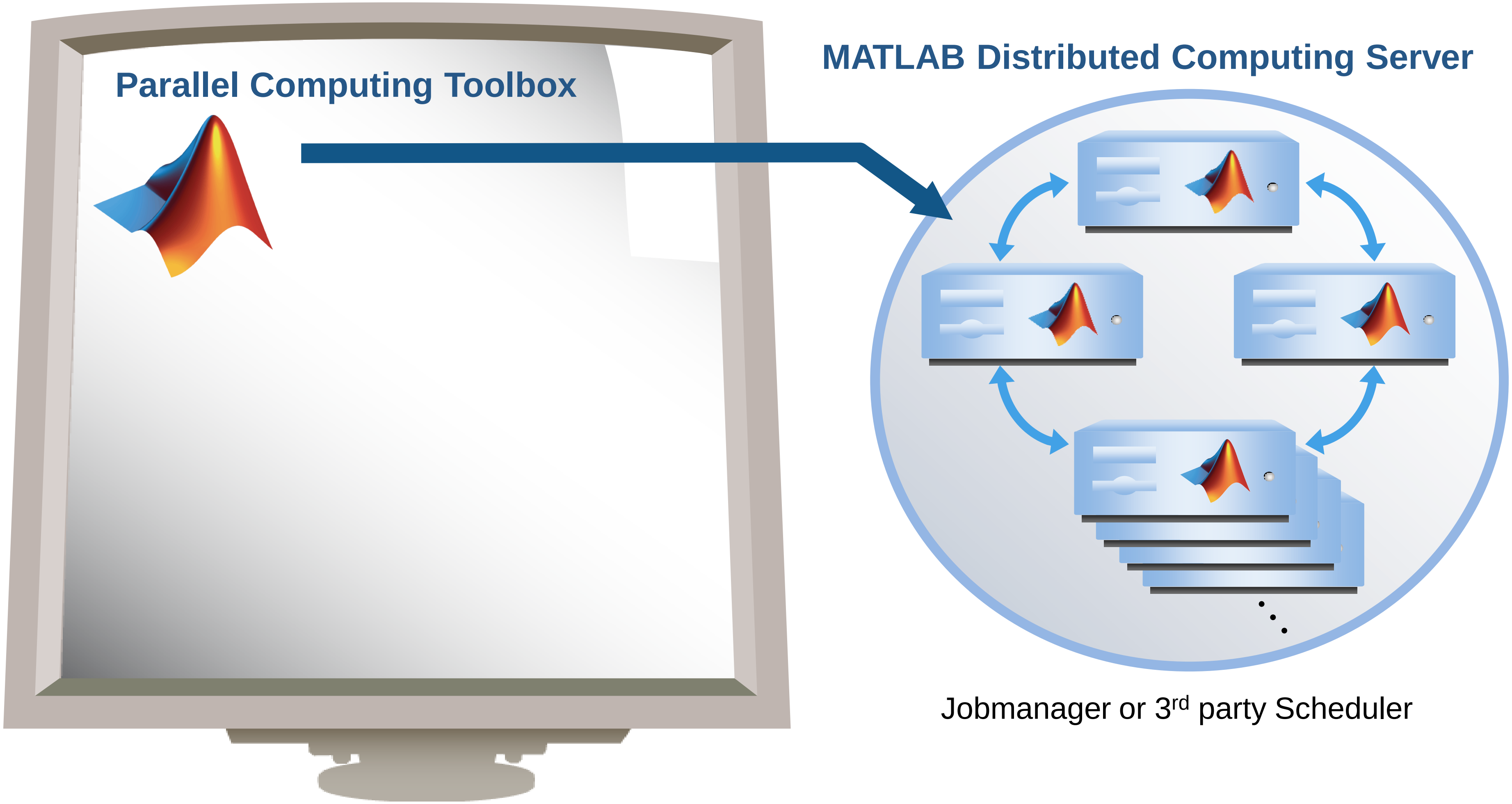
Parallel Computing with MATLAB



User’s Desktop

- Easily experiment with explicit parallelism on multicore machines
- Rapidly develop parallel applications on local computer
- Take full advantage of desktop power, incl. GPU(s)
- Separate computer cluster not required

Parallel Computing with MATLAB



Parallel Computing Toolbox

MATLAB Distributed Computing Server

Jobmanager or 3rd party Scheduler

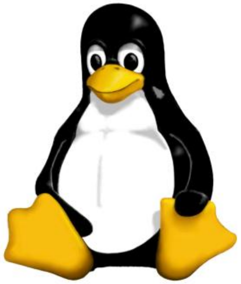
User's Desktop

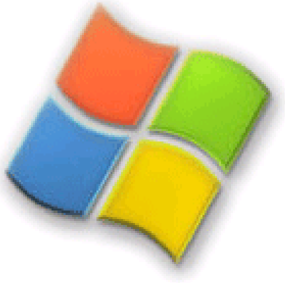
Compute Cluster / Grid / Cloud

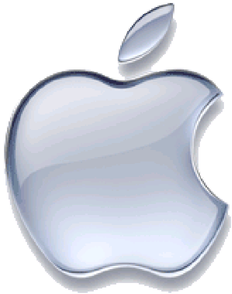
Why scale up to a cluster?

- Solve larger, computationally-intensive problems with more processing power
- Solve memory-intensive problems
- Schedule computations to offload from your local machine

Supported on All Platforms That Support MATLAB







Job Schedulers

- MathWorks Job Scheduler:
  turn-key solution for MATLAB-only clusters

- Direct support for existing scheduler:
  MDCS is simply another application

Platform

PBS

Windows HPC Server 2008

TORQUE

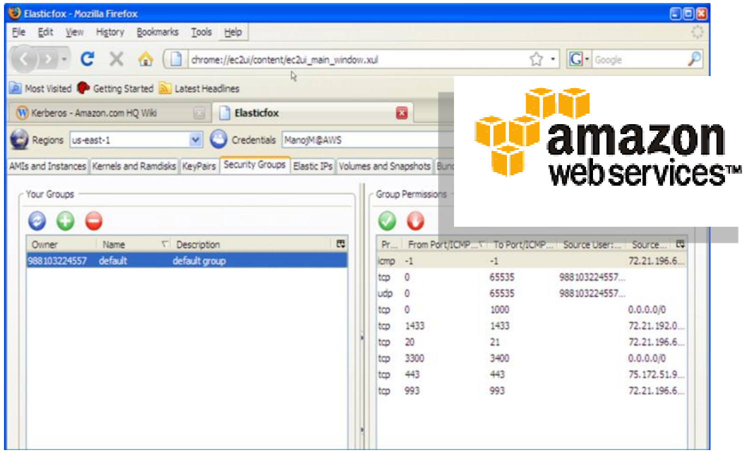
- Open API to support other schedulers

Grid Engine

UNIVA UD

Cloud Computing:
Dynamic Computing and Storage Resources

- Characteristics
  - Scalable (“elastic” resource)
  - Virtualization
  - Service over the Internet



chrome://ec2ui/content/ec2ui_main_window.xul

Regions: us-east-1

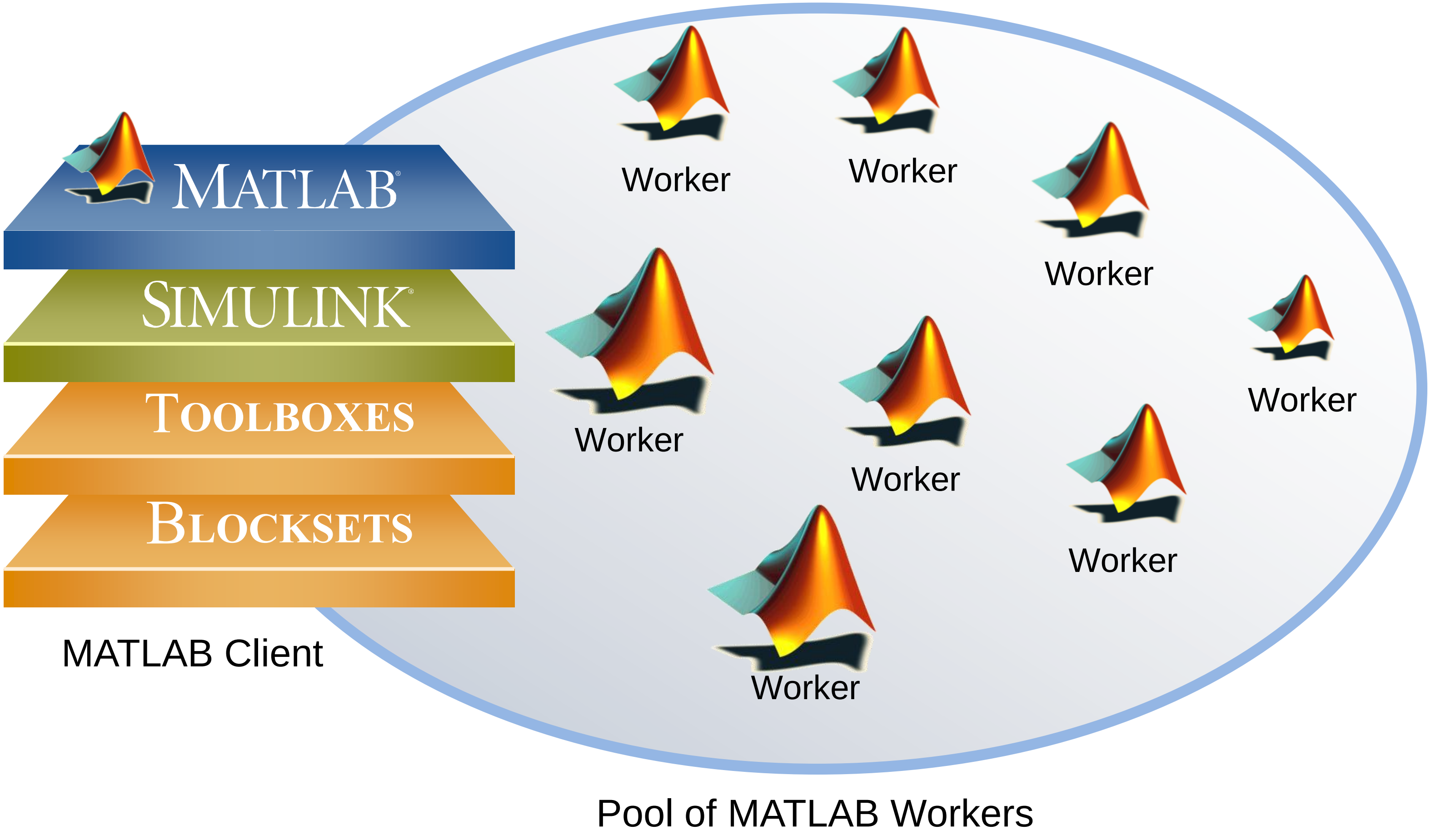
Credentials: ManojM@AWS

Owner	Name	Description
988103224557	default	default group

Pr...	From Port/ICMP...	To Port/ICMP...	Source User:...	Source...
icmp	-1	-1		72.21.196.6...
tcp	0	65535	988103224557...	
udp	0	65535	988103224557...	
tcp	0	1000		0.0.0.0/0
tcp	1433	1433		72.21.192.0...
tcp	20	21		72.21.196.6...
tcp	3300	3400		0.0.0.0/0
tcp	443	443		75.172.51.9...
tcp	993	993		72.21.196.6...

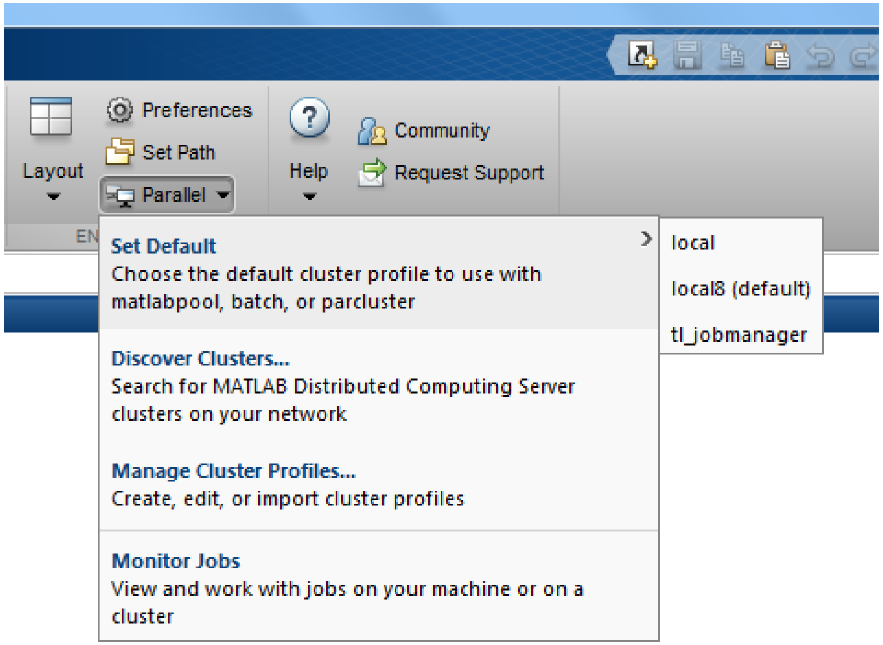
- Benefits
  - Scale computing capacity as needed
  - Purchase and maintenance of cluster is not required
  - Choose desired configuration (e.g., CPU, memory)

Going Beyond Serial Applications



Configurations

- Save environment-specific parameters for your cluster
- Benefits
  - Enter cluster information only once
  - Modify configurations without changing MATLAB code
  - Apply multiple configurations when running within same session



Set Default
Choose the default cluster profile to use with matlabpool, batch, or parcluster

Discover Clusters...
Search for MATLAB Distributed Computing Server clusters on your network

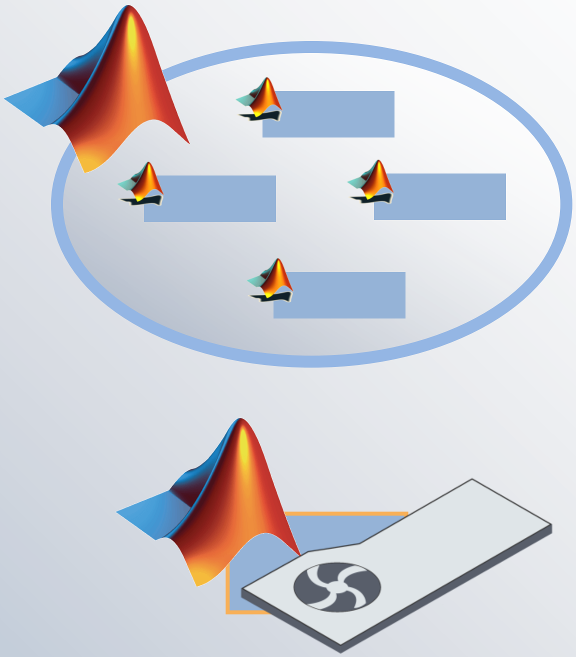
Manage Cluster Profiles...
Create, edit, or import cluster profiles

Monitor Jobs
View and work with jobs on your machine or on a cluster

Parallel Computing with MATLAB enables you to …

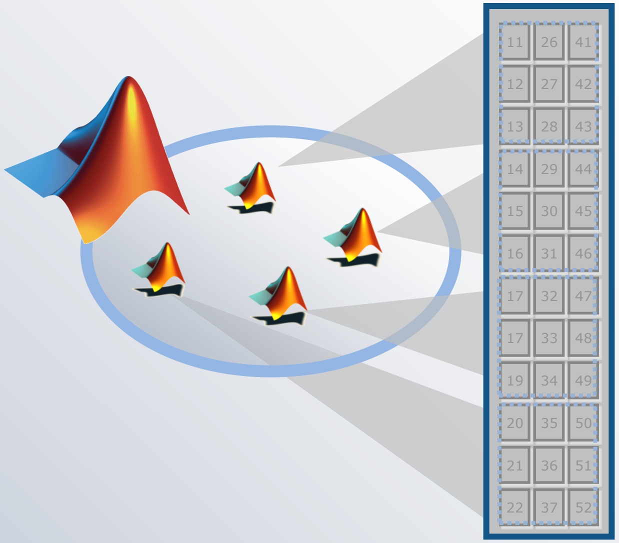
Larger Compute Pool

Speed up Computations

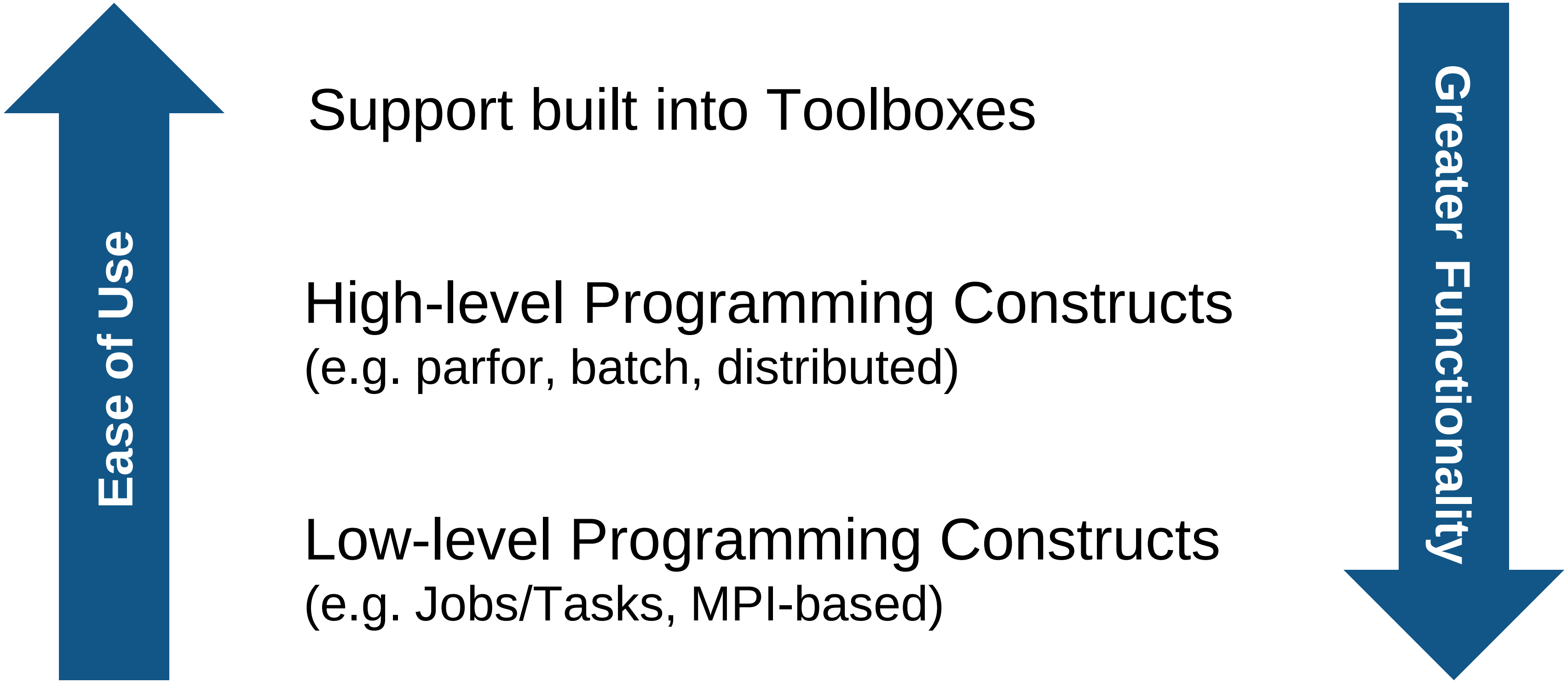


Larger Memory Pool

Work with Large Data



Three levels of integration



Support built into Toolboxes

High-level Programming Constructs
(e.g. parfor, batch, distributed)

Low-level Programming Constructs
(e.g. Jobs/Tasks, MPI-based)

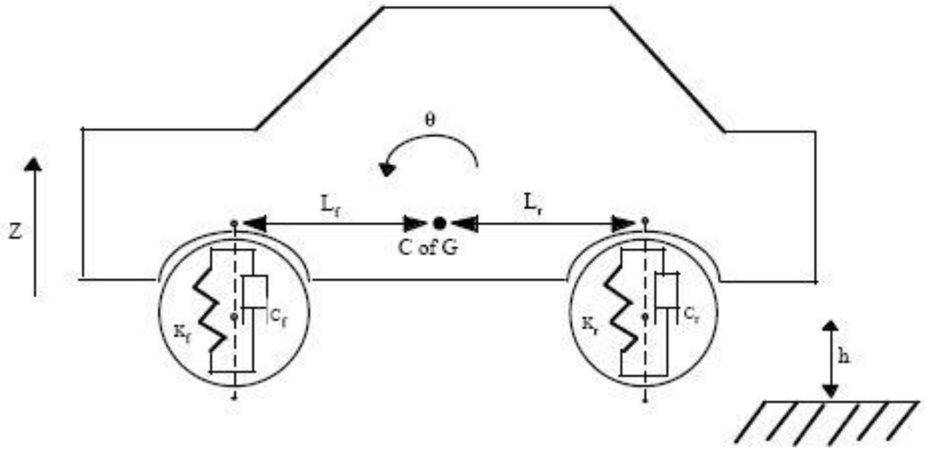
Agenda

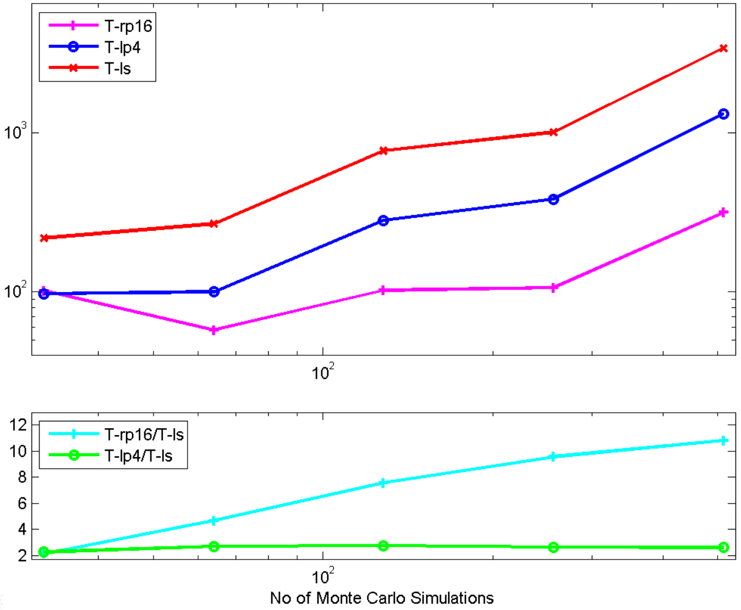
13:30	**Welcome and Introduction**
13:45	**Introduction to Parallel Computing with MATLAB**
	MATLAB–extensions with built-in support for Parallel Computing
14:15	**Interactive development of task- and data-parallel Algorithms**
15:15	***Coffee Break***
15:30	**GPU programming with MATLAB**
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	**Q&A Session**
17:00	***End of Seminar***

Example: Optimization combined with Monte Carlo Simulation *using built in parallelization*

- Design of suspension system to minimize mean and standard deviation of acceleration

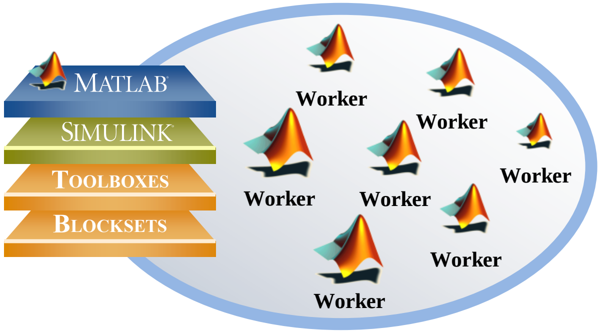
- Account for uncertainty in mass distribution via Monte Carlo simulation





Other Tools Providing Parallel Computing Support

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



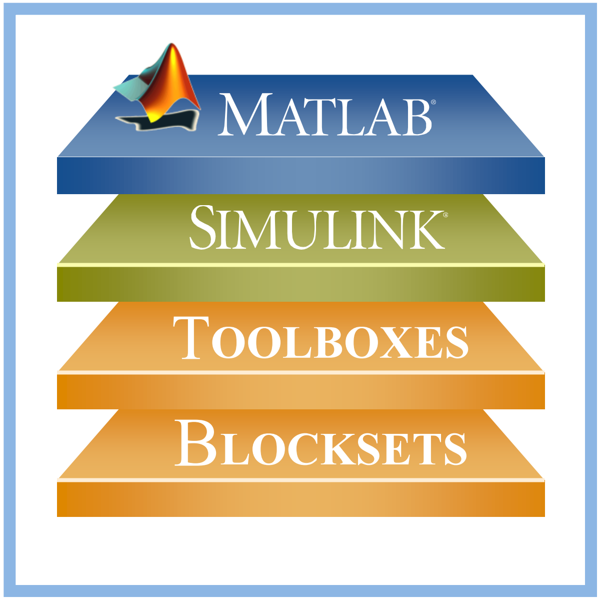
http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html

