

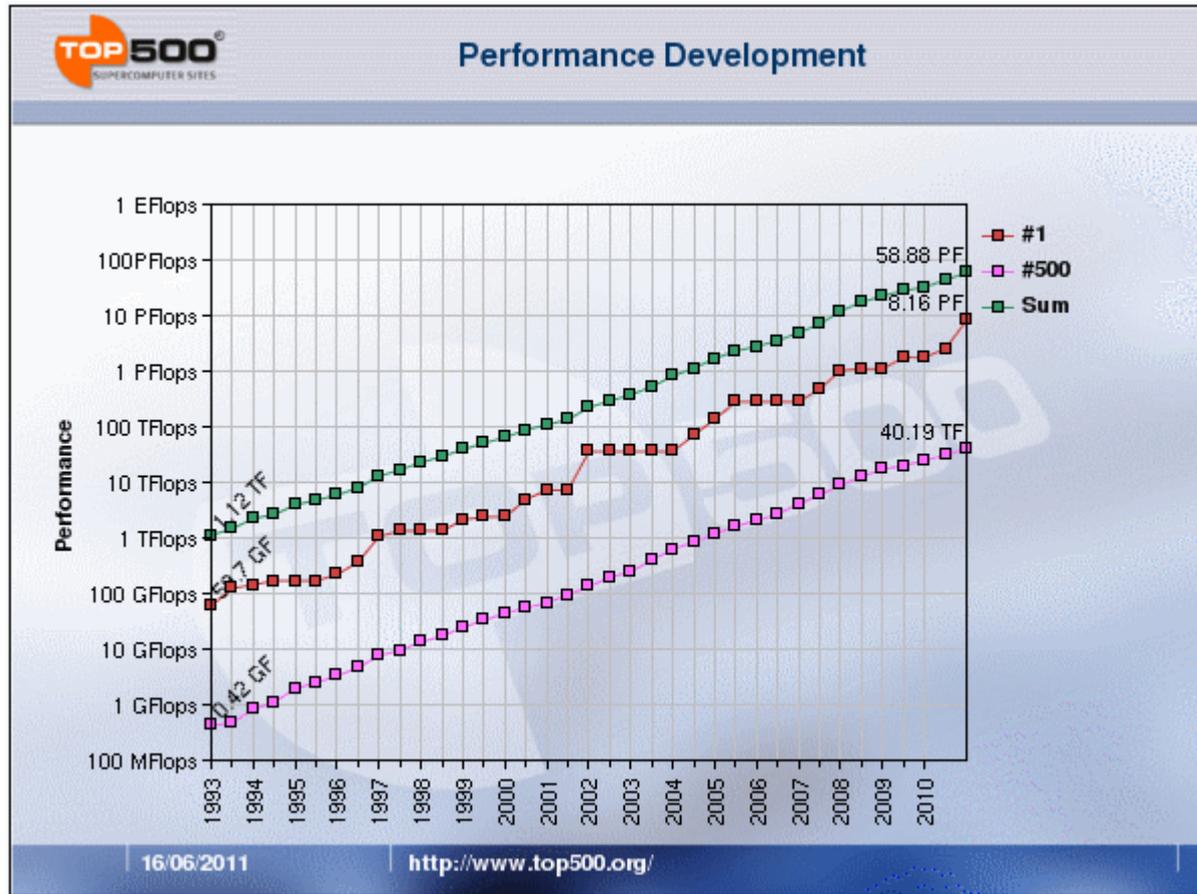
# Taxes, Death, and **Parallelism** are Inevitable



# Introduction to Parallel Computing

- Abstract
- Overview
  - What is Parallel Computing?
  - Why Use Parallel Computing?
- Concepts and Terminology
  - von Neumann Computer Architecture
  - Flynn's Classical Taxonomy
  - Some General Parallel Terminology

# The FUTURE: *The race is already on for Exascale Computing!*



### Computer performance

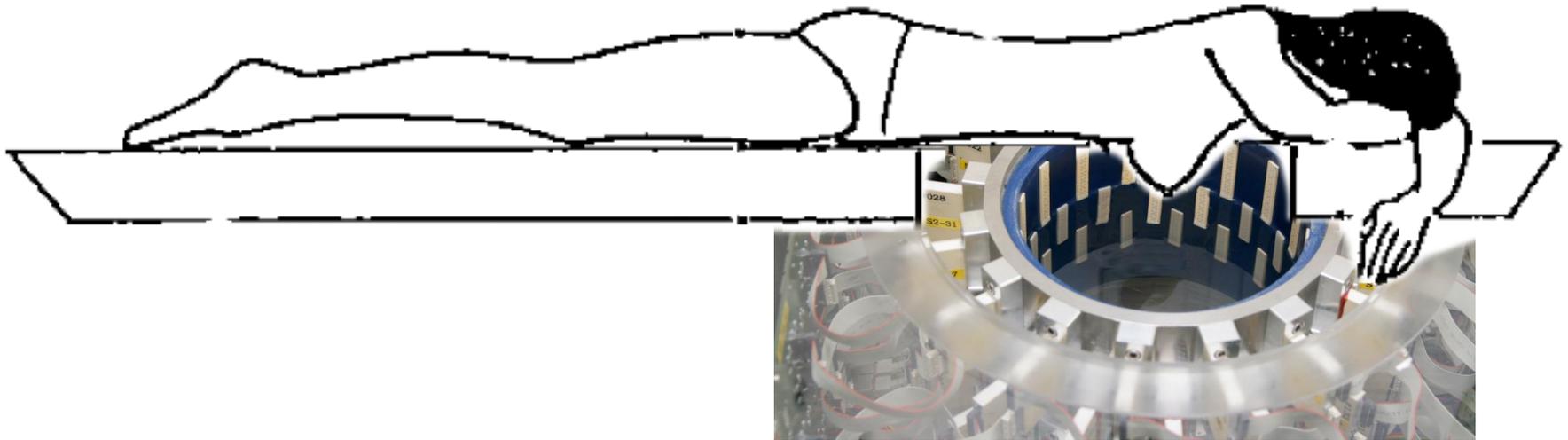
Name	FLOPS
yottaFLOPS	$10^{24}$
zettaFLOPS	$10^{21}$
exaFLOPS	$10^{18}$
petaFLOPS	$10^{15}$
teraFLOPS	$10^{12}$
gigaFLOPS	$10^9$
megaFLOPS	$10^6$
kiloFLOPS	$10^3$

# Ultrasound Computer Tomography (USCT)

***<http://www.interactive-grid.eu>***

- New method for medical imaging
  - Focus: Breast cancer diagnosis

# USCT setup



# USCT Algorithm

- Characteristics:
  - Input: 20 GB (full set)
  - Computing time depends
    - on output size / resolution
    - amount of input data

35MB	20GB	20GB	Data
4096 <sup>2</sup>	128 <sup>2</sup> x100	4096 <sup>2</sup> x3410	Voxels
1 Hour	1.5 Months	150 Years	Time

- Matlab
  - Strategic development platform (95% sourcecode)

# USCT Algorithm

- Characteristics:
  - Input: 20 GB (full set)
  - Computing time depends
    - on output size / resolution
    - amount of input data

35MB	20GB	20GB	Data
4096 <sup>2</sup>	128 <sup>2</sup> x100	4096 <sup>2</sup> x3410	Voxels
1 Hour	1.5 Months	150 Years	Time

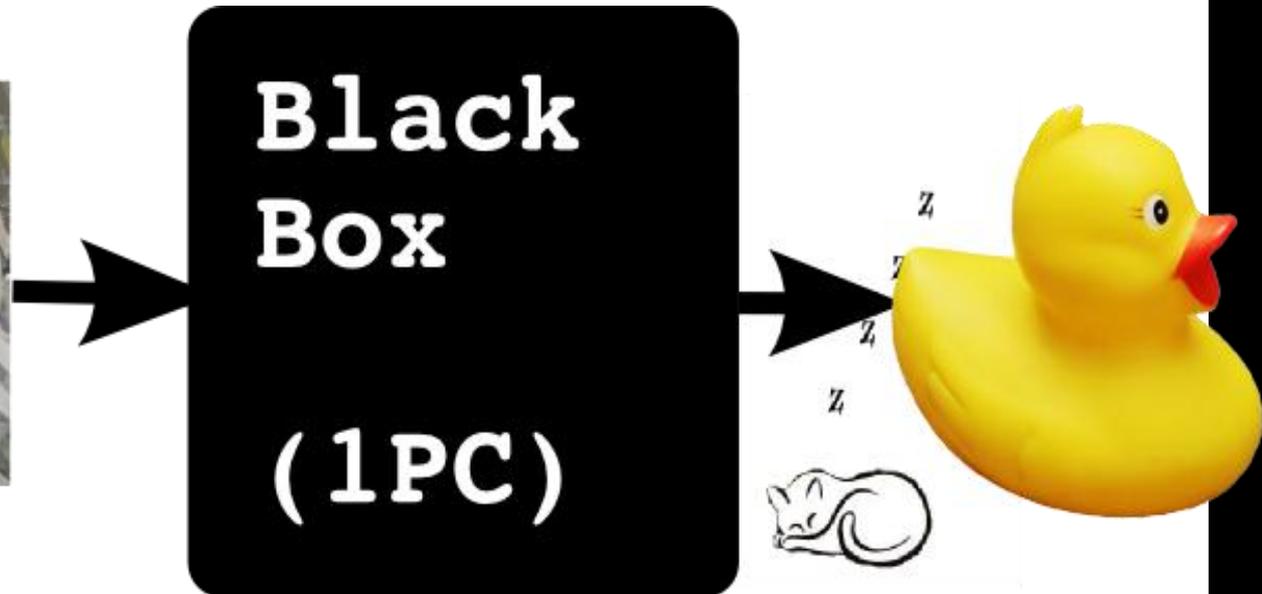
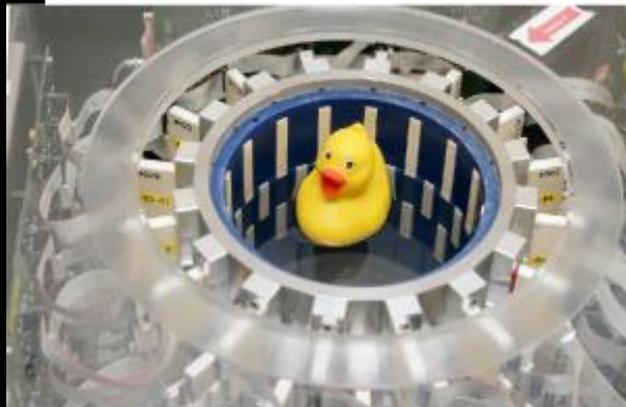
## • Matlab