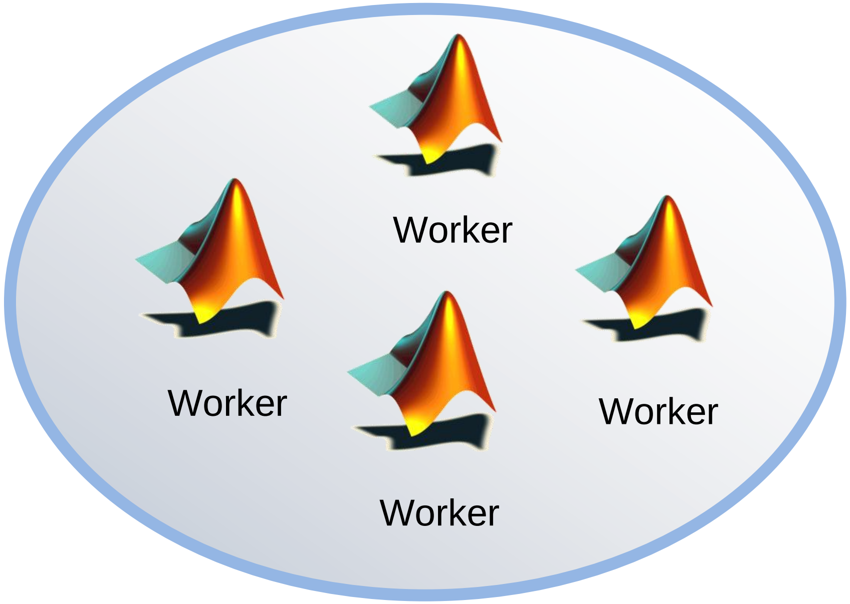
Directly leverage functions in Parallel Computing Toolbox

Agenda

13:30	**Welcome and Introduction**
13:45	**Introduction to Parallel Computing with MATLAB**
	MATLAB–extensions with built-in support for Parallel Computing
14:15	**Interactive development of task- and data-parallel Algorithms**
15:15	***Coffee Break***
15:30	**GPU programming with MATLAB**
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	**Q&A Session**
17:00	***End of Seminar***

Task Parallel Applications



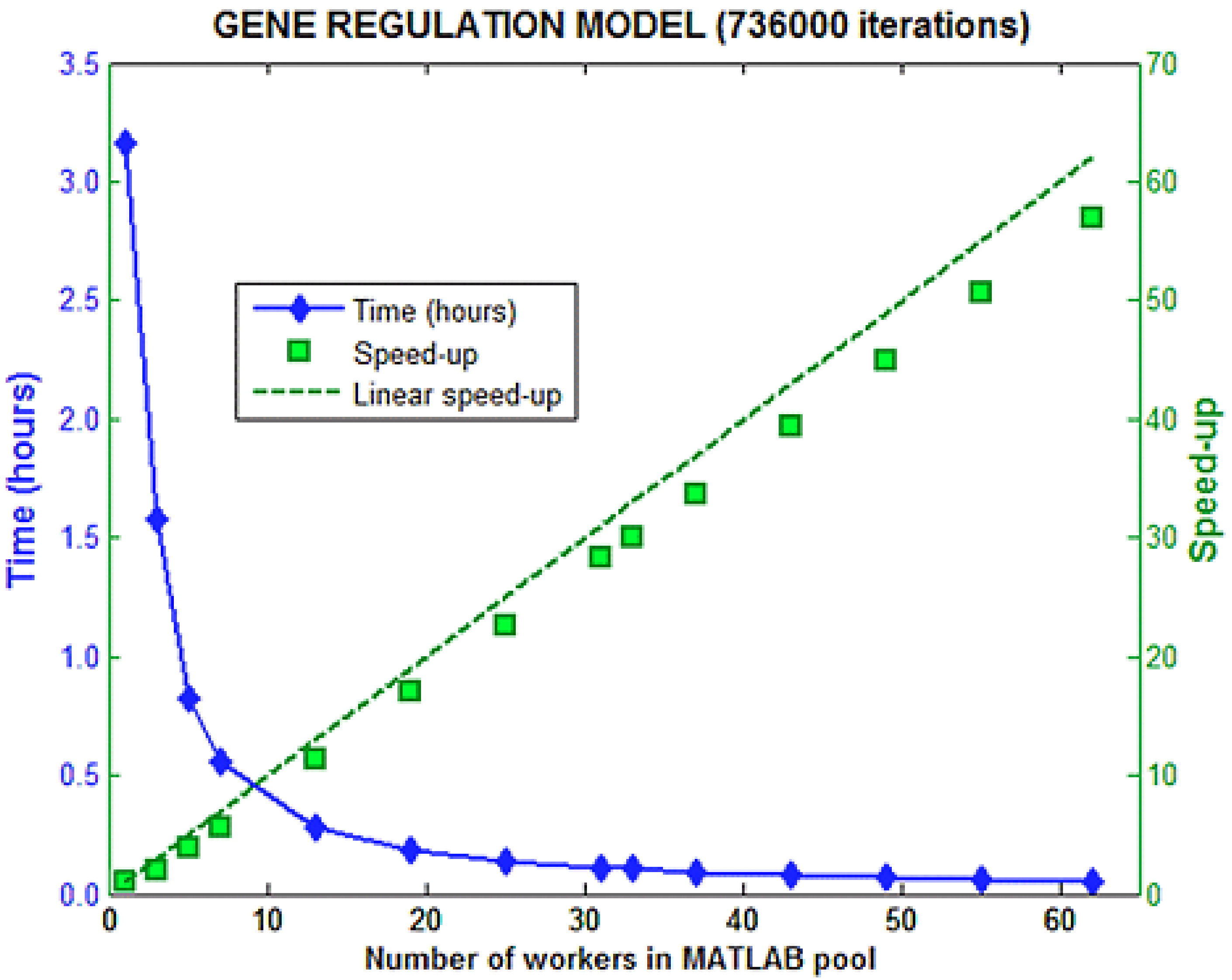


Task 1	Task 2	Task 3	Task 4

Time

Time

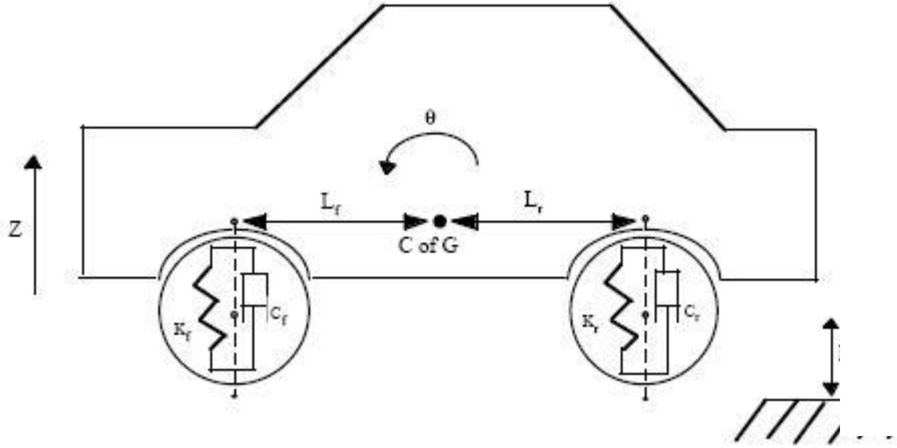
Benchmark: Multiple Independent Simulations

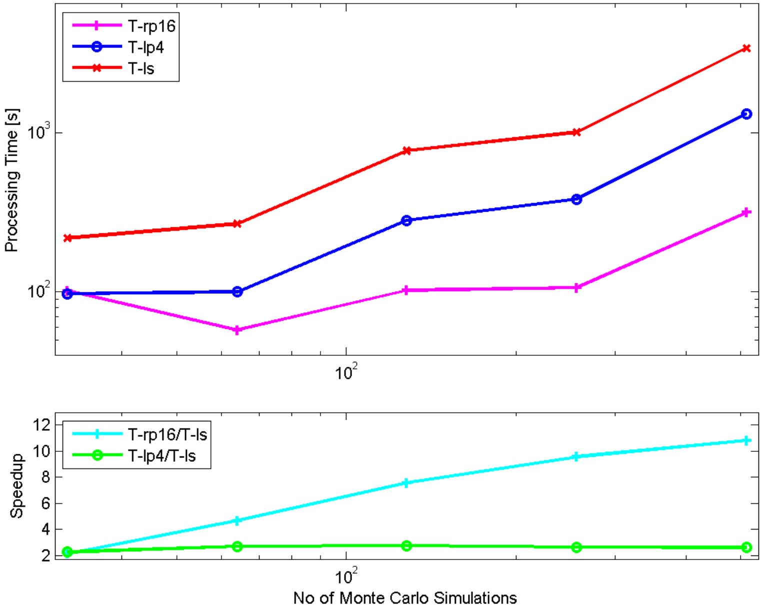


Example: Optimization combined with Monte Carlo Simulation *using a parallel for loop*

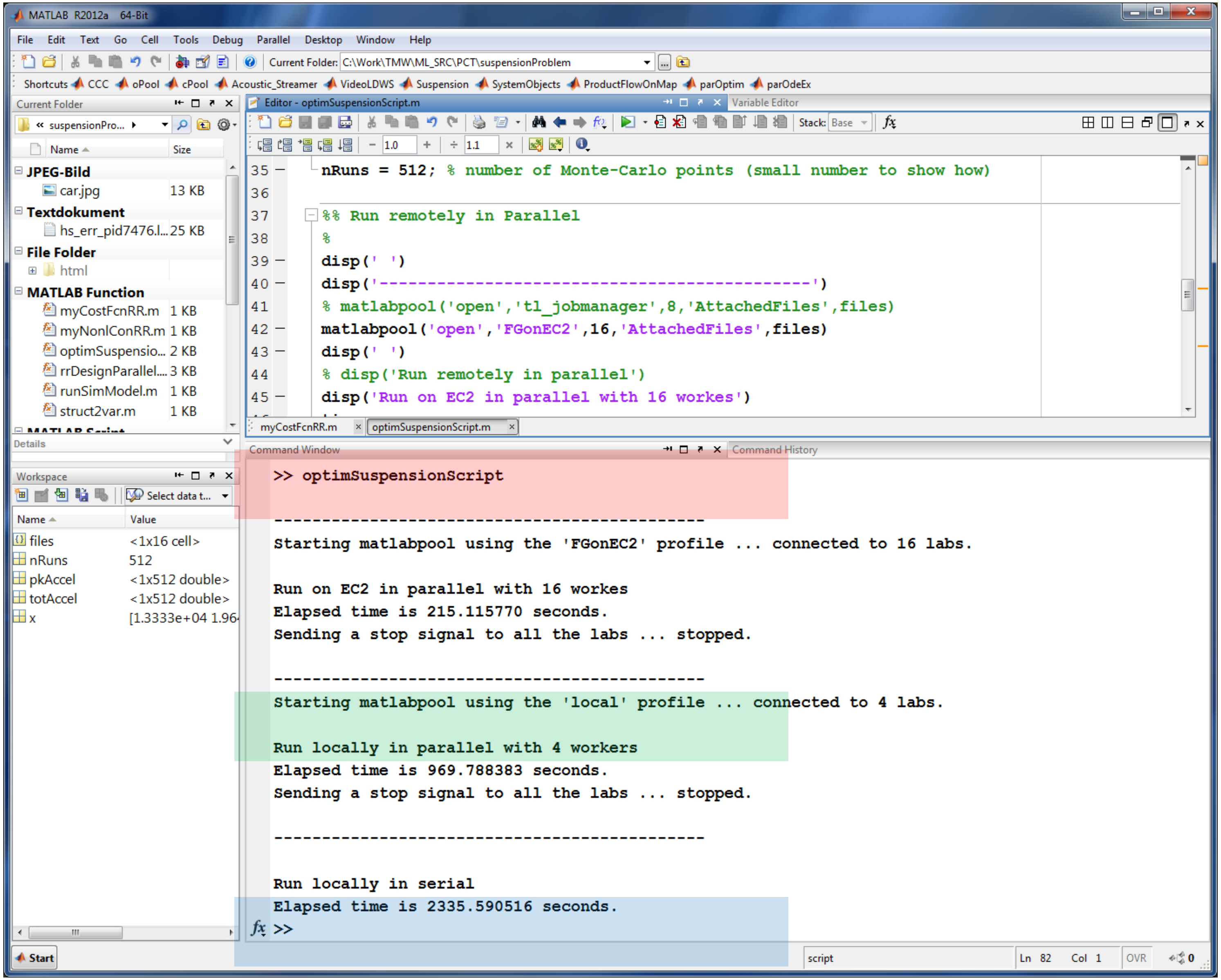
- Design of suspension system to minimize mean and standard deviation of acceleration

- Account for uncertainty in mass distribution via Monte Carlo simulation





Results from Amazon EC2


MATLAB R2012a 64-Bit
File Edit Text Go Cell Tools Debug Parallel Desktop Window Help
Current Folder: C:\Work\TMW\ML_SRC\PCT\suspensionProblem
Shortcuts CCC oPool cPool Acoustic_Streamer VideoLDWS Suspension SystemObjects ProductFlowOnMap parOptim parOdeEx
Current Folder
Editor - optimSuspensionScript.m
Variable Editor
Stack: Base
myCostFcnRR.m
optimSuspensionScript.m
Details
Workspace
Select data t...
Command Window
Command History
Start
script
Ln 82 Col 1 OVR

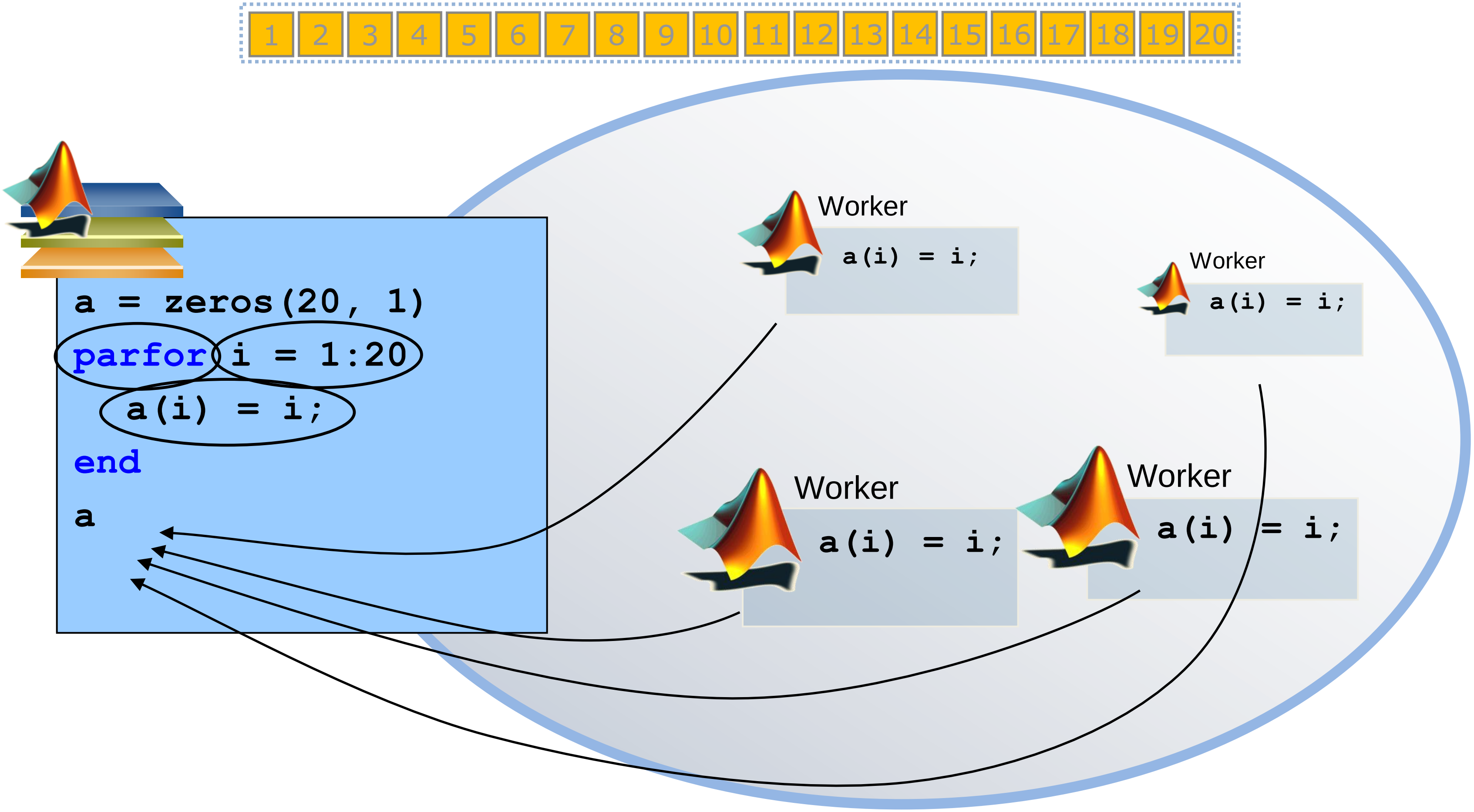
Name	Size
JPEG-Bild	
car.jpg	13 KB
Textdokument	
hs_err_pid7476.l...	25 KB
File Folder	
html	
MATLAB Function	
myCostFcnRR.m	1 KB
myNonlConRR.m	1 KB
optimSuspensio...	2 KB
rrDesignParallel....	3 KB
runSimModel.m	1 KB
struct2var.m	1 KB

```matlab
nRuns = 512; % number of Monte-Carlo points (small number to show how)

%% Run remotely in Parallel
%
disp(' ')
disp('---------------------------------------------')
% matlabpool('open','tl_jobmanager',8,'AttachedFiles',files)
matlabpool('open','FGonEC2',16,'AttachedFiles',files)
disp(' ')
% disp('Run remotely in parallel')
disp('Run on EC2 in parallel with 16 workes')
```

Name	Value
files	<1x16 cell>
nRuns	512
pkAccel	<1x512 double>
totAccel	<1x512 double>
x	[1.3333e+04 1.96

```
>> optimSuspensionScript


---------------------------------------------
Starting matlabpool using the 'FGonEC2' profile ... connected to 16 labs.

Run on EC2 in parallel with 16 workes
Elapsed time is 215.115770 seconds.
Sending a stop signal to all the labs ... stopped.

---------------------------------------------
Starting matlabpool using the 'local' profile ... connected to 4 labs.

Run locally in parallel with 4 workers
Elapsed time is 969.788383 seconds.
Sending a stop signal to all the labs ... stopped.

---------------------------------------------

Run locally in serial
Elapsed time is 2335.590516 seconds.
fx >>
```

The Mechanics of `parfor` Loops



```
a = zeros(20, 1)
parfor i = 1:20
  a(i) = i;
end
a
```

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. `eval`, `load`, `global`, etc.)
  - Cannot contain `break` or `return` statements
  - Cannot contain another `parfor` loop

`parfor` Variable Classification

- All variables referenced at the top level of the **`parfor`** must be resolved and classified

Classification	Description
Loop	Serves as a loop index for arrays
Sliced	An array whose segments are operated on by different iterations of the loop
Broadcast	A variable defined before the loop whose value is used inside the loop, but never assigned inside the loop
Reduction	Accumulates a value across iterations of the loop, regardless of iteration order
Temporary	Variable created inside the loop, but unlike sliced or reduction variables, not available outside the loop

Considerations When Using `parfor`

Advantages

- **`parfor`** often involves just minimal code changes
- parallel execution of independent iterations of a for-loop
- working interactively on local or remote cluster

Limitations

- **`parfor`** automatically quits on error
- **`parfor`** doesn’t provide intermediate results

Advice for Converting `for` to `parfor`

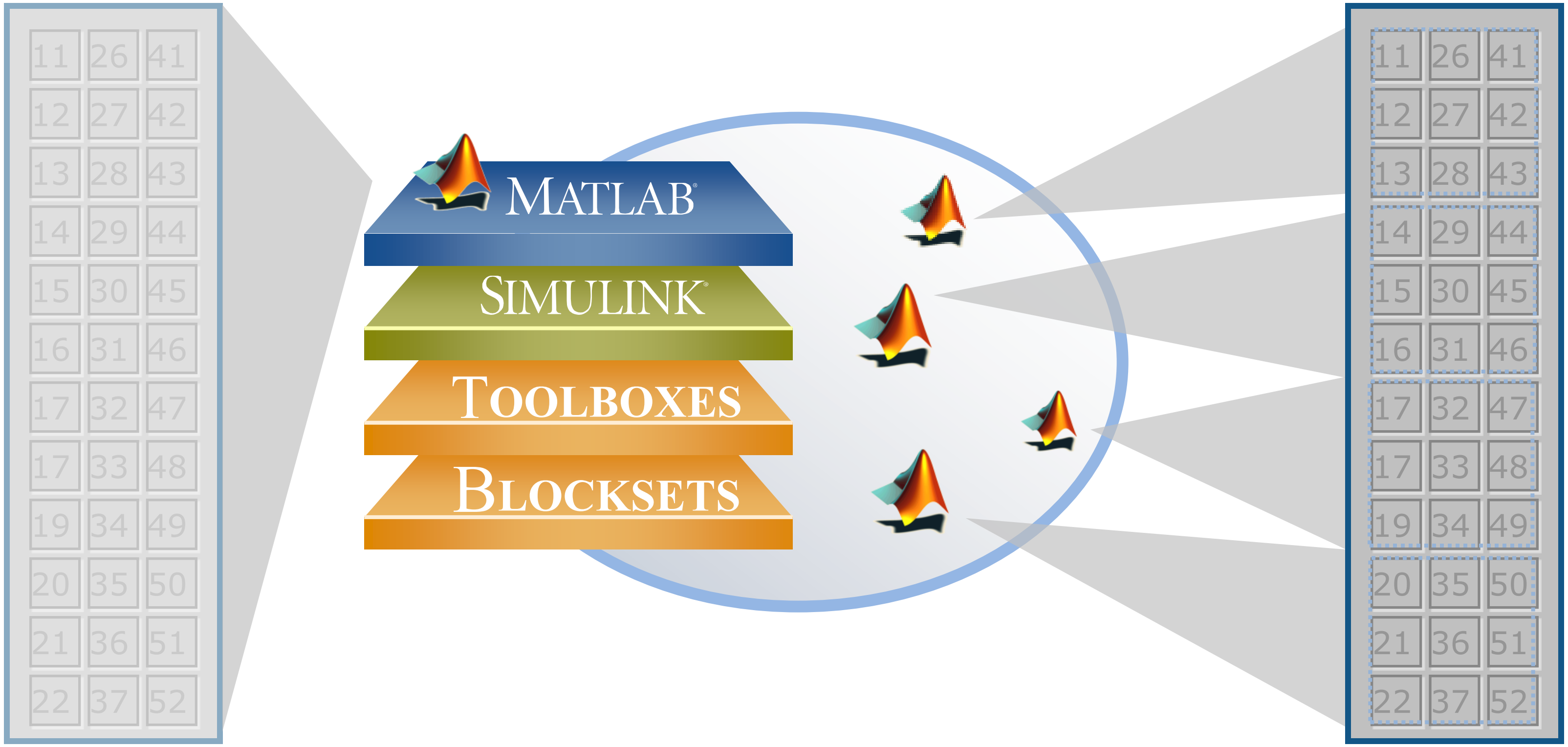
- Use the Code-Analyzer to diagnose `parfor` issues

- 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

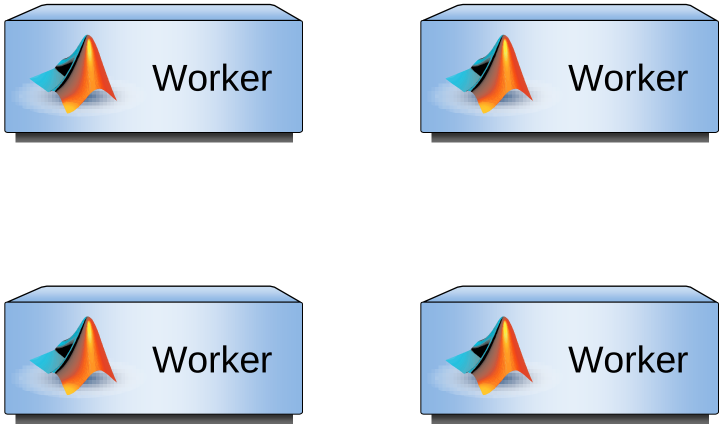
- http://blogs.mathworks.com/loren/2009/10/02/using-parfor-loops-getting-up-and-running/

Large Datasets (Data Parallel)

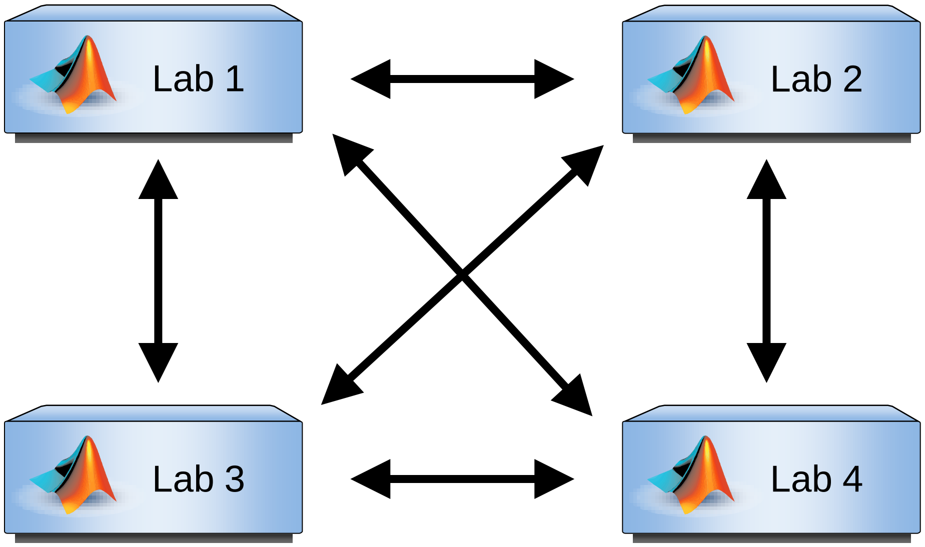


Parallel Terminology

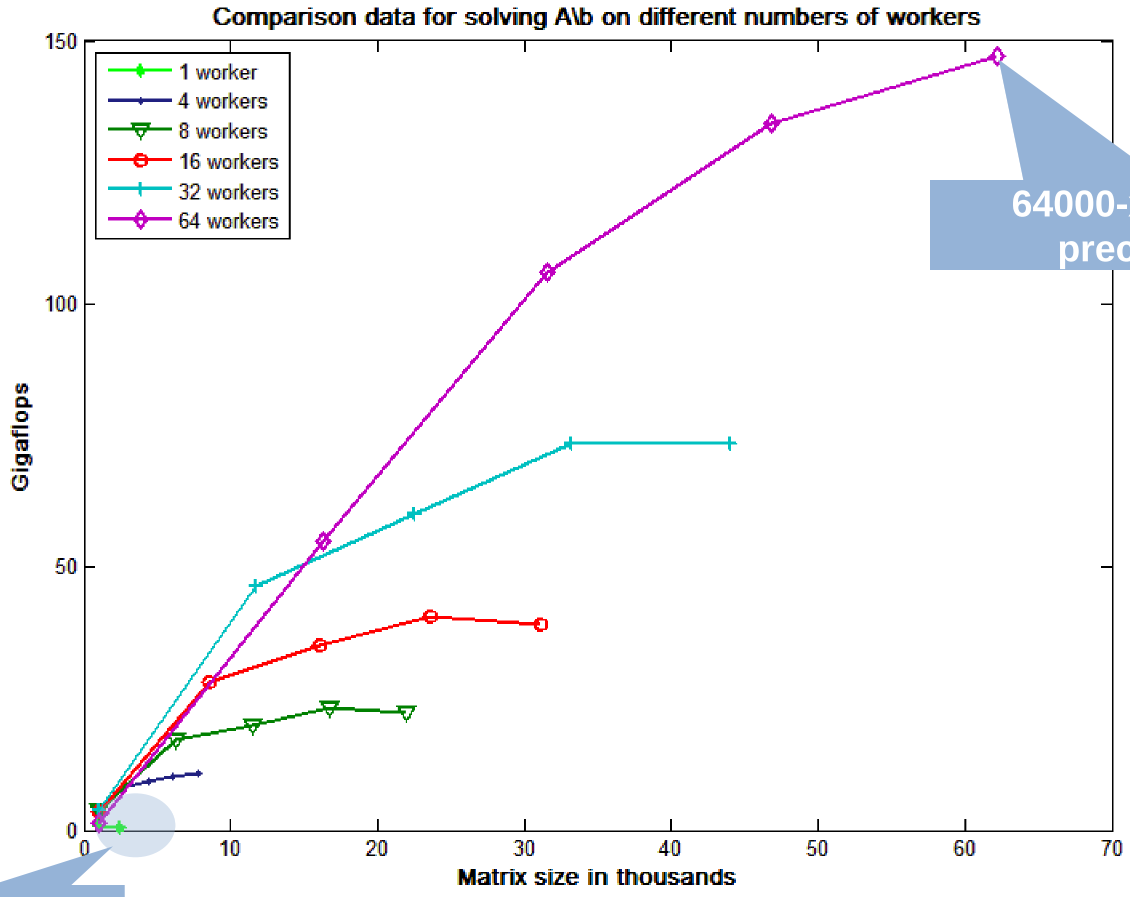
Task-Parallel
(Independent)



Data-Parallel
(Communication)



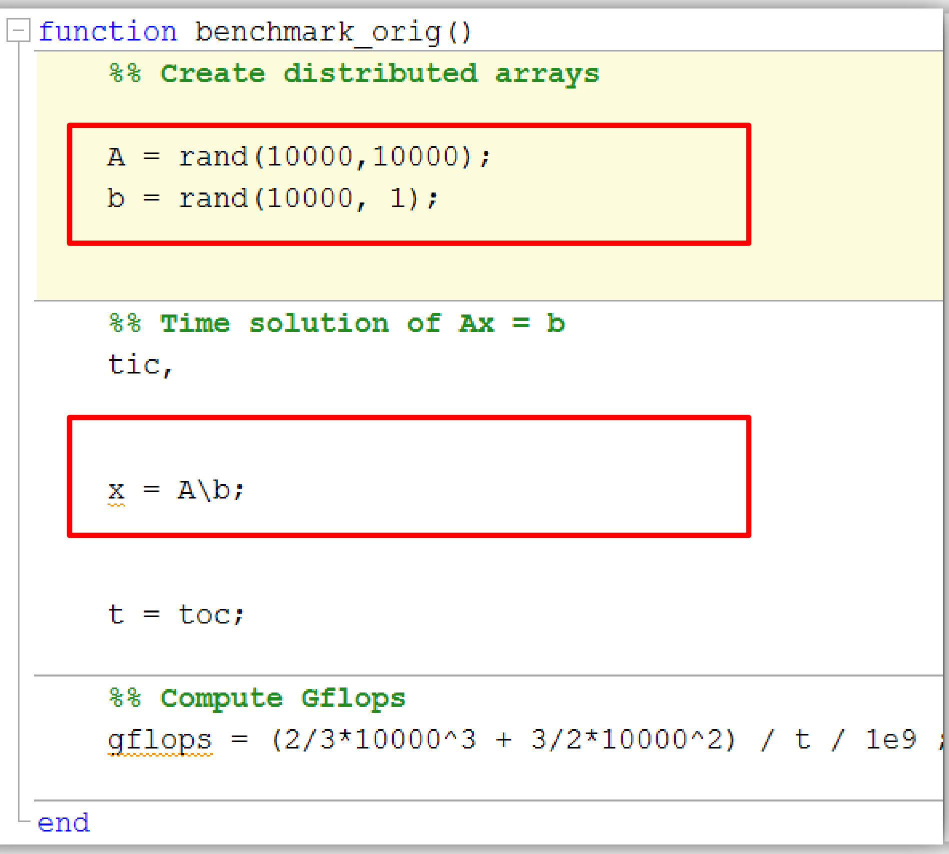
Benchmark: Solving A\b



64000-x-64000 double-precision > 30GB

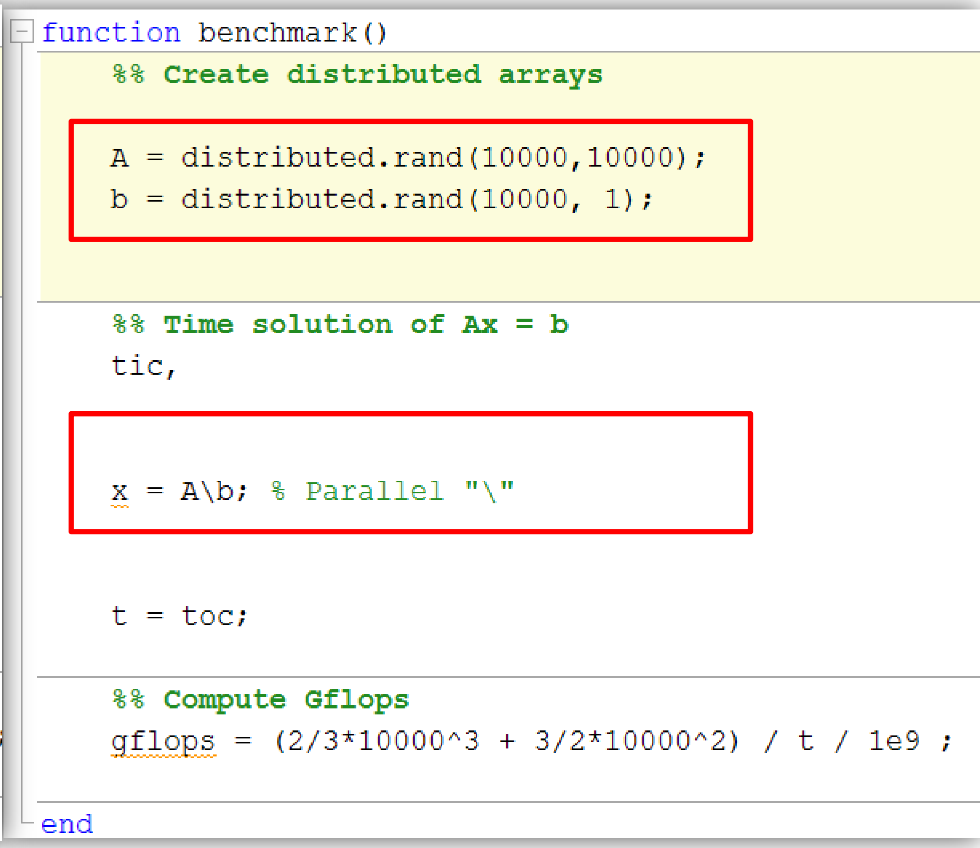
Range of a typical desktop computer

Regular MATLAB code



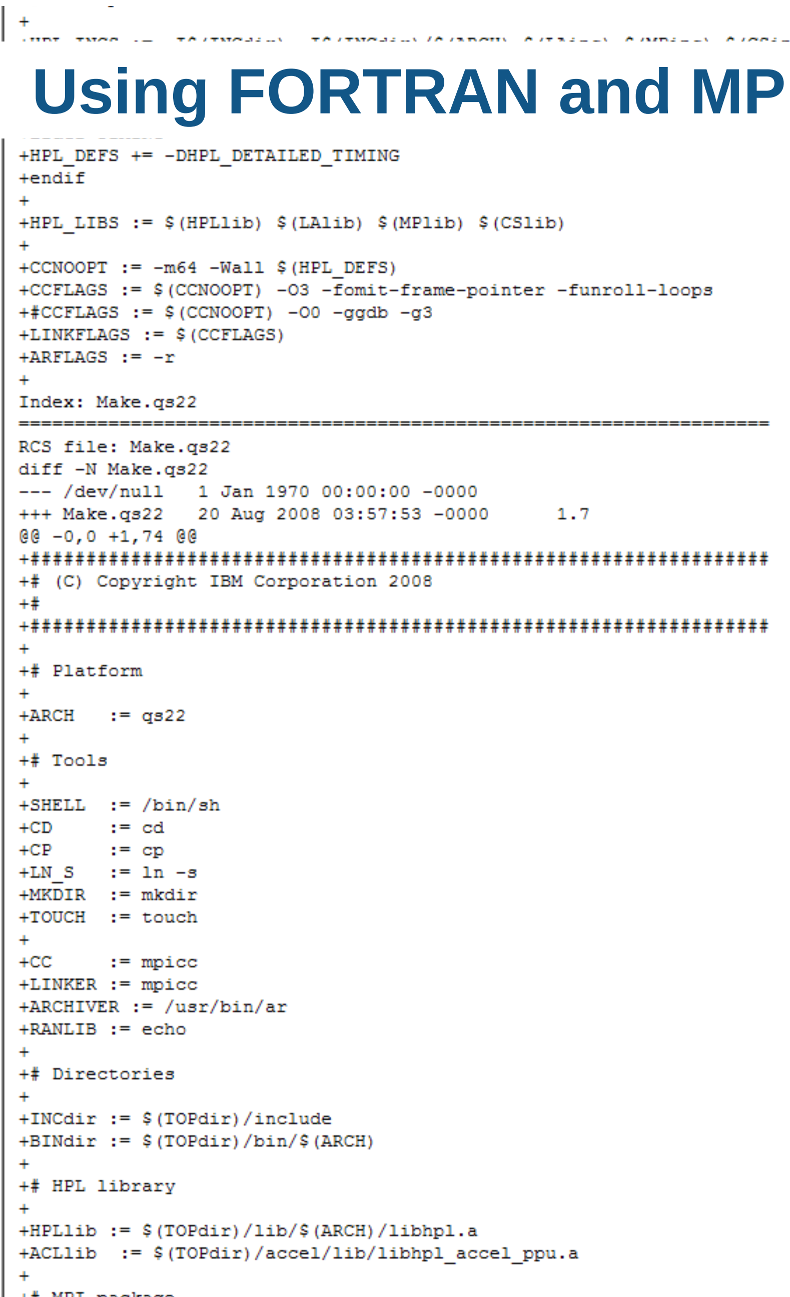
```
function benchmark_orig()
    %% Create distributed arrays

    A = rand(10000,10000);
    b = rand(10000, 1);


    %% Time solution of Ax = b
    tic,


    x = A\b;


    t = toc;

    %% Compute Gflops
    gflops = (2/3*10000^3 + 3/2*10000^2) / t / 1e9 ;

end
```

Using Distributed Arrays



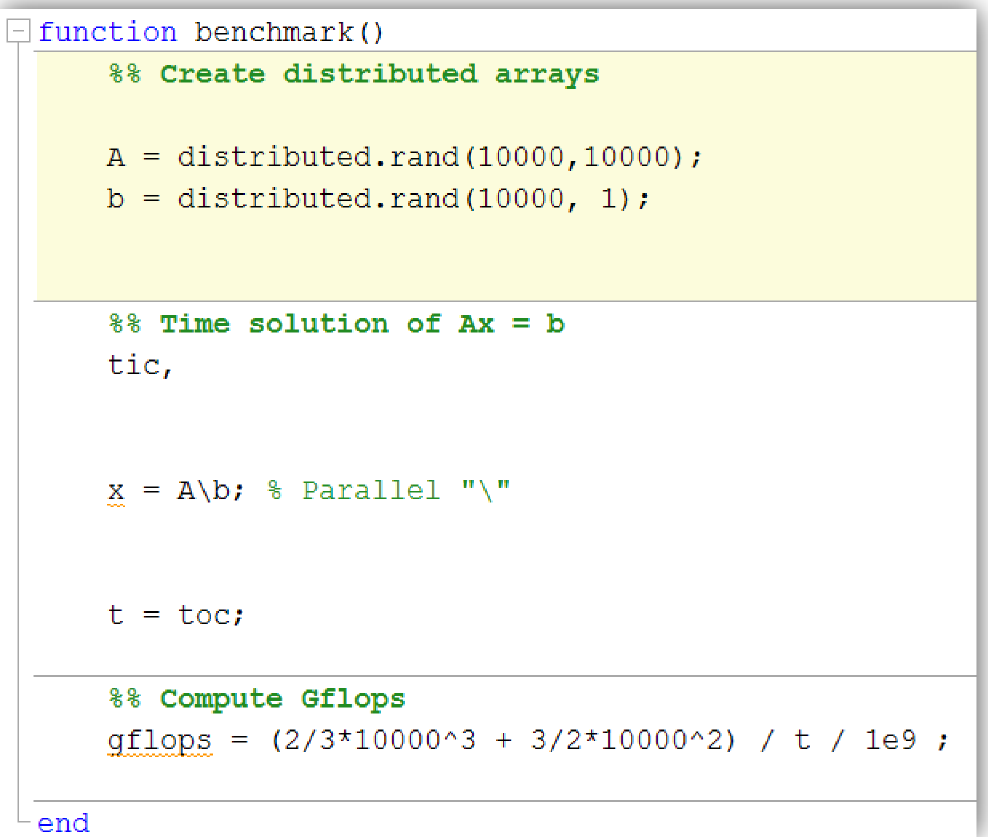
```
function benchmark()
    %% Create distributed arrays

    A = distributed.rand(10000,10000);
    b = distributed.rand(10000, 1);


    %% Time solution of Ax = b
    tic,


    x = A\b; % Parallel "\"


    t = toc;

    %% Compute Gflops
    gflops = (2/3*10000^3 + 3/2*10000^2) / t / 1e9 ;

end
```

Using FORTRAN and MPI



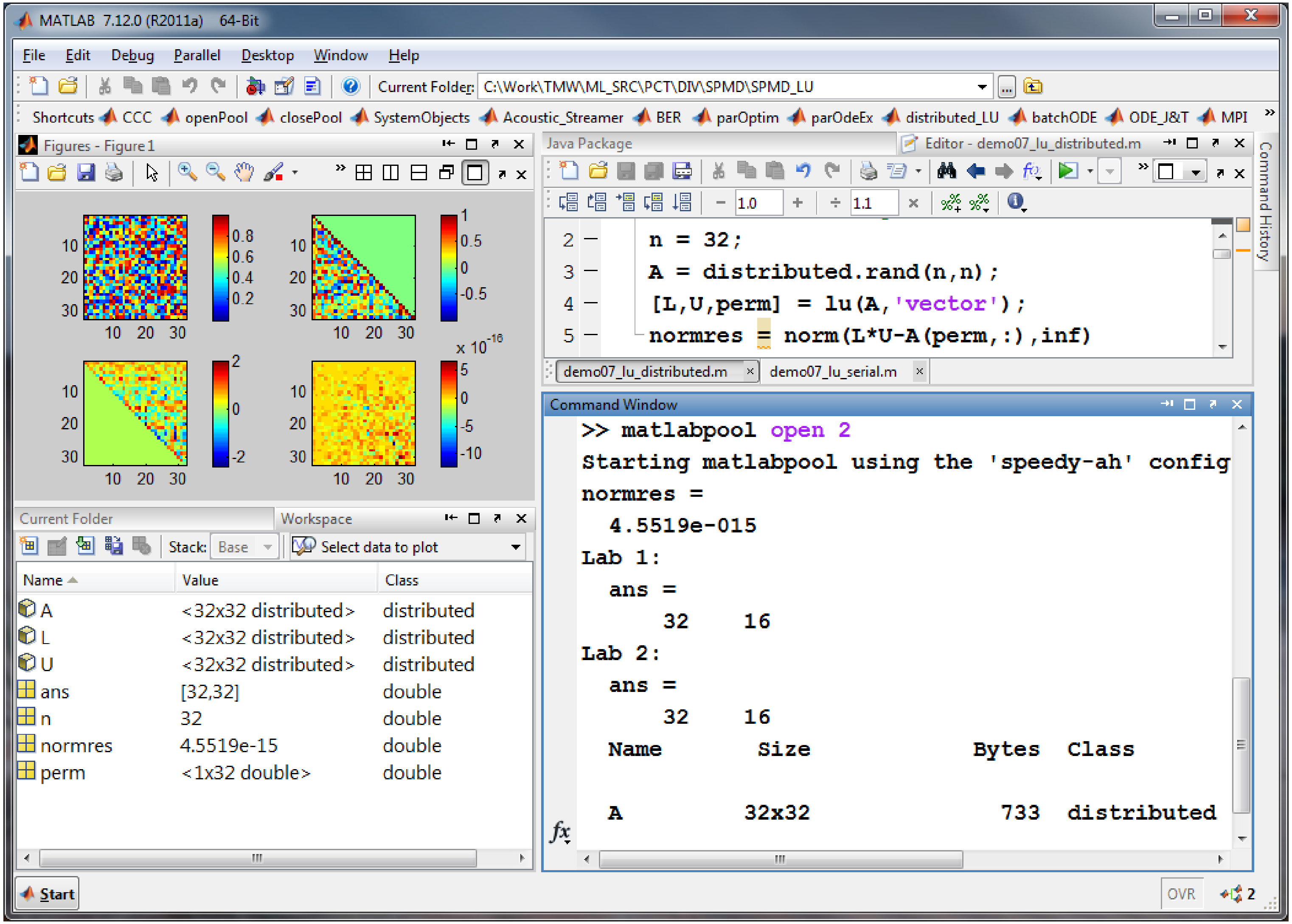
```
+
+HPL_INCS := -I$(INCdir) -I$(INCdir)/$(ARCH) $(LAinc) $(MPinc) $(CSinc)
+HPL_DEFS += -DHPL_DETAILED_TIMING
+endif
+
+HPL_LIBS := $(HPLlib) $(LAlib) $(MPlib) $(CSlib)
+
+CCNOOPT := -m64 -Wall $(HPL_DEFS)
+CCFLAGS := $(CCNOOPT) -O3 -fomit-frame-pointer -funroll-loops
+#CCFLAGS := $(CCNOOPT) -O0 -ggdb -g3
+LINKFLAGS := $(CCFLAGS)
+ARFLAGS := -r
+
Index: Make.qs22
===================================================================
RCS file: Make.qs22
diff -N Make.qs22
--- /dev/null   1 Jan 1970 00:00:00 -0000
+++ Make.qs22   20 Aug 2008 03:57:53 -0000      1.7
@@ -0,0 +1,74 @@
+##################################################################
+# (C) Copyright IBM Corporation 2008
+#
+##################################################################
+
+# Platform
+
+ARCH    := qs22
+
+# Tools
+
+SHELL   := /bin/sh
+CD      := cd
+CP      := cp
+LN_S    := ln -s
+MKDIR   := mkdir
+TOUCH   := touch
+
+CC      := mpicc
+LINKER  := mpicc
+ARCHIVER := /usr/bin/ar
+RANLIB  := echo
+
+# Directories
+
+INCdir := $(TOPdir)/include
+BINdir := $(TOPdir)/bin/$(ARCH)
+
+# HPL library
+
+HPLlib := $(TOPdir)/lib/$(ARCH)/libhpl.a
+ACLlib  := $(TOPdir)/accel/lib/libhpl_accel_ppu.a
+
+# MPI package
```

Using Distributed Arrays



```matlab
function benchmark()
    %% Create distributed arrays

    A = distributed.rand(10000,10000);
    b = distributed.rand(10000, 1);


    %% Time solution of Ax = b
    tic,


    x = A\b; % Parallel "\"


    t = toc;

    %% Compute Gflops
    gflops = (2/3*10000^3 + 3/2*10000^2) / t / 1e9 ;

end
```

Example: LU Factorization with Distributed Arrays



Current Folder: C:\Work\TMW\ML_SRC\PCT\DIV\SPMD\SPMD_LU

```
n = 32;
A = distributed.rand(n,n);
[L,U,perm] = lu(A,'vector');
normres = norm(L*U-A(perm,:),inf)
```

```
>> matlabpool open 2
Starting matlabpool using the 'speedy-ah' config
normres =
  4.5519e-015
Lab 1:
  ans =
      32    16
Lab 2:
  ans =
      32    16
  Name        Size              Bytes  Class

  A           32x32               733  distributed
```

Name	Value	Class
A	<32x32 distributed>	distributed
L	<32x32 distributed>	distributed
U	<32x32 distributed>	distributed
ans	[32,32]	double
n	32	double
normres	4.5519e-15	double
perm	<1x32 double>	double

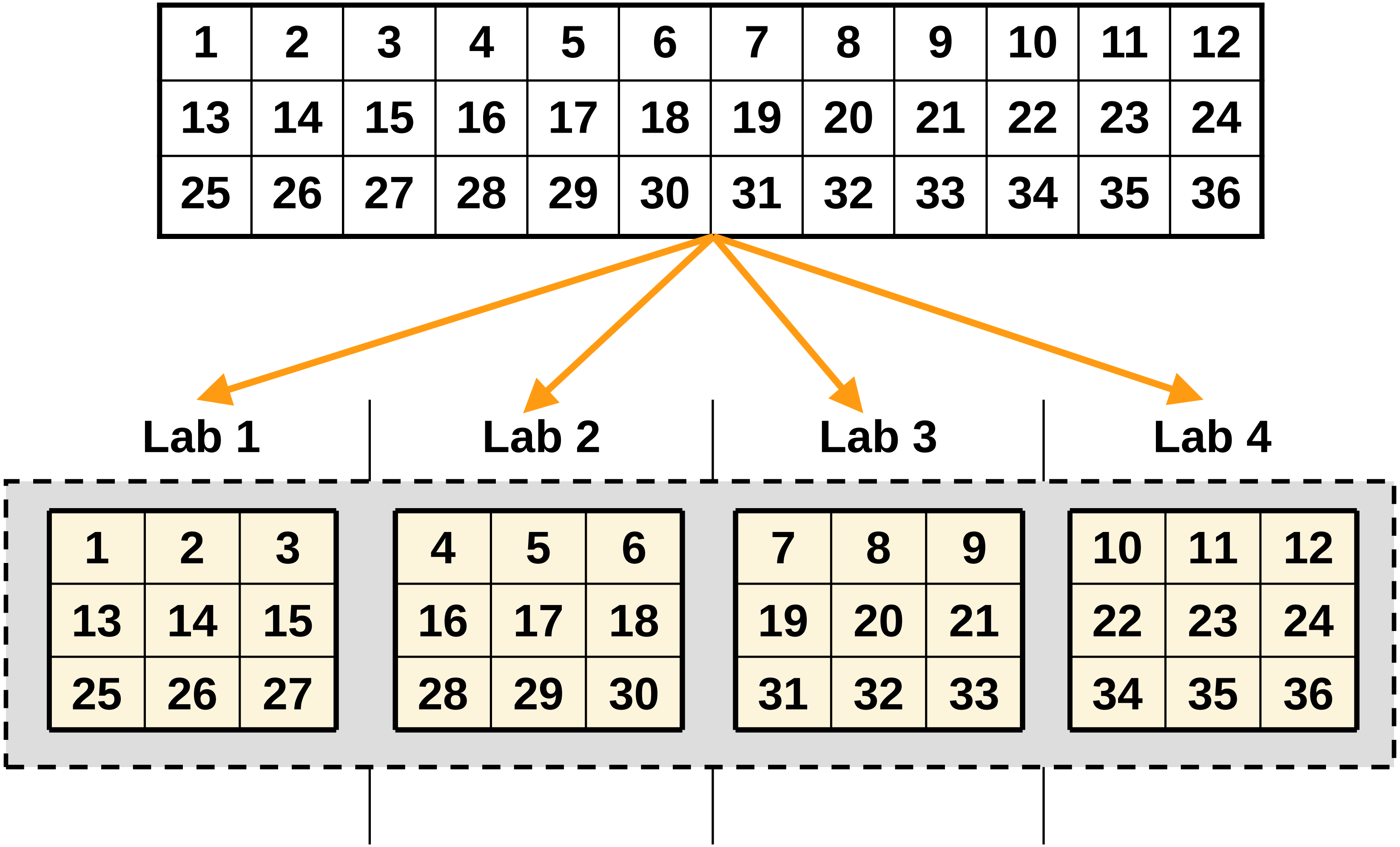
Client-side Distributed Arrays and SPMD

- Client-side distributed arrays
  - Class `distributed`
  - Can be created and manipulated directly from the client.
  - Simpler access to memory on labs
  - Client-side visualization capabilities
- `spmd`
  - Block of code executed on workers
  - Worker specific commands
  - Explicit communication between workers
  - Mixture of parallel and serial code

`spmd`

- **S**ingle **P**rogram, **M**ultiple **D**ata
- Unlike variables used in multiple **`parfor`** loops, distributed arrays used in multiple **`spmd`** blocks retain state
- Use Code Analyzer to diagnose **`spmd`** issues