### Goals for grid access: •

- Seamless •
- Interactive •
- from Matlab •

platform (95%)

# USCT reconstruction := “Black Box”



- Computation takes long (days, weeks, years)
- Grid in order to speed up

Idea: **Computer power  $\Leftrightarrow$  Electrical power**

From Electrical power grid  $\Rightarrow$  computational grid

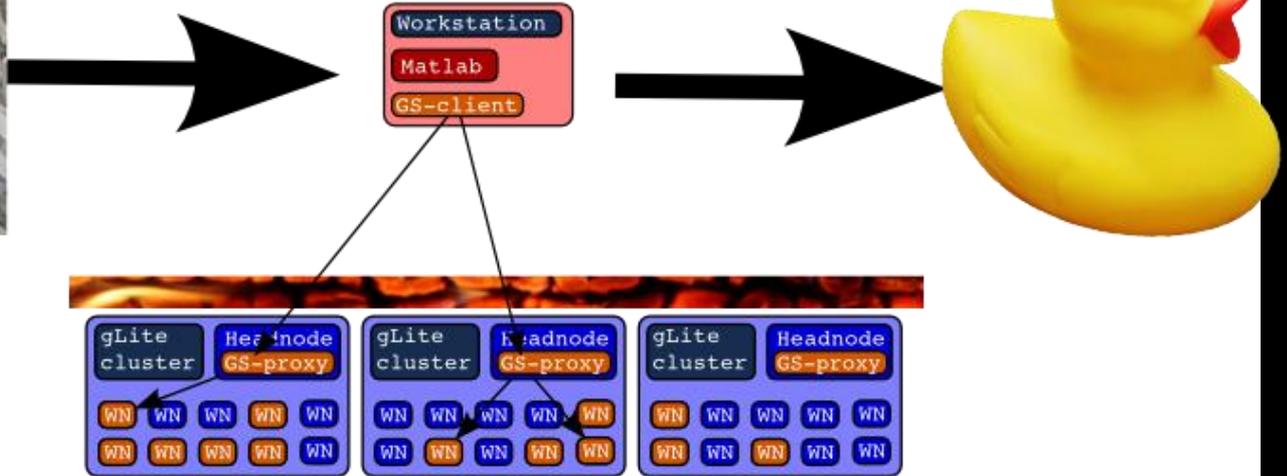
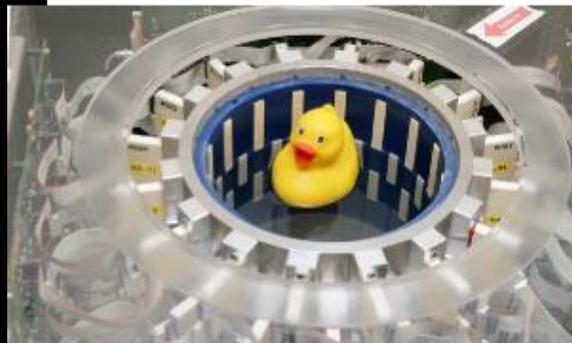
- Across organisational domains / countries
- Transparent access to
  - Computing
  - Data
  - Network
- Large scale installations