Distributed Arrays



1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36

Lab 1

1	2	3
13	14	15
25	26	27

Lab 2

4	5	6
16	17	18
28	29	30

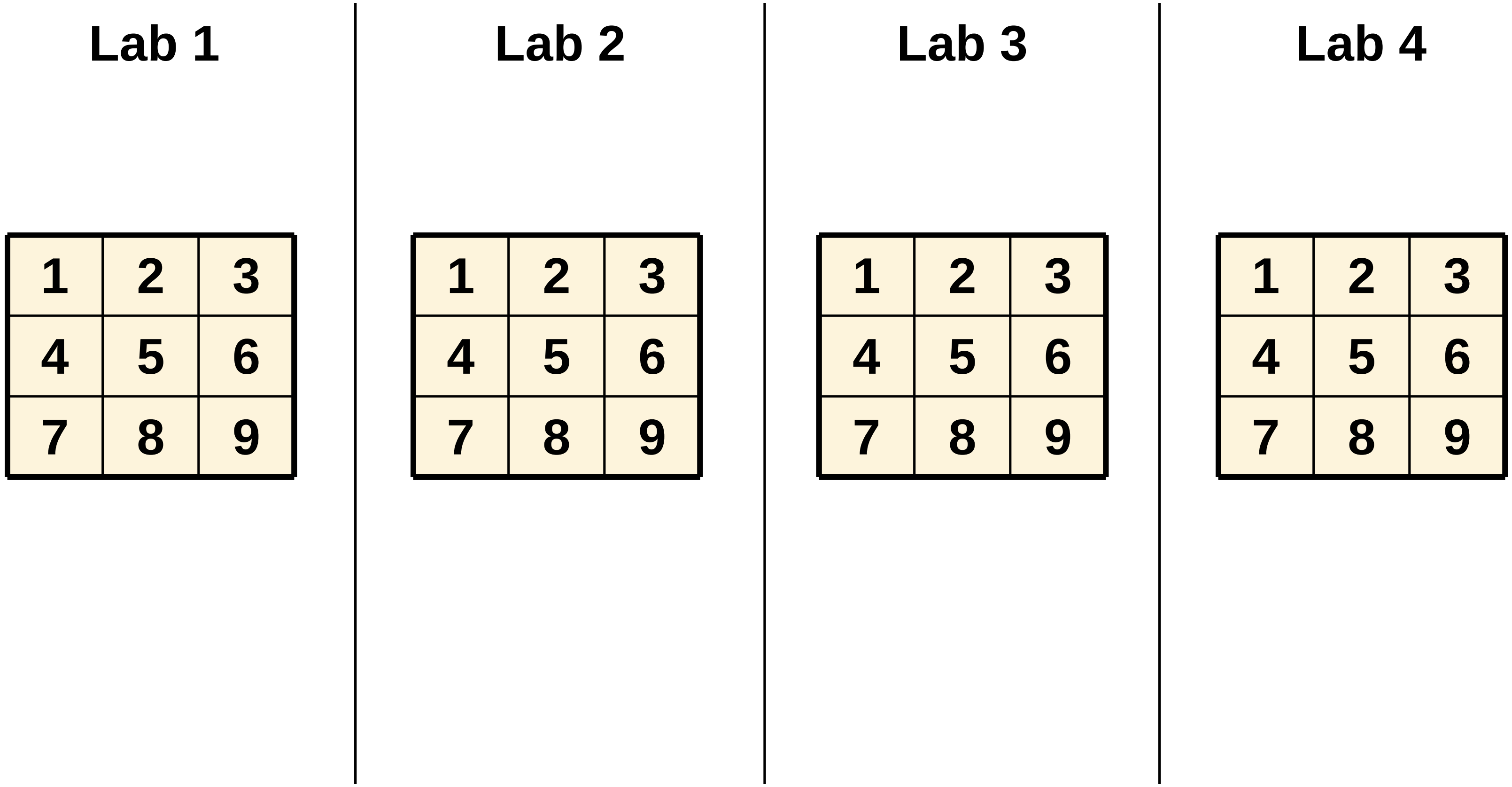
Lab 3

7	8	9
19	20	21
31	32	33

Lab 4

10	11	12
22	23	24
34	35	36

Replicated Arrays



Lab 1

1	2	3
4	5	6
7	8	9

Lab 2

1	2	3
4	5	6
7	8	9

Lab 3

1	2	3
4	5	6
7	8	9

Lab 4

1	2	3
4	5	6
7	8	9

Variant Arrays

Lab 1

1	2	3
4	5	6
7	8	9

Lab 2

2	4	6
8	10	12
14	16	18

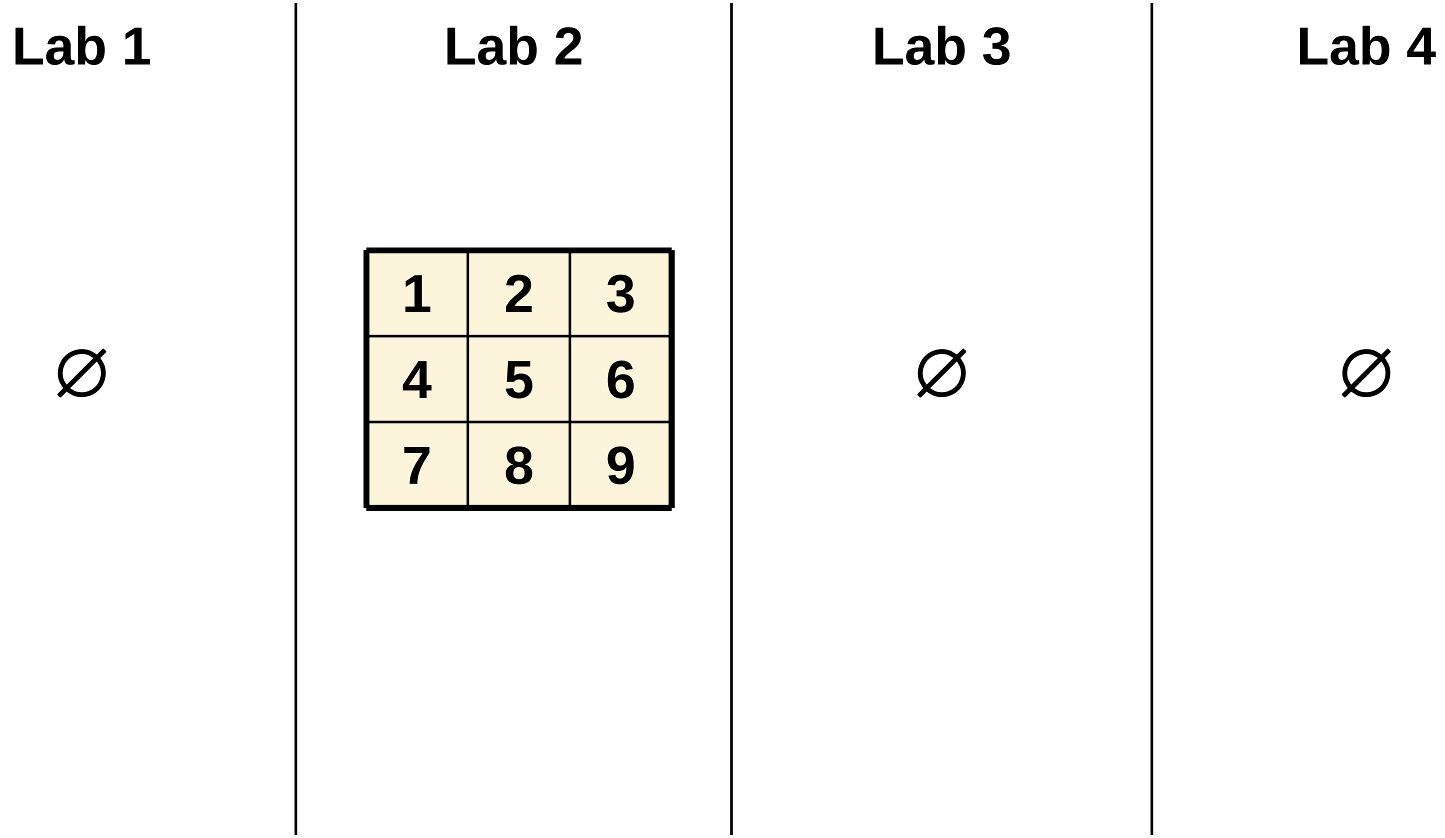
Lab 3

3	6	9
12	15	18
21	24	27

Lab 4

4	8	12
16	20	24
28	32	36

Private Arrays



Lab 1

∅

Lab 2

1	2	3
4	5	6
7	8	9

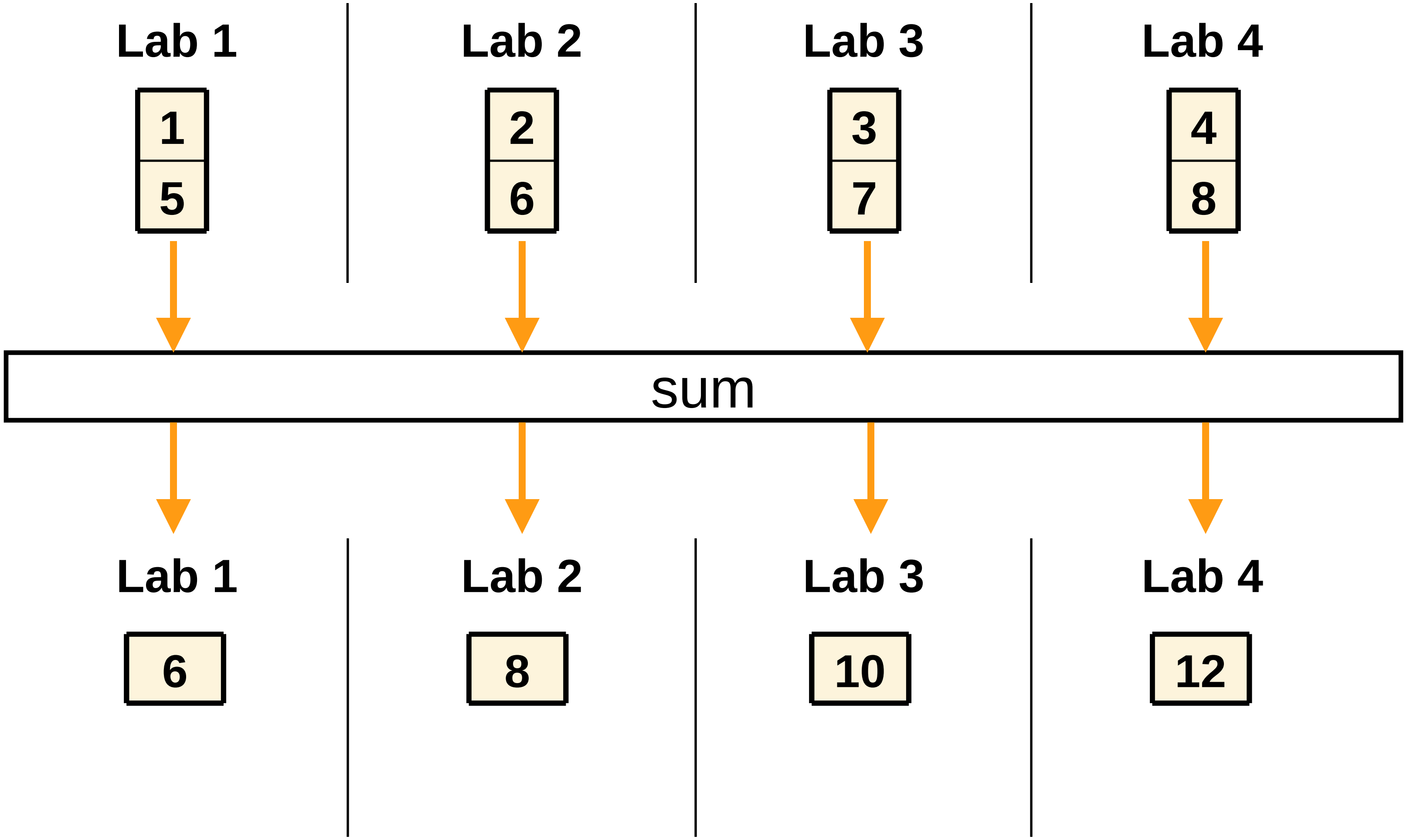
Lab 3

∅

Lab 4

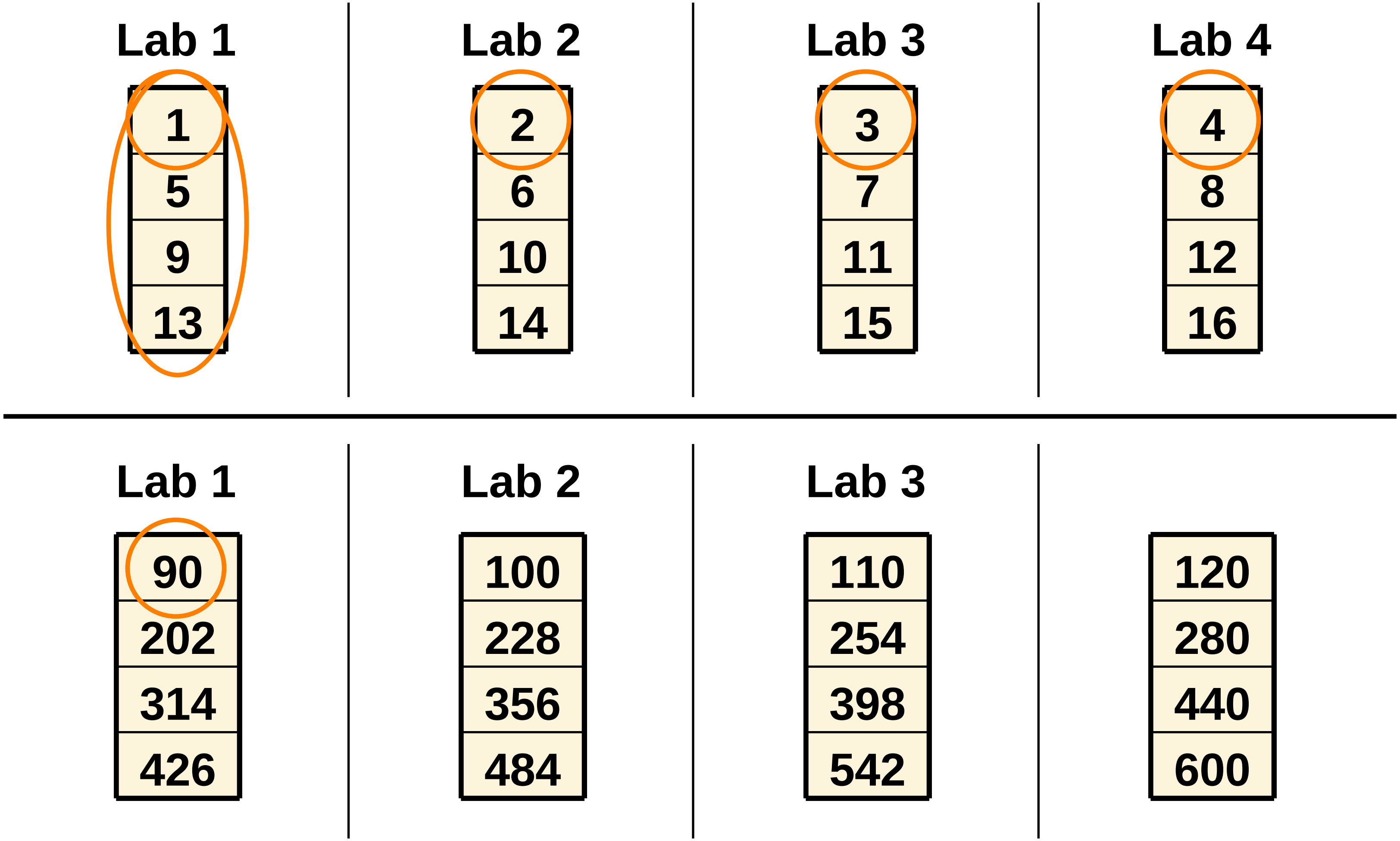
∅

Parallel Functions



Operations with Communication

```
>> spmd, D * D, end
```

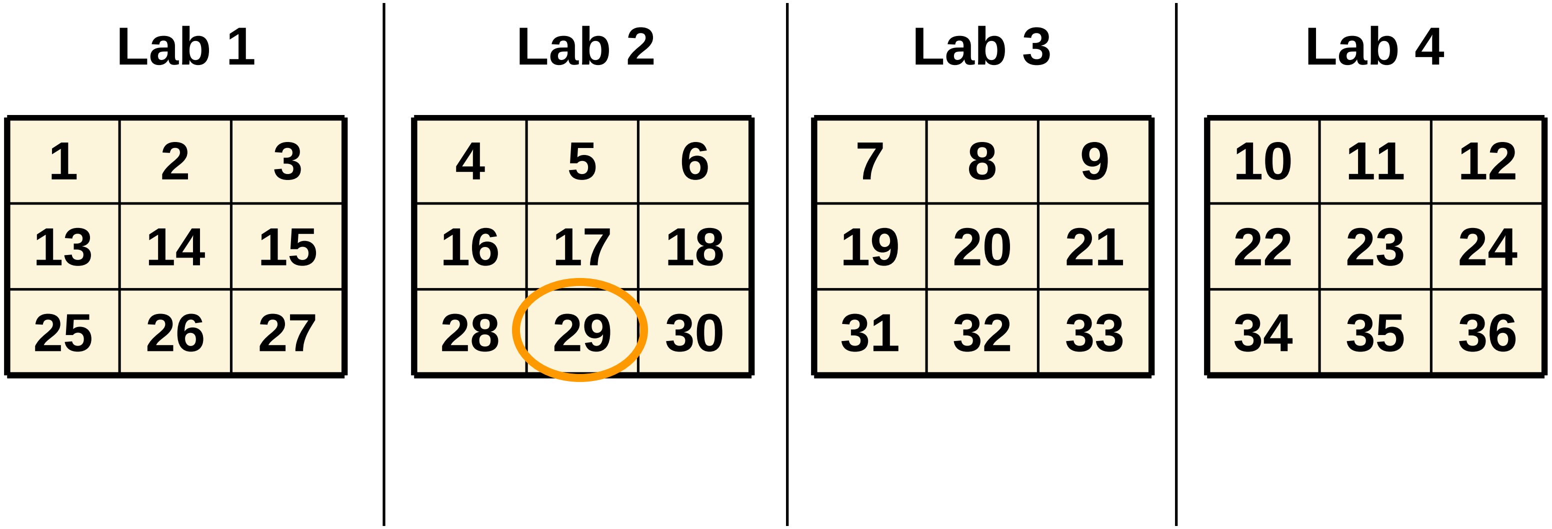


Lab 1	Lab 2	Lab 3	Lab 4
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Lab 1	Lab 2	Lab 3	
90	100	110	120
202	228	254	280
314	356	398	440
426	484	542	600

Indexing

```
>> data = D(3, 5)
```



Lab 1

1	2	3
13	14	15
25	26	27

Lab 2

4	5	6
16	17	18
28	29	30

Lab 3

7	8	9
19	20	21
31	32	33

Lab 4

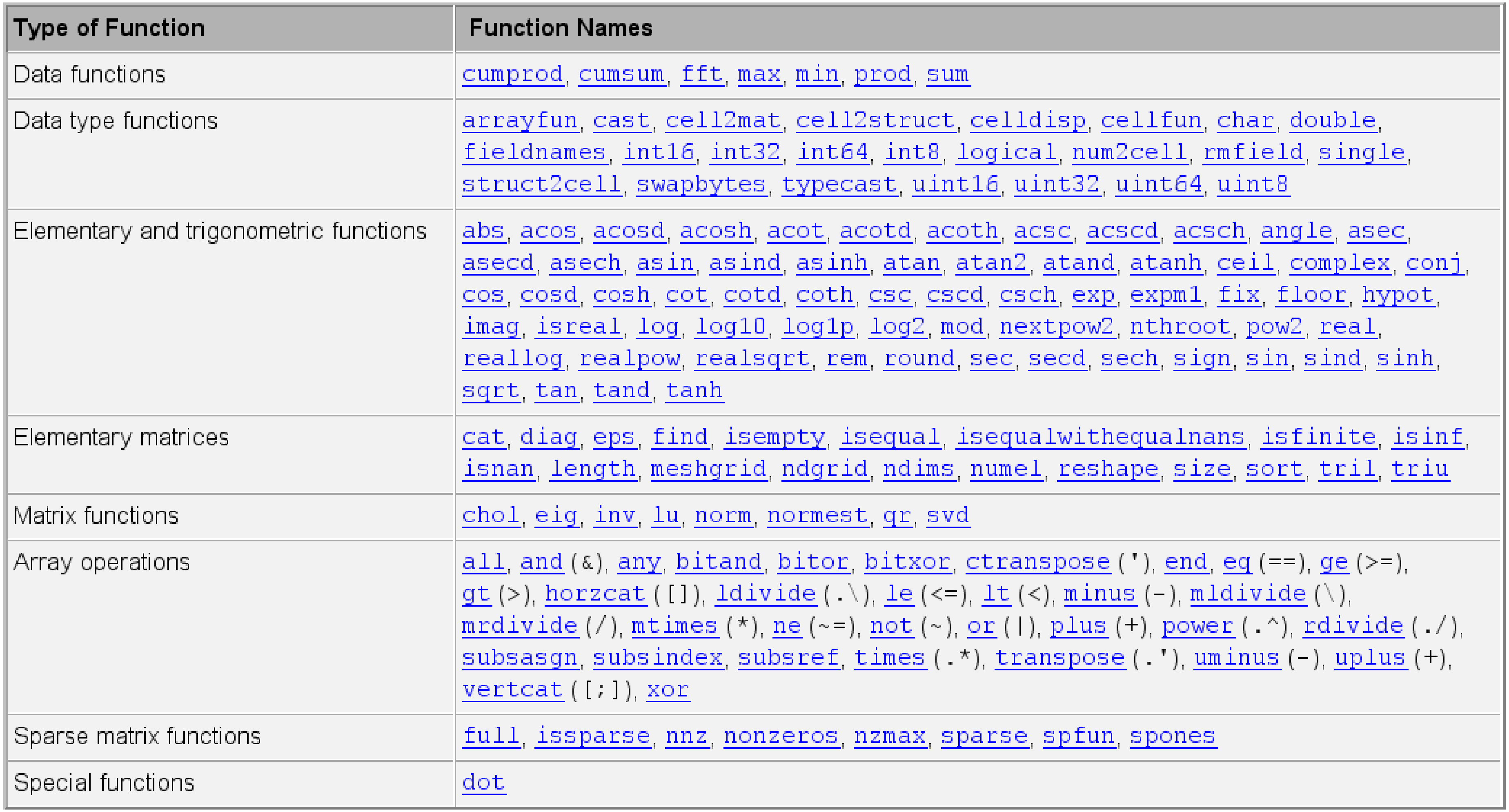
10	11	12
22	23	24
34	35	36

Distributed Arrays and Parallel Algorithms

- Distributed arrays
  - Store segments of data across participating workers
  - Create from any built-in class in MATLAB
    - Examples: doubles, sparse, logicals, cell arrays, and arrays of structs

- Parallel algorithms for codistributed arrays
  - Matrix manipulation operations
    - Examples: indexing, data type conversion, and transpose
  - Parallel linear algebra functions, such as **`svd`** and **`lu`**
  - Data distribution
    - Automatic, specify your own, or change at any time

Enhanced MATLAB Functions That Operate on Codistributed Arrays



Type of Function	Function Names
Data functions	cumprod, cumsum, fft, max, min, prod, sum
Data type functions	arrayfun, cast, cell2mat, cell2struct, celldisp, cellfun, char, double, fieldnames, int16, int32, int64, int8, logical, num2cell, rmfield, single, struct2cell, swapbytes, typecast, uint16, uint32, uint64, uint8
Elementary and trigonometric functions	abs, acos, acosd, acosh, acot, acotd, acoth, acsc, acscd, acsch, angle, asec, asecd, asech, asin, asind, asinh, atan, atan2, atand, atanh, ceil, complex, conj, cos, cosd, cosh, cot, cotd, coth, csc, cscd, csch, exp, expm1, fix, floor, hypot, imag, isreal, log, log10, log1p, log2, mod, nextpow2, nthroot, pow2, real, reallog, realpow, realsqrt, rem, round, sec, secd, sech, sign, sin, sind, sinh, sqrt, tan, tand, tanh
Elementary matrices	cat, diag, eps, find, isempty, isequal, isequalwithequalnans, isfinite, isinf, isnan, length, meshgrid, ndgrid, ndims, numel, reshape, size, sort, tril, triu
Matrix functions	chol, eig, inv, lu, norm, normest, qr, svd
Array operations	all, and (&), any, bitand, bitor, bitxor, ctranspose ('), end, eq (==), ge (>=), gt (>), horzcat ([]), ldivide (.\), le (<=), lt (<), minus (-), mldivide (\), mrdivide (/), mtimes (*), ne (~=), not (~), or (|), plus (+), power (.^), rdivide (./), subsasgn, subsindex, subsref, times (.*), transpose (.'), uminus (-), uplus (+), vertcat ([;]), xor
Sparse matrix functions	full, issparse, nnz, nonzeros, nzmax, sparse, spfun, spones
Special functions	dot

MPI-Based Functions in Parallel Computing Toolbox

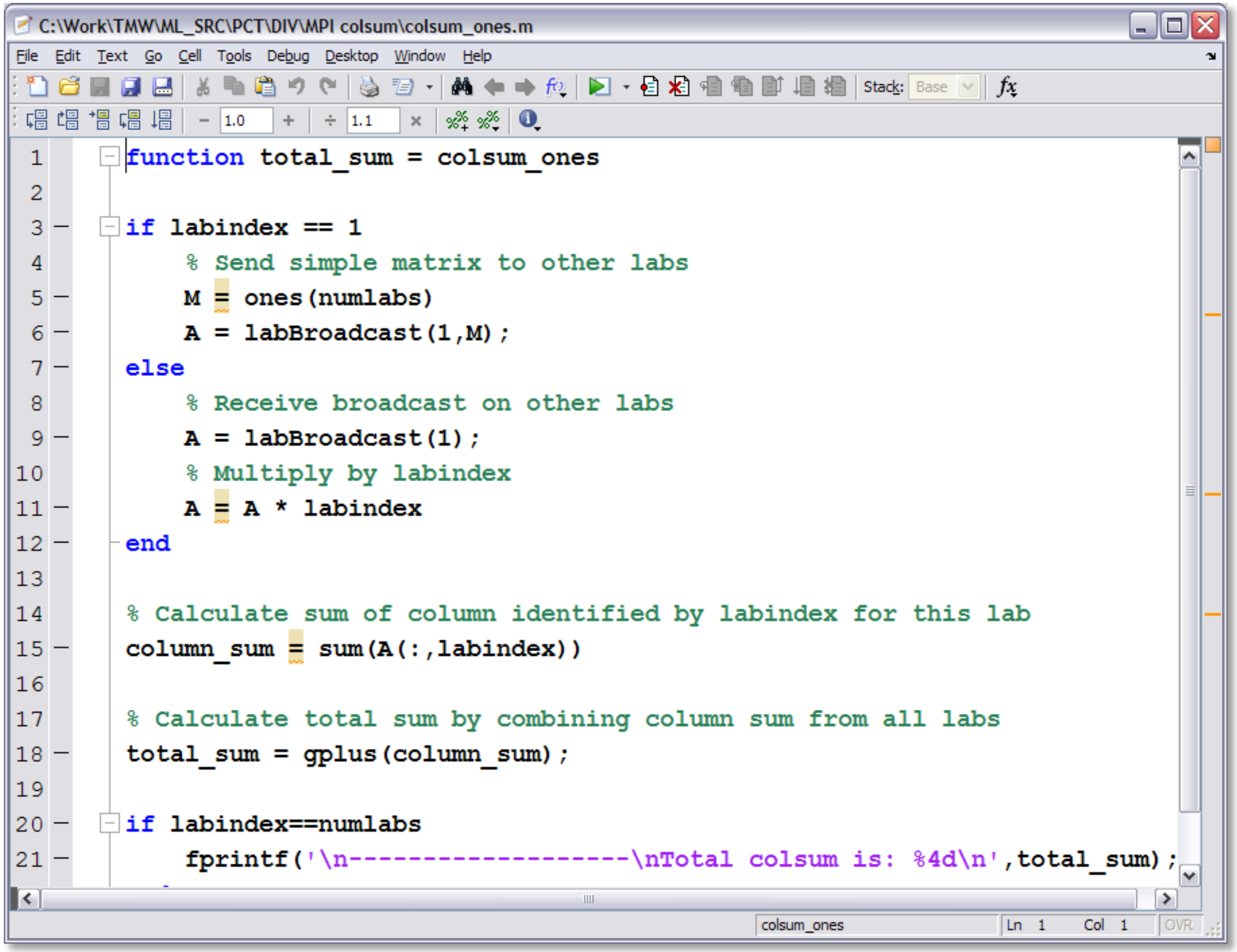
Use when a high degree of control over parallel algorithm is required

- High-level abstractions of MPI functions
  - `labSendReceive`, `labBroadcast`, and others
  - Send, receive, and broadcast any data type in MATLAB

- Automatic bookkeeping
  - Setup: communication, ranks, etc.
  - Error detection: deadlocks and miscommunications

- Pluggable
  - Use any MPI implementation that is *binary*-compatible with MPICH2

Example: MPI-based Functions


C:\Work\TMW\ML_SRC\PCT\DIV\MPI colsum\colsum_ones.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1
colsum_ones
Ln 1 Col 1
OVR

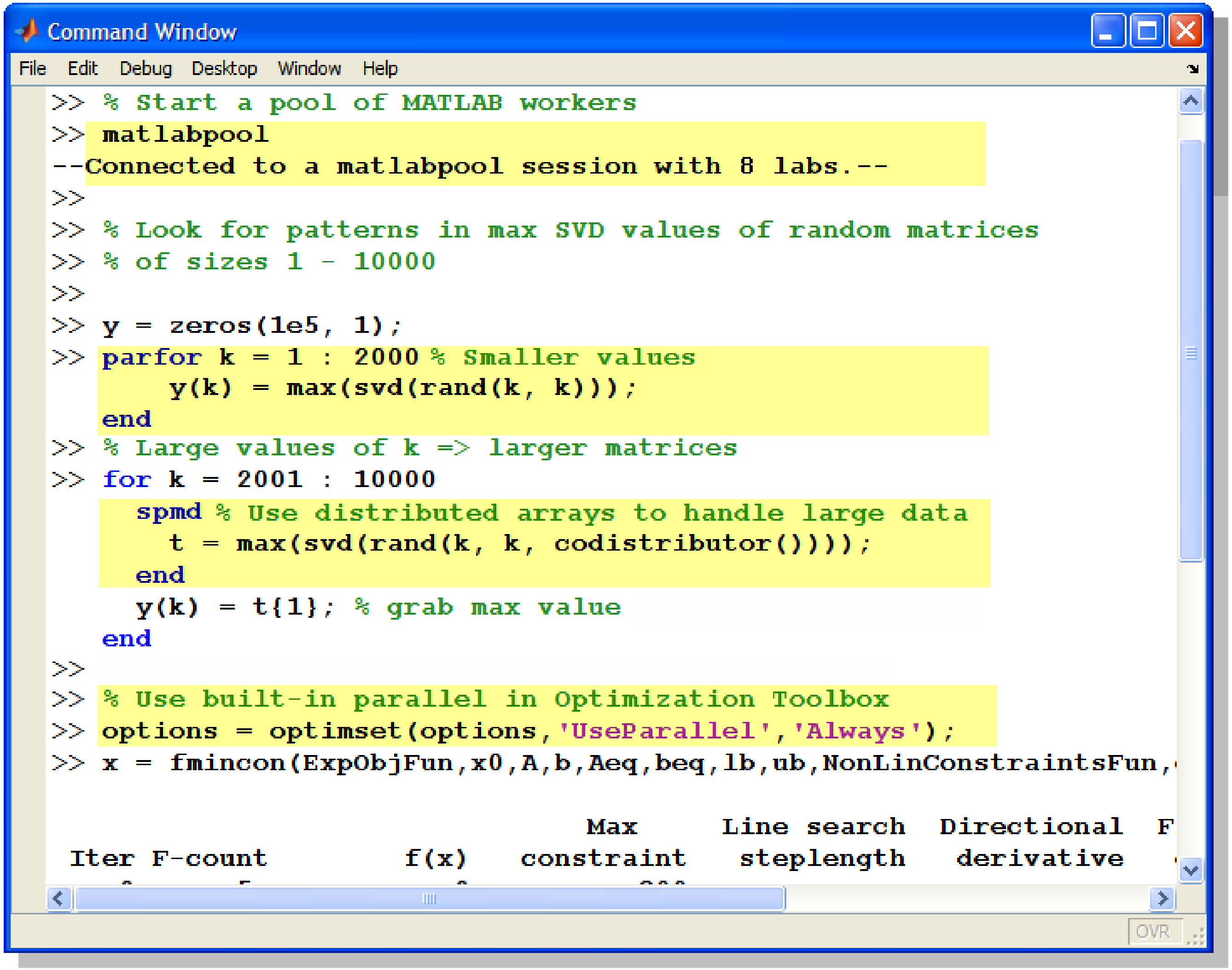
```
function total_sum = colsum_ones

if labindex == 1
    % Send simple matrix to other labs
    M = ones(numlabs)
    A = labBroadcast(1,M);
else
    % Receive broadcast on other labs
    A = labBroadcast(1);
    % Multiply by labindex
    A = A * labindex
end

% Calculate sum of column identified by labindex for this lab
column_sum = sum(A(:,labindex))

% Calculate total sum by combining column sum from all labs
total_sum = gplus(column_sum);

if labindex==numlabs
    fprintf('\n-------------------\nTotal colsum is: %4d\n',total_sum);
```

Parallel Profiler

- Profiles the execution time for a function
  - Similar to the MATLAB profiler
  - Includes information about the communication between labs
    - Time spent in communication
    - Amount of data passed between labs

- Benefits
  - Identify the bottlenecks in your parallel algorithm
  - Understand which operations require communication

One MATLABPOOL, Many Uses



```
>> % Start a pool of MATLAB workers
>> matlabpool
--Connected to a matlabpool session with 8 labs.--
>>
>> % Look for patterns in max SVD values of random matrices
>> % of sizes 1 - 10000
>>
>> y = zeros(1e5, 1);
>> parfor k = 1 : 2000 % Smaller values
       y(k) = max(svd(rand(k, k)));
   end
>> % Large values of k => larger matrices
>> for k = 2001 : 10000
     spmd % Use distributed arrays to handle large data
       t = max(svd(rand(k, k, codistributor())));
     end
     y(k) = t{1}; % grab max value
   end
>>
>> % Use built-in parallel in Optimization Toolbox
>> options = optimset(options,'UseParallel','Always');
>> x = fmincon(ExpObjFun,x0,A,b,Aeq,beq,lb,ub,NonLinConstraintsFun,

                                        Max     Line search  Directional  F
 Iter F-count          f(x)    constraint   steplength   derivative
```

Parallel Computing Tools Address…

Task-Parallel

Long computations

– Multiple independent iterations

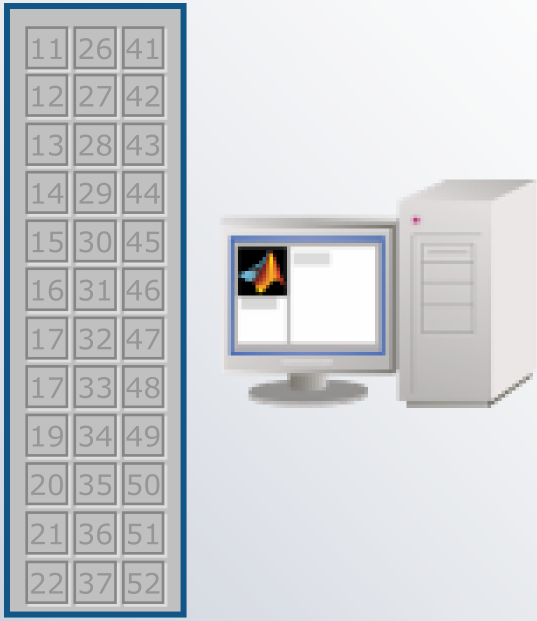
```
parfor i = 1 : n
    % do something with i
end
```

– Series of tasks

Task 1 | Task 2 | Task 3 | Task 4

Data-Parallel

Large data problems



11	26	41
12	27	42
13	28	43
14	29	44
15	30	45
16	31	46
17	32	47
17	33	48
19	34	49
20	35	50
21	36	51
22	37	52

Agenda

13:30	Welcome and Introduction
13:45	Introduction to Parallel Computing with MATLAB
	MATLAB–extensions with built-in support for Parallel Computing
14:15	Interactive development of task- and data-parallel Algorithms
15:15	*Coffee Break*
15:30	GPU programming with MATLAB
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	Q&A Session
17:00	*End of Seminar*

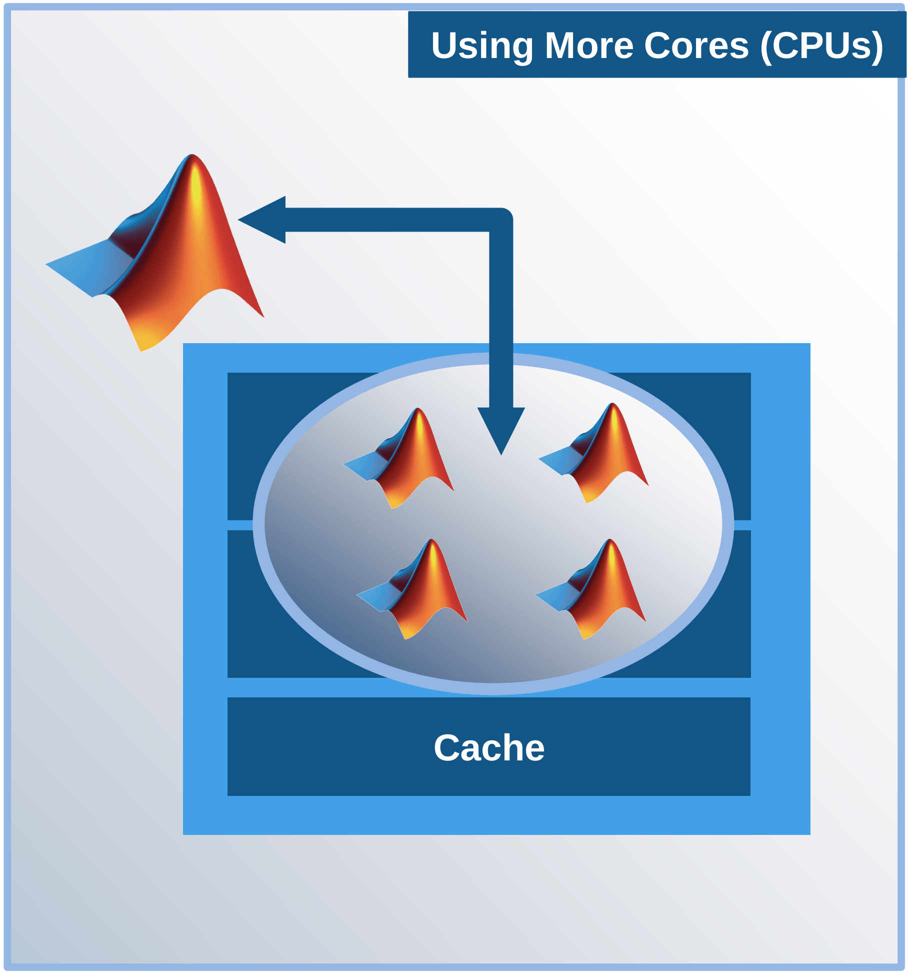
Agenda

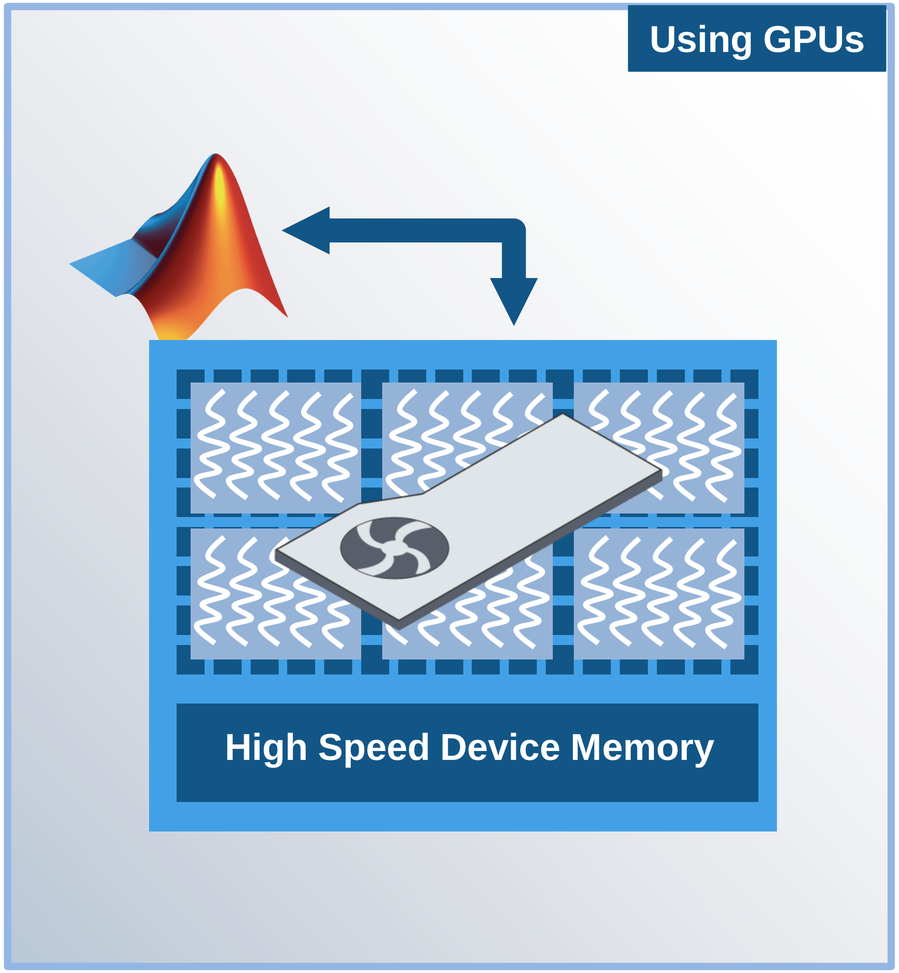
13:30	**Welcome and Introduction**
13:45	**Introduction to Parallel Computing with MATLAB**
	MATLAB–extensions with built-in support for Parallel Computing
14:15	**Interactive development of task- and data-parallel Algorithms**
15:15	***Coffee Break***
15:30	**GPU programming with MATLAB**
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	**Q&A Session**
17:00	***End of Seminar***

Graphics Processing Units (GPUs)

- 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



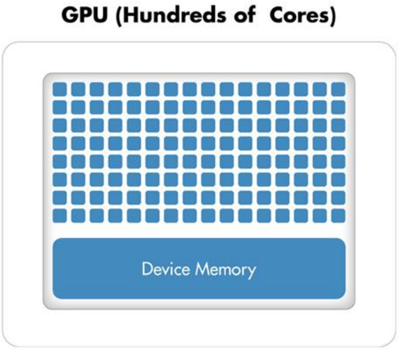


Common Terms Used in GPU Computing

- **CUDA®** : A parallel computing technology from NVIDIA®
  - Consists of a parallel computing architecture and developer tools, libraries, and programming directives for GPU computing
- **Device:** Card containing GPU and associated memory
- **Host:** CPU and system memory
- **Kernel:** Code written for execution on the GPU
  - Functions that can run on a large number of threads
  - Parallelism from each thread independently running the same program on different data

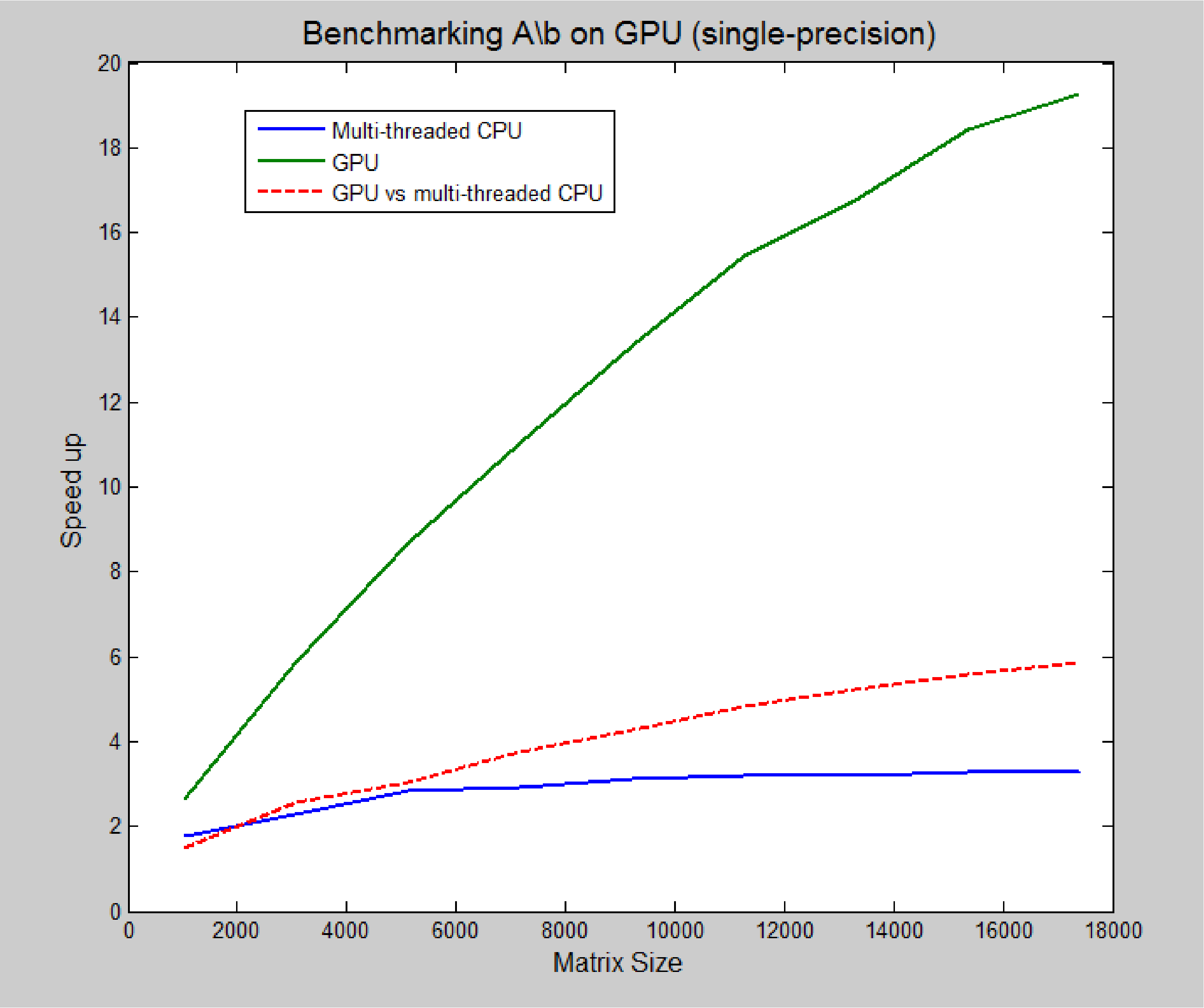
Criteria for Good Problems to Run on a GPU

- **Massively parallel:**
  - Able to break down calculations into hundreds or thousands of independent units of work
  - Motivation: Best performance when hundreds of GPU cores are kept busy



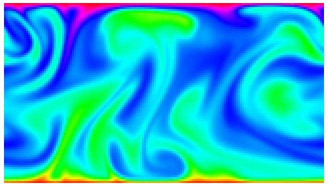
- **Computationally intensive:**
  - Computation time should significantly exceed time spent on data transfer to and from GPU
  - Motivation: Data transfer is costly since GPU is attached to CPU via the PCI Express bus

Benchmarking A\b on the GPU



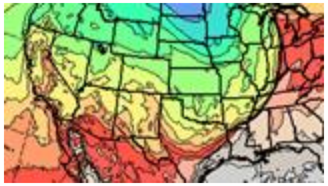
Problems for Running on the GPU

A selection of problems from the CUDA Community Showcase:

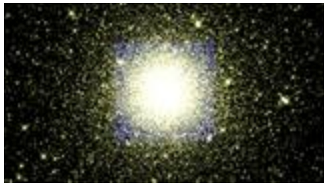


Computational Fluid Dynamics

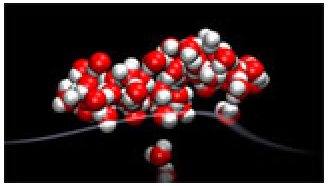
Computational Finance



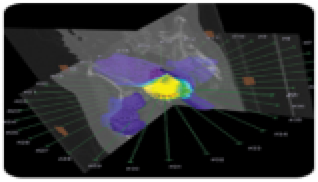
Weather Modeling



N-Body Simulations



Molecular Modeling



Digital Signal Processing

http://www.nvidia.com/object/cuda_showcase_html.html

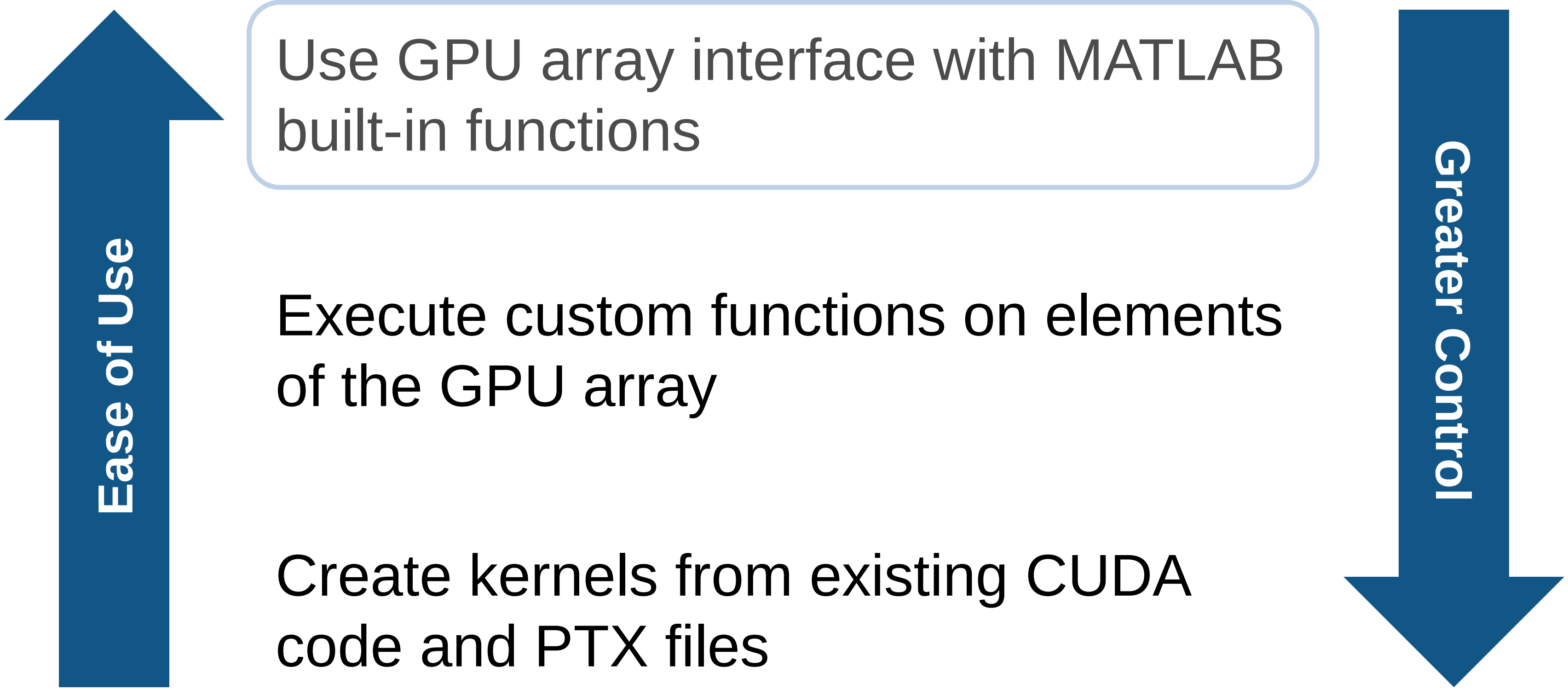
GPU Support with Parallel Computing Toolbox

- NVIDIA GPUs with compute capability 1.3 or greater
  - Includes Tesla 10-series and 20-series products (e.g., NVIDIA Tesla C2075 GPU: 448 processors, 6 GB memory)
  - http://www.nvidia.com/object/cuda_gpus.html

- Why we require compute capability 1.3
  - Support doubles (base data type in MATLAB)
  - Guarantee IEEE compliance
  - Provide cross-platform support

Evolving rapidly - Use latest release

Options for Targeting GPUs



Ease of Use

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

Greater Control

Overloaded MATLAB Functions

```
A = magic(1000);
G = gpuArray(A); %Push to GPU memory
b = parallel.gpu.GPUArray.rand(1000,1); %Create on GPU
F = fft(G);
x = G\b;
z = gather(x); %Bring back into MATLAB
```

Full list of built-in functions that support GPUArray

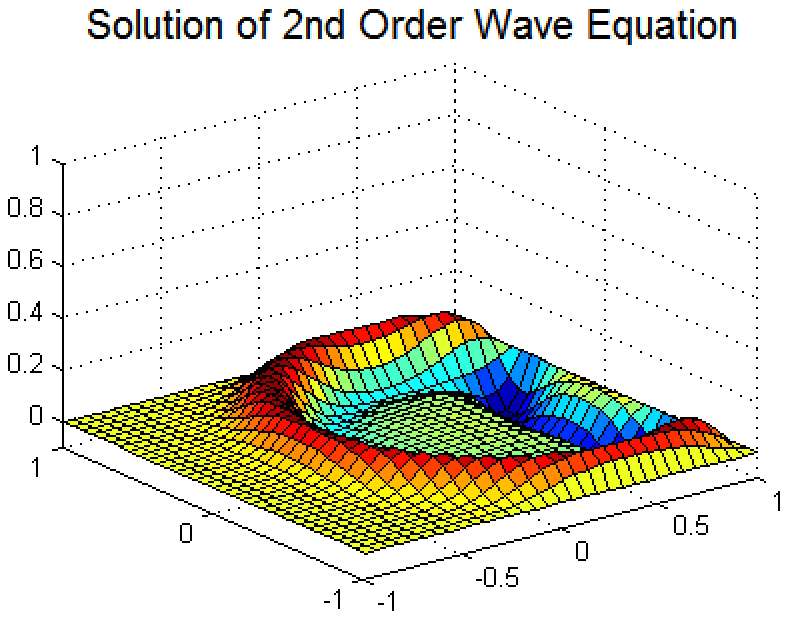
User’s Guide → GPU Computing → Using GPUArray

Example: Solving 2D Wave Equation

- Solve 2nd order wave equation using spectral methods:

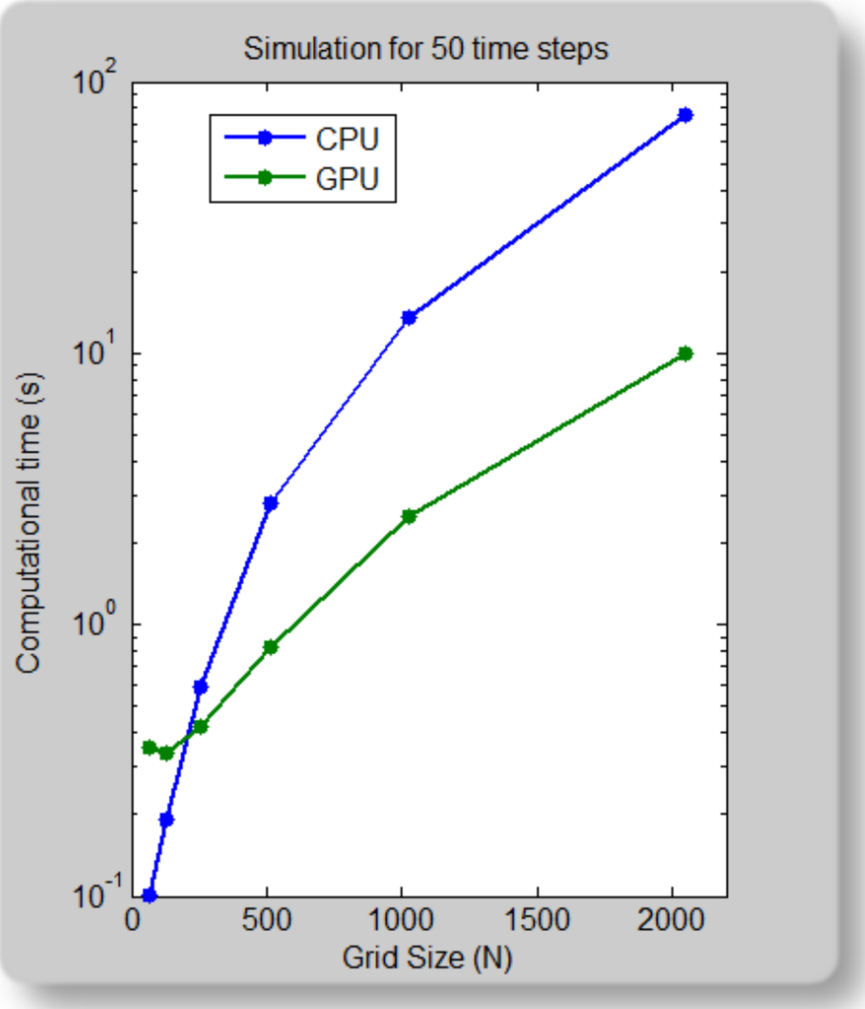
$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

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



Benchmark: Solving 2D Wave Equation

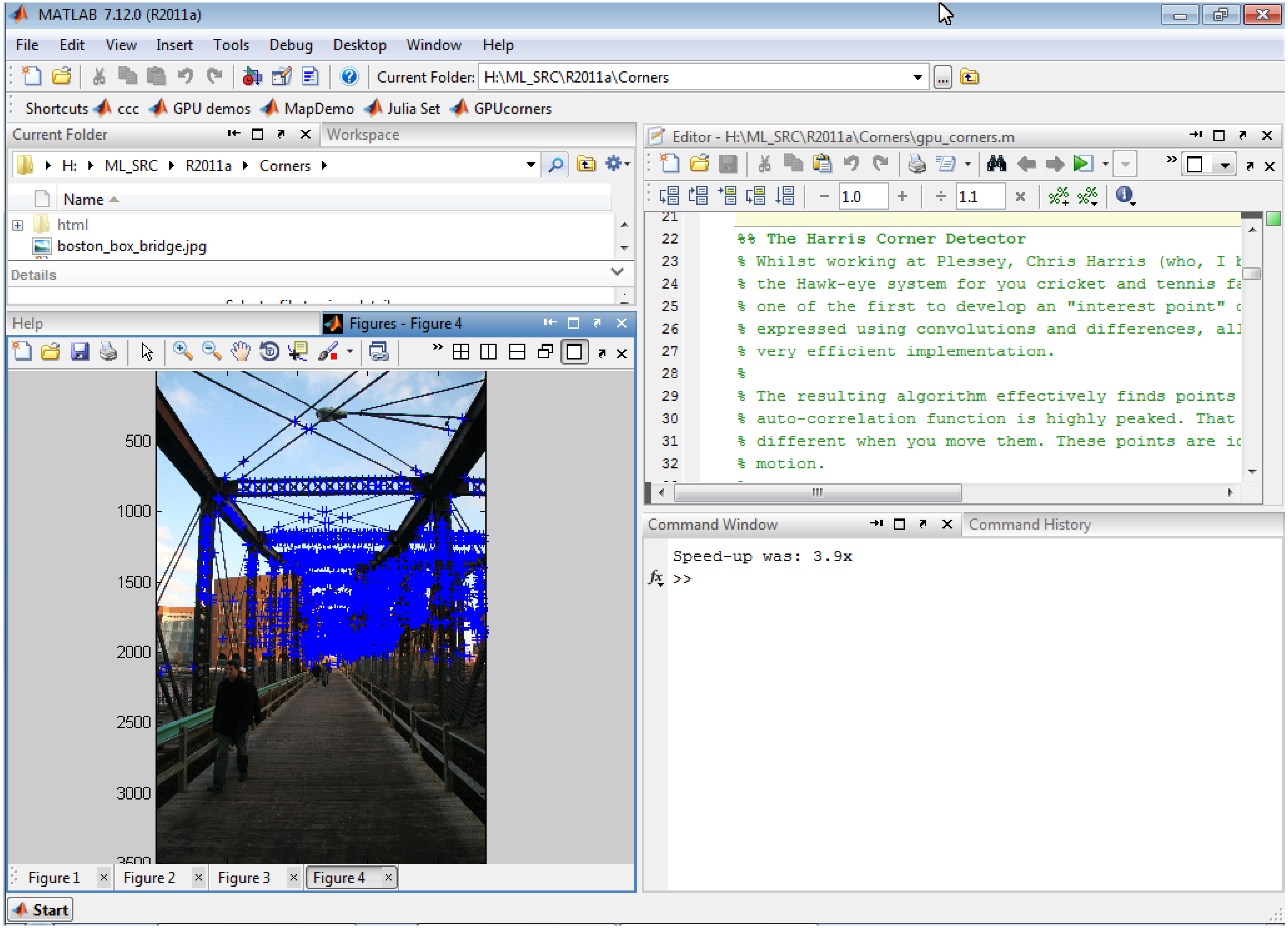
CPU vs GPU



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

Example: Corner Detection on the GPU



```
%% The Harris Corner Detector
% Whilst working at Plessey, Chris Harris (who, I k
% the Hawk-eye system for you cricket and tennis fa
% one of the first to develop an "interest point" c
% expressed using convolutions and differences, all
% very efficient implementation.
%
% The resulting algorithm effectively finds points
% auto-correlation function is highly peaked. That
% different when you move them. These points are ic
% motion.
```

```
Speed-up was: 3.9x
>>
```

Example: Corner Detection on the CPU

```
dx = cdata(2:end-1,3:end) - cdata(2:end-1,1:e
dy = cdata(3:end,2:end-1) - cdata(1:end-2,2:e
dx2 = dx.*dx;
dy2 = dy.*dy;
dxy = dx.*dy;

gaussHalfWidth = max( 1, ceil( 2*gaussSigma )
ssq = gaussSigma^2;
t = -gaussHalfWidth : gaussHalfWidth;
gaussianKernel1D = exp(-(t.*t)/(2*ssq))/(2*pi*ssq);     % The Gaussian 1D filter
gaussianKernel1D = gaussianKernel1D / sum(gaussianKernel1D);
smooth_dx2 = conv2( gaussianKernel1D, gaussianKernel1D, dx2, 'valid' );
smooth_dy2 = conv2( gaussianKernel1D, gaussianKernel1D, dy2, 'valid' );
smooth_dxy = conv2( gaussianKernel1D, gaussianKernel1D, dxy, 'valid' );

det = smooth_dx2 .* smooth_dy2 - smooth_dxy .* smooth_dxy;
trace = smooth_dx2 + smooth_dy2;
score = det - 0.25*edgePhobia*(trace.*trace);
```

1. Calculate derivatives

2. Smooth using convolution

3. Calculate score

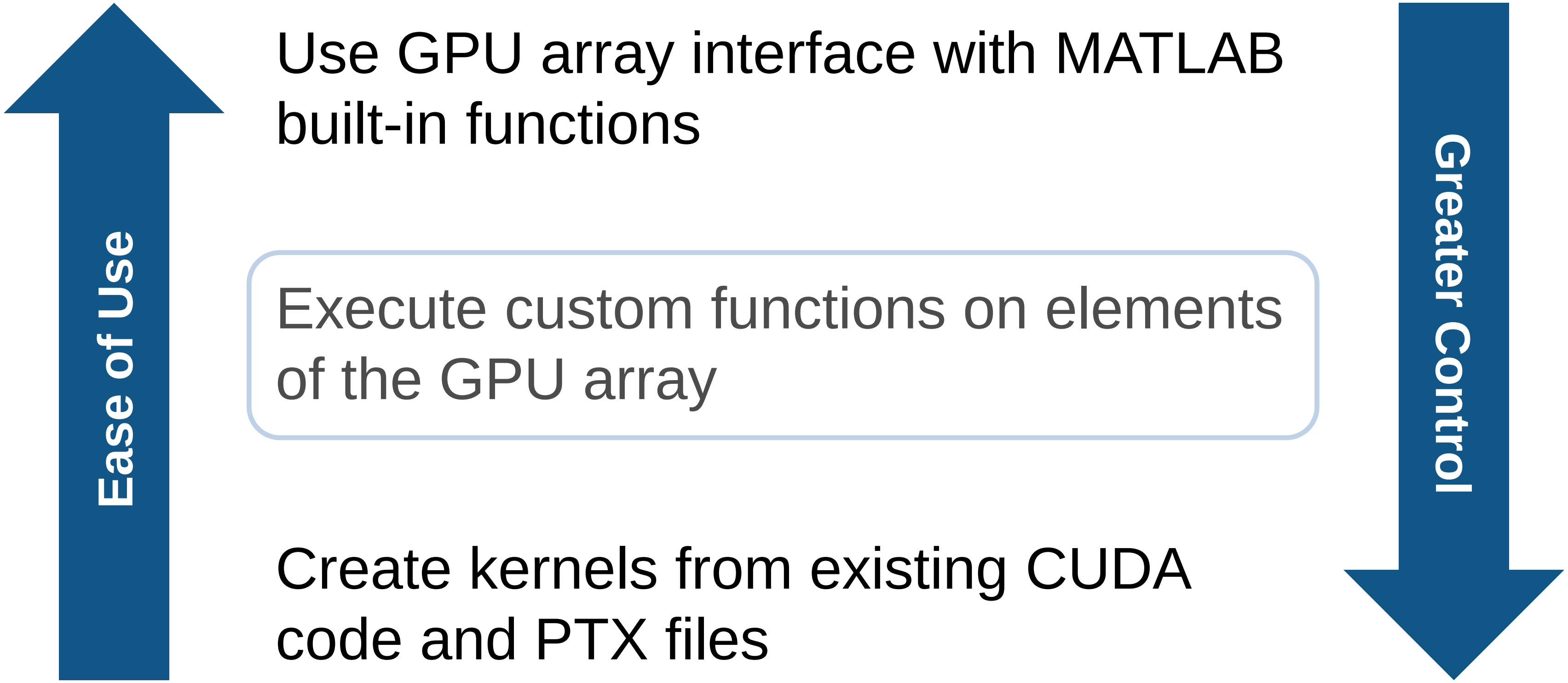
Example: Corner Detection on the GPU

```
cdata = gpuArray( cdata );

dx = cdata(2:end-1,3:end) - cdata(2:end-1,1:end-2);
dy = cdata(3:end,2:end-1) - cdata(1:end-2,2:end-1);
dx2 = dx.*dx;
dy2 = dy.*dy;
dxy = dx.*dy;

gaussHalfWidth = max( 1, ceil( 2*gaussSigma ) );
ssq = gaussSigma^2;
t = -gaussHalfWidth : gaussHalfWidth;
gaussianKernel1D = exp(-(t.*t)/(2*ssq))/(2*pi*ssq);     % The Gaussian 1D filter
gaussianKernel1D = gaussianKernel1D / sum(gaussianKernel1D);
smooth_dx2 = conv2( gaussianKernel1D, gaussianKernel1D, dx2, 'valid' );
smooth_dy2 = conv2( gaussianKernel1D, gaussianKernel1D, dy2, 'valid' );
smooth_dxy = conv2( gaussianKernel1D, gaussianKernel1D, dxy, 'valid' );

det = smooth_dx2 .* smooth_dy2 - smooth_dxy .* smooth_dxy;
trace = smooth_dx2 + smooth_dy2;
score = det - 0.25*edgePhobia*(trace.*trace);

score = gather( score );
```

0. Move data to GPU

4. Bring data back

Options for Targeting GPUs



Ease of Use

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

Greater Control

Using `arrayfun` on GPU

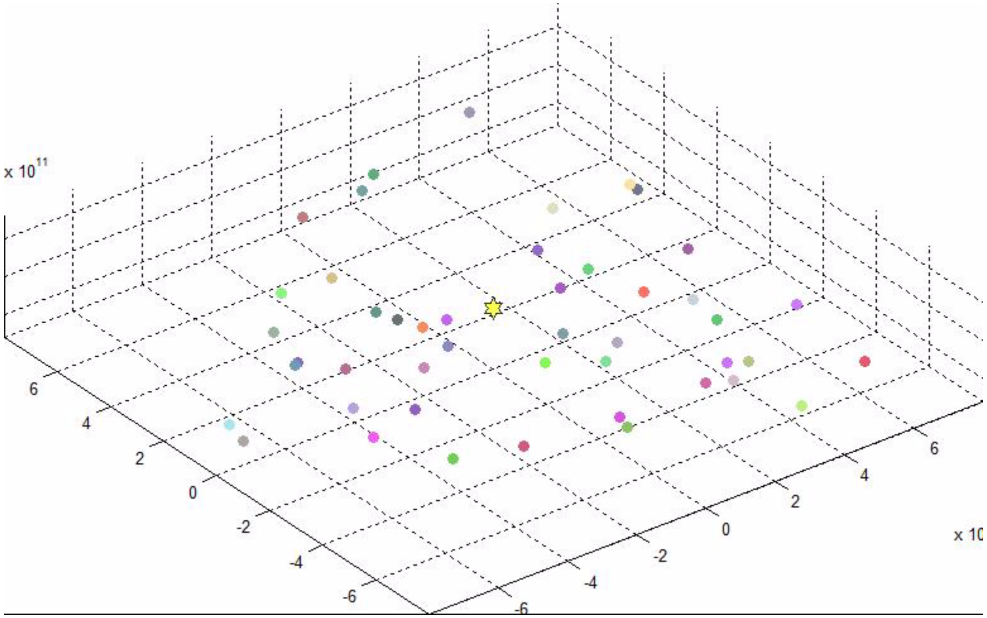
```
gain = 1.5;
offset = -0.1;
x = parallel.gpu.GPUArray.rand(1000,1); %Create on GPU
fh = @(x) myGPUfun(x, gain, offset);
x = arrayfun(fh, x)%Execute on GPU

function c = myGPUfun(x, gain, offset)
c = (x .* gain) + offset;
end
```

Full list of functions for use with `arrayfun` on GPU

User’s Guide → GPU Computing → Execute MATLAB Code on a GPU

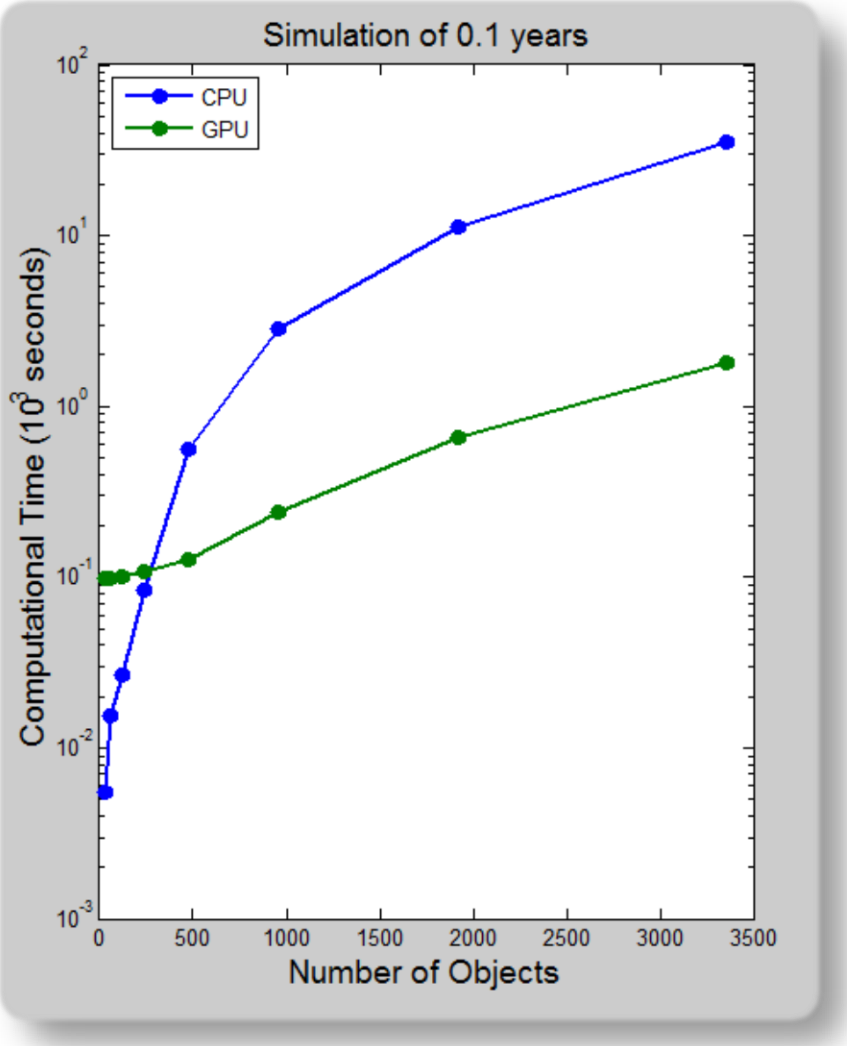
Example: N-Body Simulation

- Simulation of the mutual gravitational influence of (celestial) objects
- Compute orbits for a given number of bodies for a given length of time (in years)
- Using **`arrayfun`** and **`gpuArray`**



Benchmark: N-Body Simulation

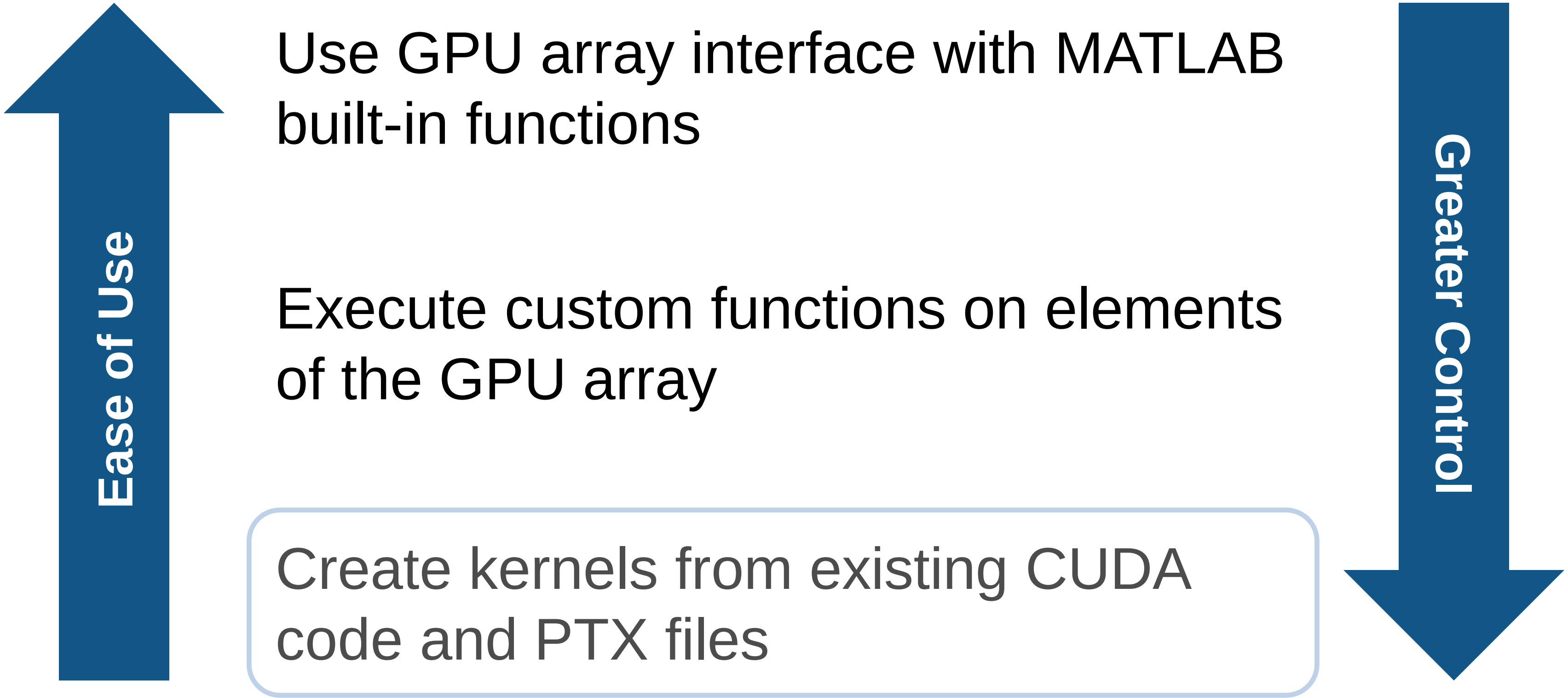
CPU vs GPU



Objects	CPU (10^3s)	GPU (10^3 s)	Speed up
60	0.015	0.099	0.15
120	0.027	0.099	0.27
240	0.083	0.108	0.76
480	0.559	0.126	4.42
960	2.83	0.241	11.77
1920	11.3	0.655	17.17
3360	35.3	1.822	19.38

Intel Xeon Processor W3550, NVIDIA Tesla C2050 GPU

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

Webinar: “GPU Computing with MATLAB”
http://www.mathworks.com/company/events/webinars

Invoking CUDA Kernels

```
% Setup
kernel = parallel.gpu.CUDAKernel(‘myKern.ptx’,‘myKern.cu’);

% Configure
kernel.ThreadBlockSize = 512;
kernel.GridSize = [2 2];

% Run
[c, d] = feval(kernel, a, b);
```

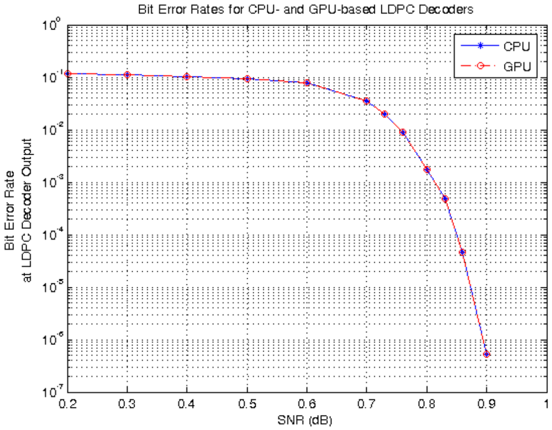
Best Practices for using GPU with MATLAB

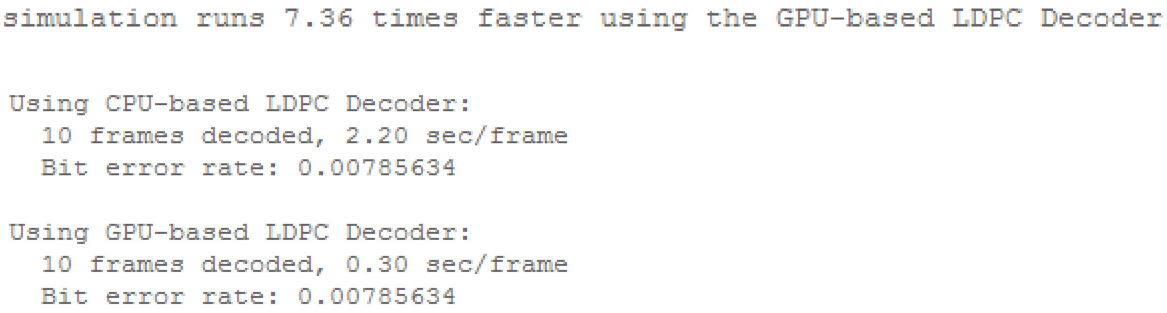
- Profile your code to identify your bottlenecks
- Work on large enough matrices to see the benefits of GPU parallelization
- Minimize data transfer between CPU and GPU
  - Sustained use of supported functionality
  - Create variables directly on the GPU
- Combine multiple element-wise calculations together into a single function call by using **`arrayfun`**

Support for Communications System Toolbox

- GPU implementations of LDPC Decoder, Viterbi Decoder, AWGN Channel, PSK Modulator, Block Interleaver, Block Deinterleaver

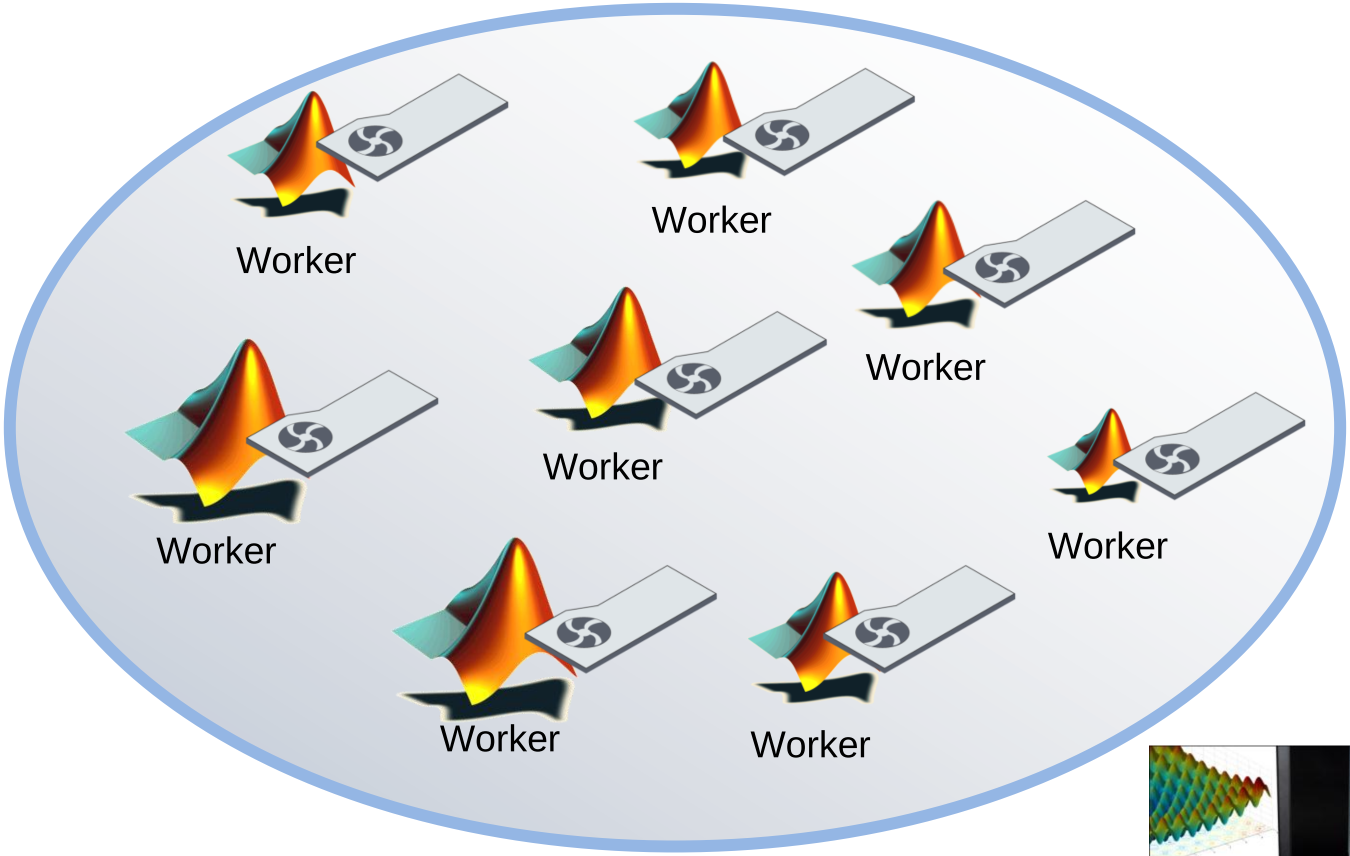
- DVB-S System Simulation Demo
  http://www.mathworks.com/products/communications/demos.html





```
simulation runs 7.36 times faster using the GPU-based LDPC Decoder


Using CPU-based LDPC Decoder:
  10 frames decoded, 2.20 sec/frame
  Bit error rate: 0.00785634

Using GPU-based LDPC Decoder:
  10 frames decoded, 0.30 sec/frame
  Bit error rate: 0.00785634
```

Scaling Up to Run on Multiple GPUs



Summary GPU Functionality

- GPU array data type
  - Store arrays in GPU device memory
  - Algorithm support for over 100 functions
  - Integer and double support

- GPU functions
  - Invoke element-wise MATLAB functions on the GPU

- CUDA kernel interface
  - Invoke CUDA kernels directly from MATLAB
  - No MEX programming necessary

Additional Resources

- MATLAB documentation
  - MATLAB → Programming Fundamentals → Performance
- GPU Demos and Benchmarks
  - http://www.mathworks.com/products/parallel-computing/demos.html
- A Mandelbrot Set on The GPU
  - http://blogs.mathworks.com/loren/2011/07/18/a-mandelbrot-set-on-the-gpu/
- GPU Programming in MATLAB
  - http://www.mathworks.com/company/newsletters/articles/gpu-programming-in-matlab.html
- Parallel Computing with MATLAB on Multicore Desktops and GPUs
  - http://www.mathworks.com/company/events/webinars/wbnr56334.html

Agenda

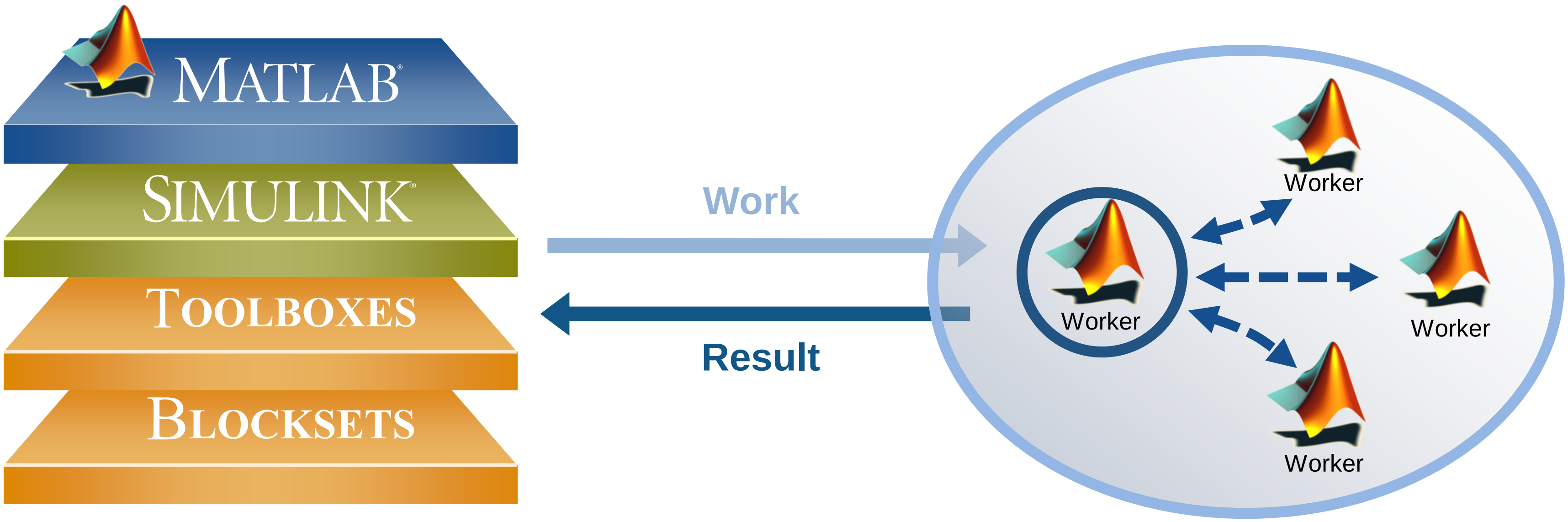
13:30	**Welcome and Introduction**
13:45	**Introduction to Parallel Computing with MATLAB**
	MATLAB–extensions with built-in support for Parallel Computing
14:15	**Interactive development of task- and data-parallel Algorithms**
15:15	***Coffee Break***
15:30	**GPU programming with MATLAB**
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	**Q&A Session**
17:00	***End of Seminar***

Interactive to Scheduling

- Interactive
  - Great for prototyping
  - Immediate access to MATLAB workers

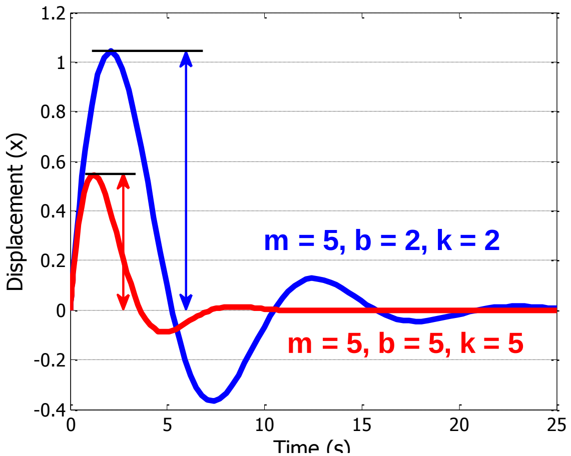
- Scheduling
  - Offloads work to other MATLAB workers (local or on a cluster)
  - Access to more computing resources for improved performance
  - Frees up local MATLAB session

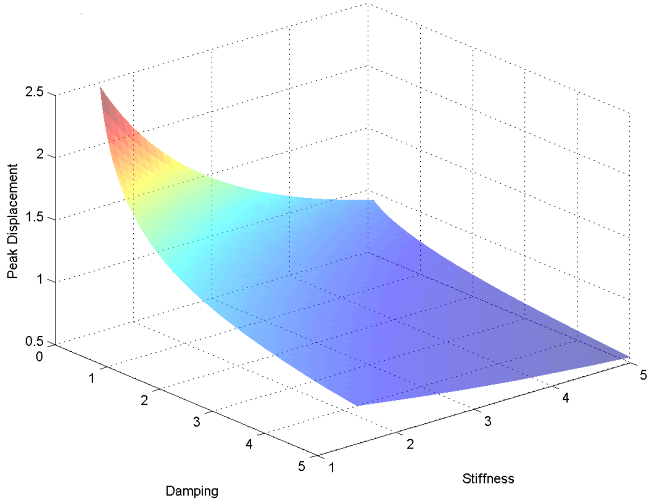
Scheduling Scripts and Functions with `batch`



Example: Schedule Processing

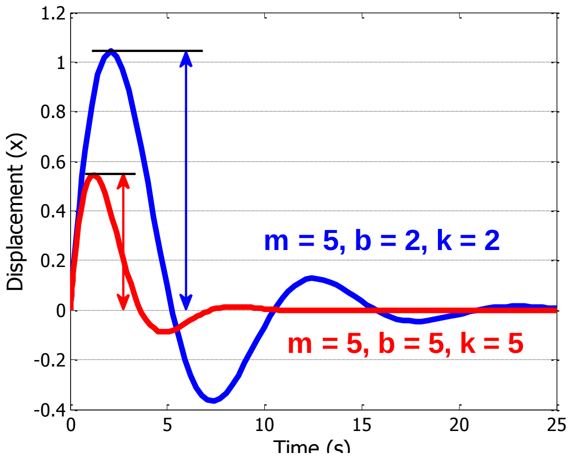
- Offload parameter sweep to local workers
- Get peak value results when processing is complete
- Plot results in local MATLAB

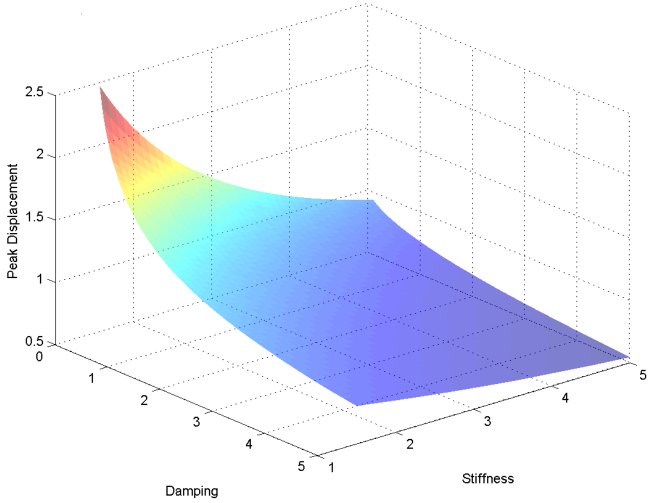




Summary of Example

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

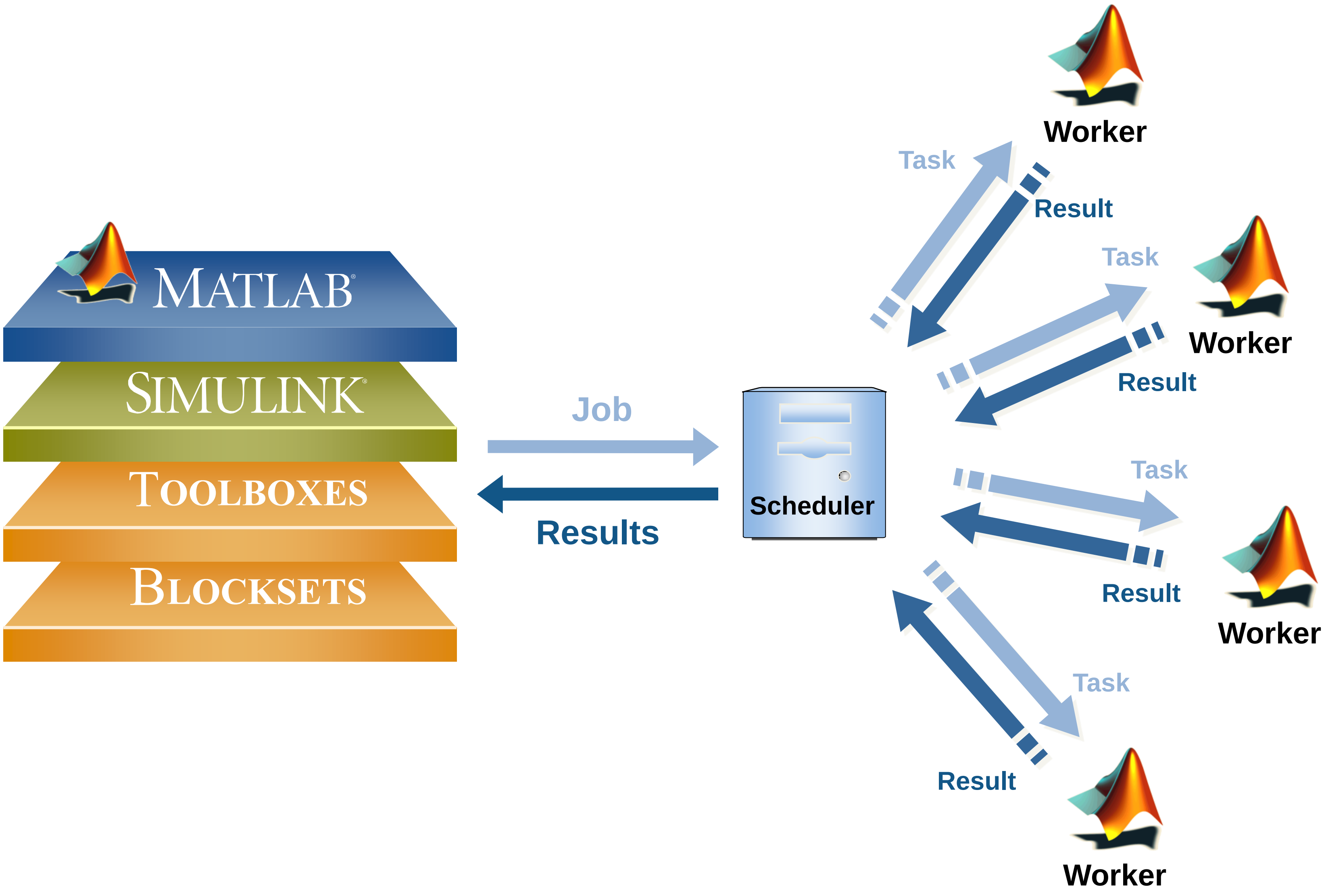




Agenda

13:30	Welcome and Introduction
13:45	Introduction to Parallel Computing with MATLAB
	MATLAB–extensions with built-in support for Parallel Computing
14:15	Interactive development of task- and data-parallel Algorithms
15:15	*Coffee Break*
15:30	GPU programming with MATLAB
	Parallel batch-jobs
	Cluster Computing with MATLAB
16:15	Q&A Session
17:00	*End of Seminar*

Scheduling Jobs and Tasks



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`

Optimal Number of Tasks

	Short Execution Time	**Long Execution Time**
Few Function Calls	Local Sequential Execution	Distributed (One task per function call)
Many Function Calls	Distributed (Aggregate function calls in tasks)	Distributed

When to Use `parfor` vs. jobs and tasks

`parfor`

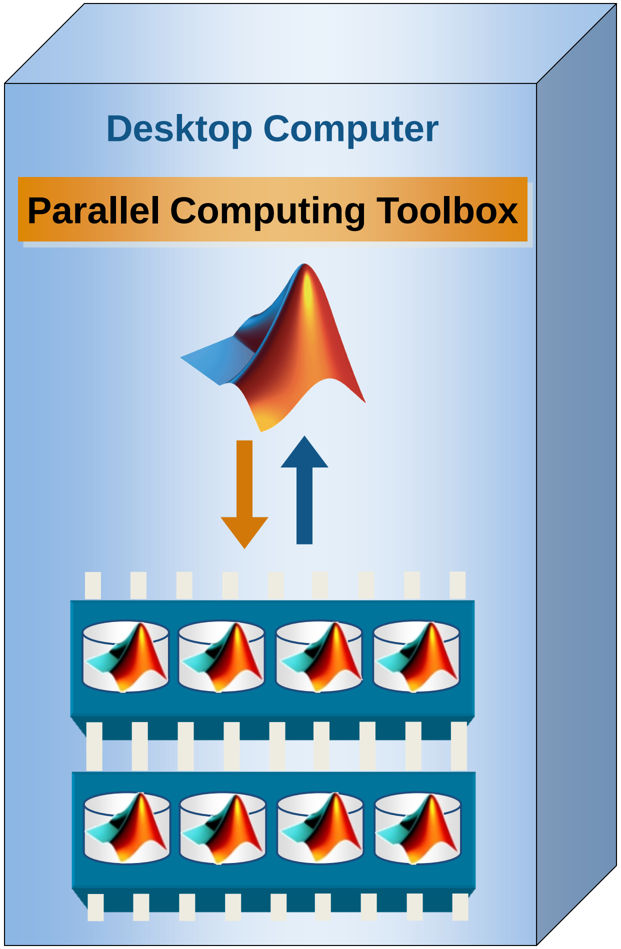
- Seamless integration to user’s code
- Several **`for`** loops throughout the code to convert
- Automatic load balancing

Jobs and tasks

- Explicit control
- Query results after each task is finished

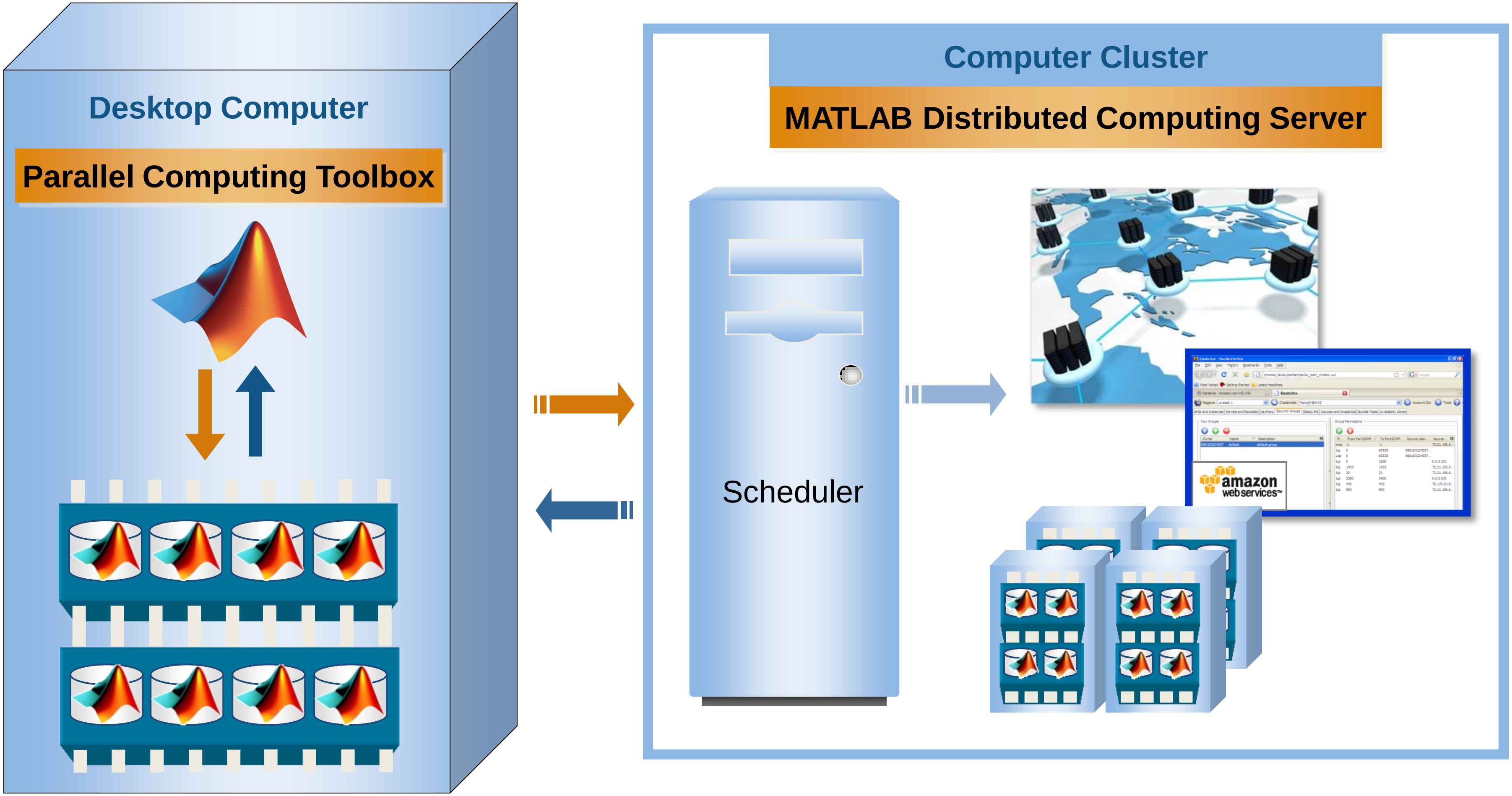
*Try **`parfor`** first. If it doesn’t apply to your application, create jobs and tasks.*

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



Desktop Computer

Parallel Computing Toolbox

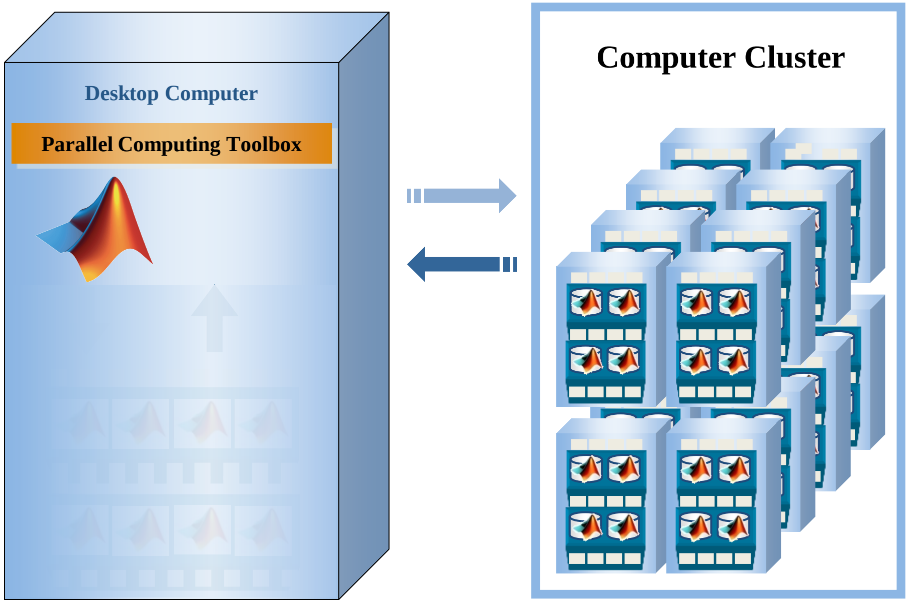
Computer Cluster

MATLAB Distributed Computing Server

Scheduler

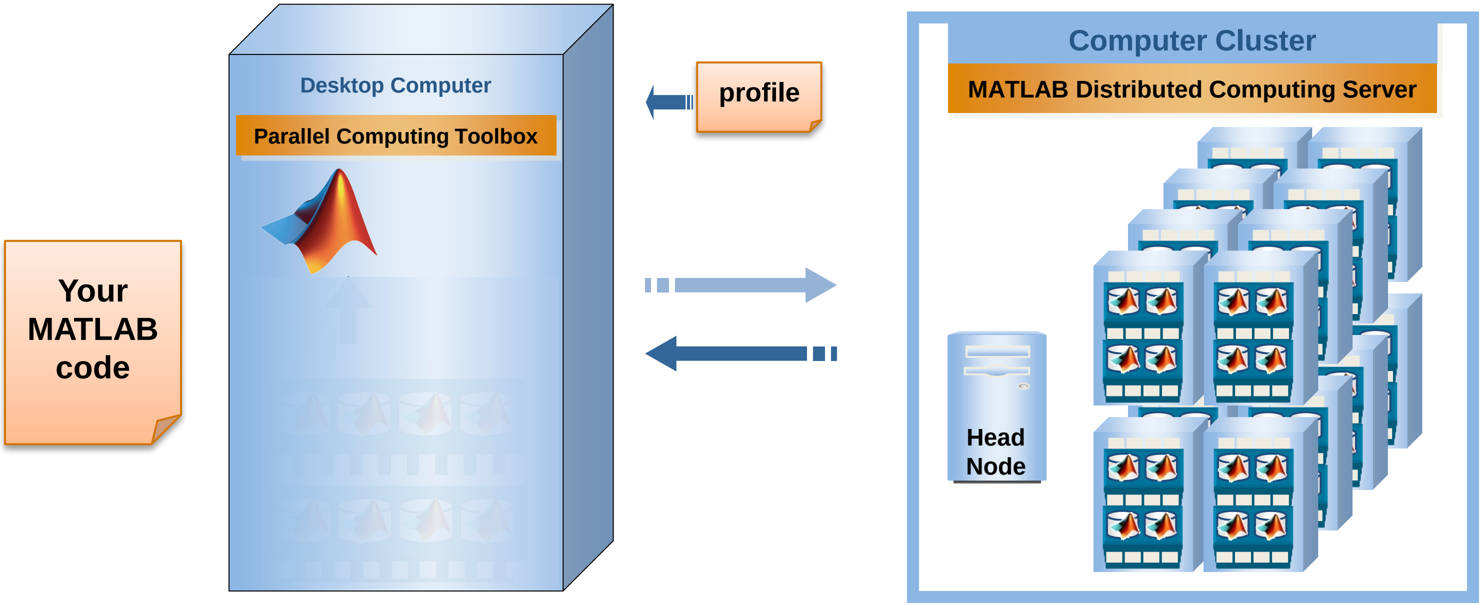
Moving beyond the desktop

- Offload Computation:
  - Free up desktop
  - Access better computers
- Scale speed-up:
  - Use more cores
  - Go from hours to minutes
- Scale memory:
  - Utilize distributed arrays
  - Solve larger problems without re-coding



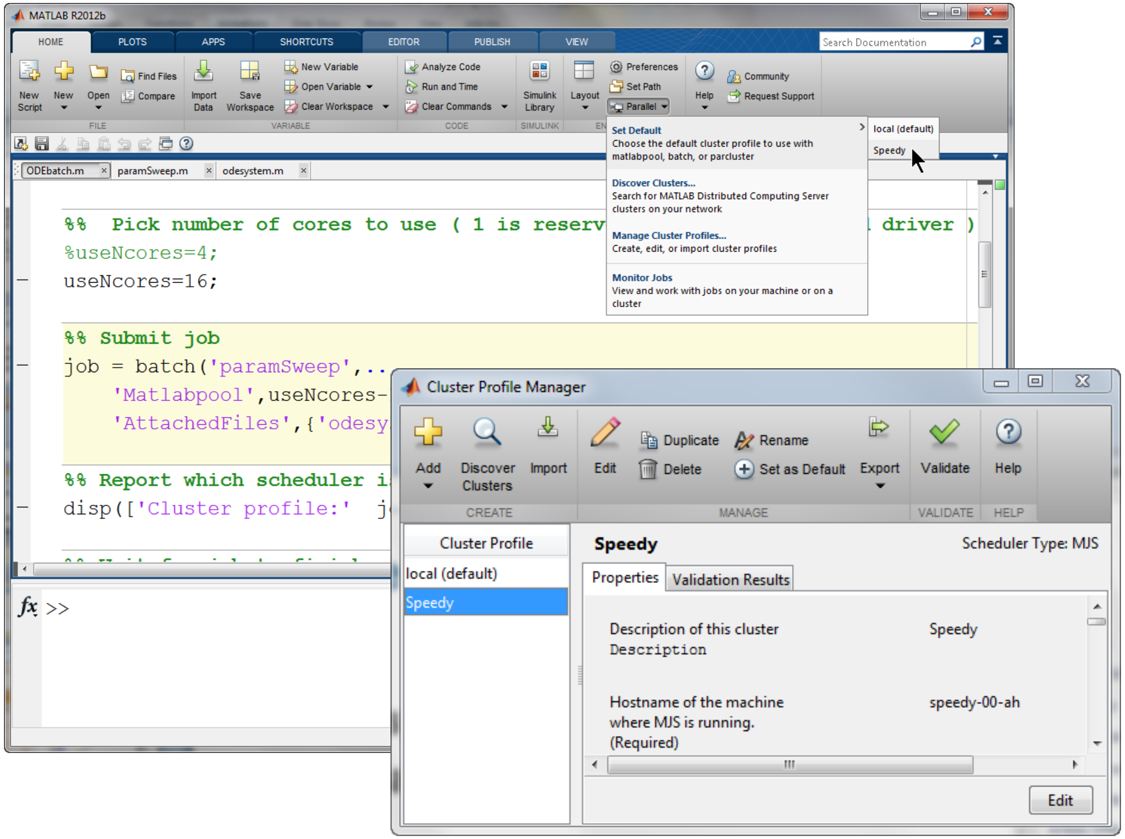
Utilize MATLAB Distributed Computing Server

1. Prototype code
2. Switch cluster profile
3. Utilize cluster



Migrate from Desktop to Cluster

- Change hardware without changing algorithmic code



Set Default
Choose the default cluster profile to use with matlabpool, batch, or parcluster

Discover Clusters...
Search for MATLAB Distributed Computing Server clusters on your network

Manage Cluster Profiles...
Create, edit, or import cluster profiles

Monitor Jobs
View and work with jobs on your machine or on a cluster

```
%%  Pick number of cores to use ( 1 is reserv                    driver )
%useNcores=4;
useNcores=16;

%% Submit job
job = batch('paramSweep',..
    'Matlabpool',useNcores-
    'AttachedFiles',{'odesy

%% Report which scheduler i
disp(['Cluster profile:'  j
```

```
>>
```

Cluster Profile:
- local (default)
- Speedy

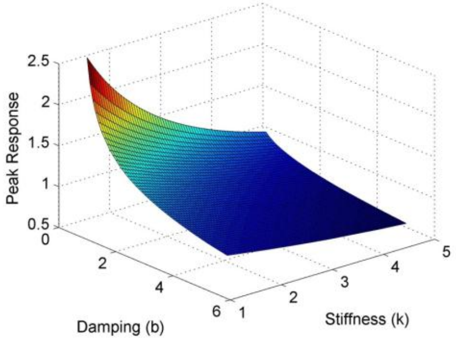
Speedy Scheduler Type: MJS

Description of this cluster Description	Speedy
Hostname of the machine where MJS is running. (Required)	speedy-00-ah

Parameter Sweep of ODEs

Parallel for-loops

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

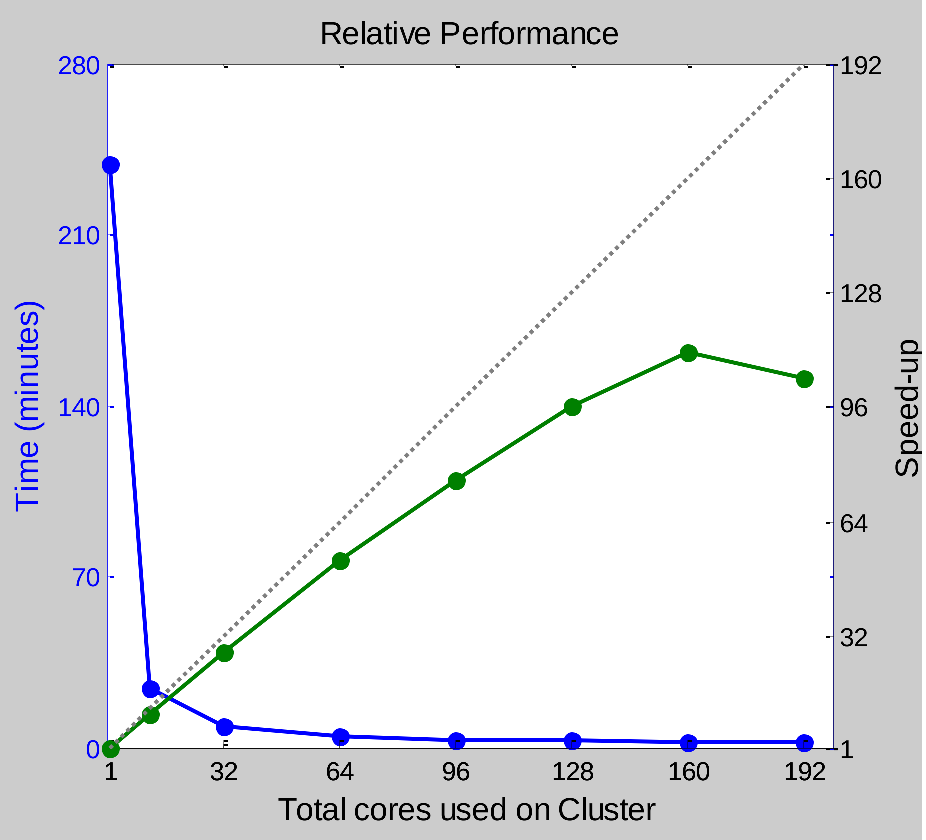


Benchmark: Parameter Sweep of ODEs

Changing number of cores used on cluster

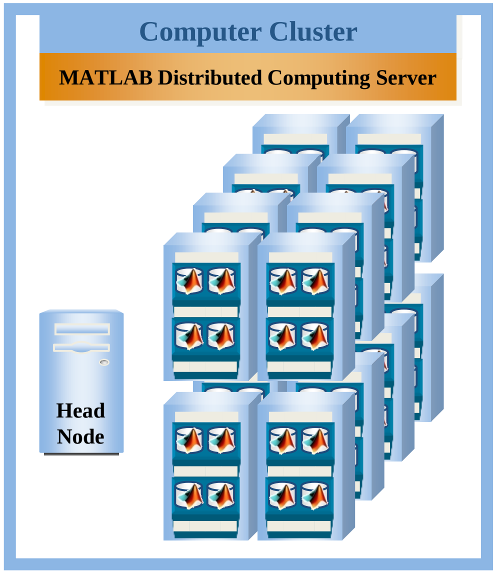
Cluster cores	Job Time minutes	Speedup
1	239 (4 hrs)	-
12	24	10
32	8.7	28
64	4.5	53
96	3.2	75
128	2.5	96
160	2.1	111
192	2.3	104

Processor: Intel Xeon E5-2670

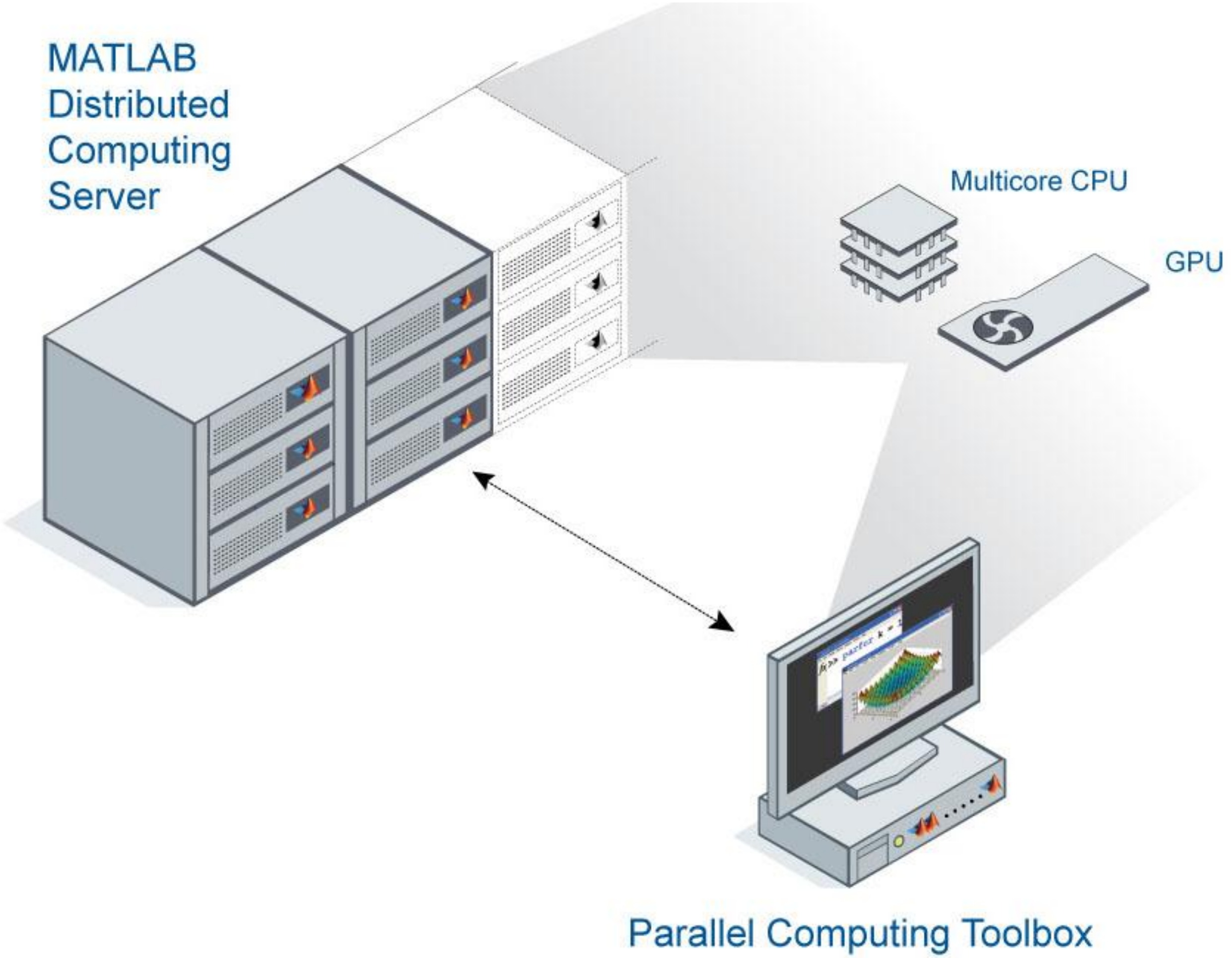


MATLAB Distributed Computing Server

- Extension of Parallel Computing Toolbox
- Complete pre-built solution
  - Framework and infrastructure
  - Communication between computers
- Cost-effective
  - License for number of cores you will use
  - Simplified maintenance



Parallel Computing Products


MATLAB Distributed Computing Server
Multicore CPU
GPU
>> parfor k = 1
Parallel Computing Toolbox

Summary Parallel Computing with MATLAB

Simple and portable

- Straightforward program speed up
- Interactive parallel programming
- Portable code

Scalable

- Support parallelism on desktop
- Treat large resource as extensions of desktop

HPC Hardware Leverage

- Supports multicore, multi-CPU, GPUs, Clusters, Grids and Clouds

Deployable

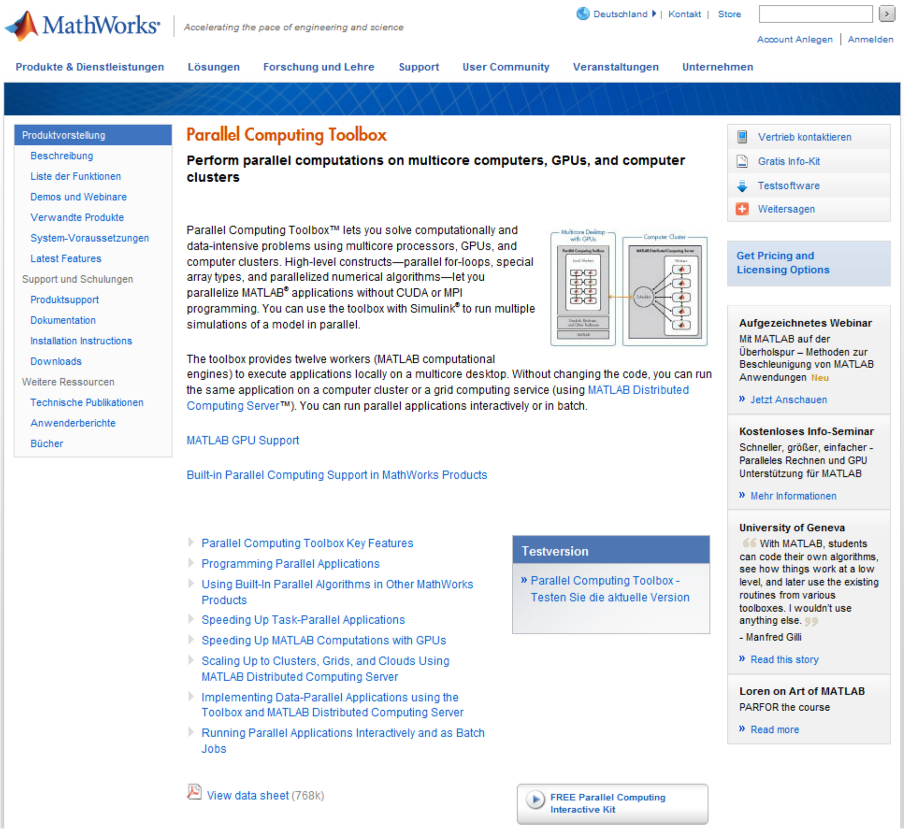
- Simple path from development to standalone application
- Supported for CPU and GPU

Integrated into organization

- Dynamic licensing
- Support for third-party schedulers

More information available on the Web

`www.mathworks.com/parallel-computing`



Parallel Computing Toolbox

Perform parallel computations on multicore computers, GPUs, and computer clusters

Parallel Computing Toolbox™ lets you solve computationally and data-intensive problems using multicore processors, GPUs, and computer clusters. High-level constructs—parallel for-loops, special array types, and parallelized numerical algorithms—let you parallelize MATLAB® applications without CUDA or MPI programming. You can use the toolbox with Simulink® to run multiple simulations of a model in parallel.

The toolbox provides twelve workers (MATLAB computational engines) to execute applications locally on a multicore desktop. Without changing the code, you can run the same application on a computer cluster or a grid computing service (using MATLAB Distributed Computing Server™). You can run parallel applications interactively or in batch.

MATLAB GPU Support

Built-in Parallel Computing Support in MathWorks Products

- Parallel Computing Toolbox Key Features
- Programming Parallel Applications
- Using Built-In Parallel Algorithms in Other MathWorks Products
- Speeding Up Task-Parallel Applications
- Speeding Up MATLAB Computations with GPUs
- Scaling Up to Clusters, Grids, and Clouds Using MATLAB Distributed Computing Server
- Implementing Data-Parallel Applications using the Toolbox and MATLAB Distributed Computing Server
- Running Parallel Applications Interactively and as Batch Jobs

View data sheet (768k)

Testversion

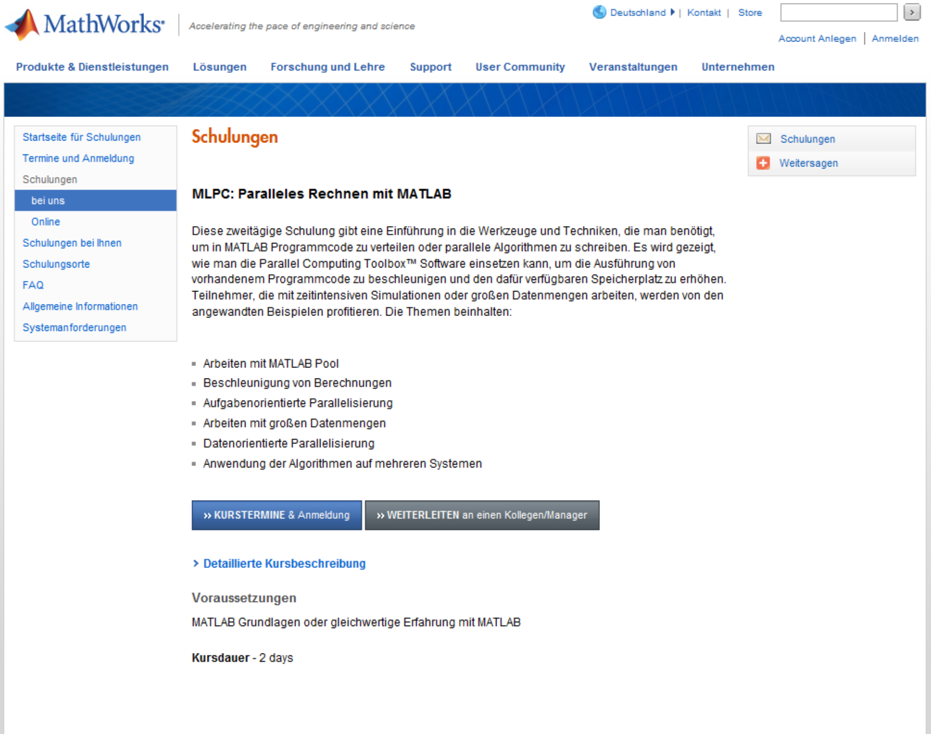
» Parallel Computing Toolbox - Testen Sie die aktuelle Version

Aufgezeichnetes Webinar
Mit MATLAB auf der Überholspur – Methoden zur Beschleunigung von MATLAB Anwendungen Neu
» Jetzt Anschauen

Kostenloses Info-Seminar
Schneller, größer, einfacher - Paralleles Rechnen und GPU Unterstützung für MATLAB
» Mehr Informationen

University of Geneva
"With MATLAB, students can code their own algorithms, see how things work at a low level, and later use the existing routines from various toolboxes. I wouldn't use anything else."
- Manfred Gilli
» Read this story

Loren on Art of MATLAB
PARFOR the course
» Read more



Schulungen

MLPC: Paralleles Rechnen mit MATLAB

Diese zweitägige Schulung gibt eine Einführung in die Werkzeuge und Techniken, die man benötigt, um in MATLAB Programmcode zu verteilen oder parallele Algorithmen zu schreiben. Es wird gezeigt, wie man die Parallel Computing Toolbox™ Software einsetzen kann, um die Ausführung von vorhandenem Programmcode zu beschleunigen und den dafür verfügbaren Speicherplatz zu erhöhen. Teilnehmer, die mit zeitintensiven Simulationen oder großen Datenmengen arbeiten, werden von den angewandten Beispielen profitieren. Die Themen beinhalten:

- Arbeiten mit MATLAB Pool
- Beschleunigung von Berechnungen
- Aufgabenorientierte Parallelisierung
- Arbeiten mit großen Datenmengen
- Datenorientierte Parallelisierung
- Anwendung der Algorithmen auf mehreren Systemen

› Detaillierte Kursbeschreibung

Voraussetzungen

MATLAB Grundlagen oder gleichwertige Erfahrung mit MATLAB

Kursdauer - 2 days