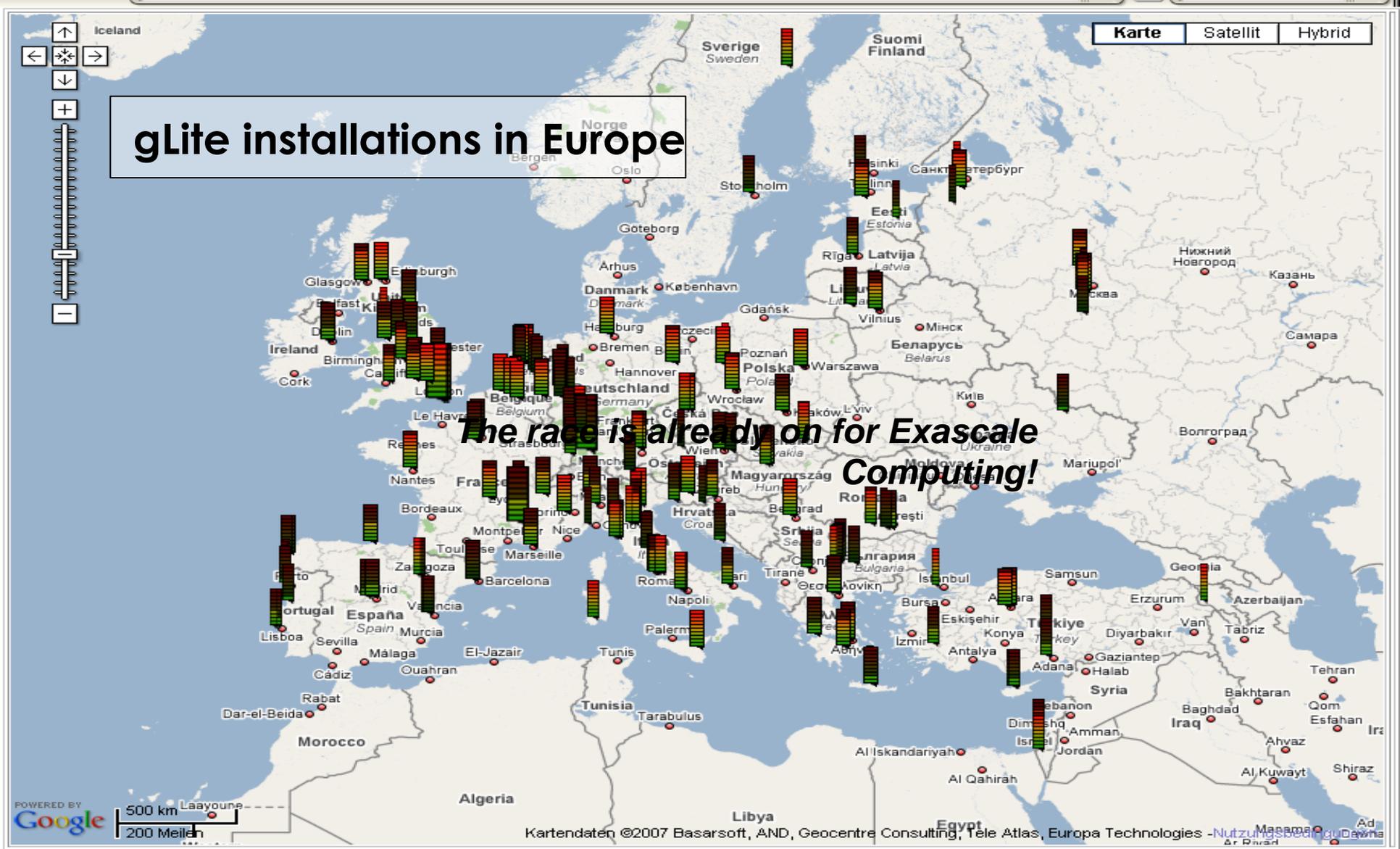
# Grid middleware

---

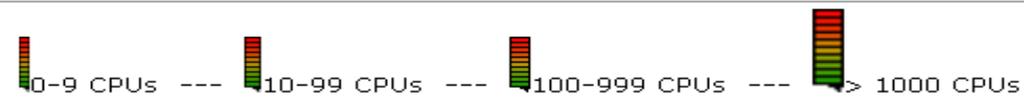
- Middleware
  - := Layer between application and operating system
- **gLite**: one grid middleware
  - Development driven by CERN
  - Tools for data+computing of new accelerator
  - 10 TB/year \* 20 years, random access
- Paradigm: **Send job to where the data is**
- Job: Self contained application

# Putting things together



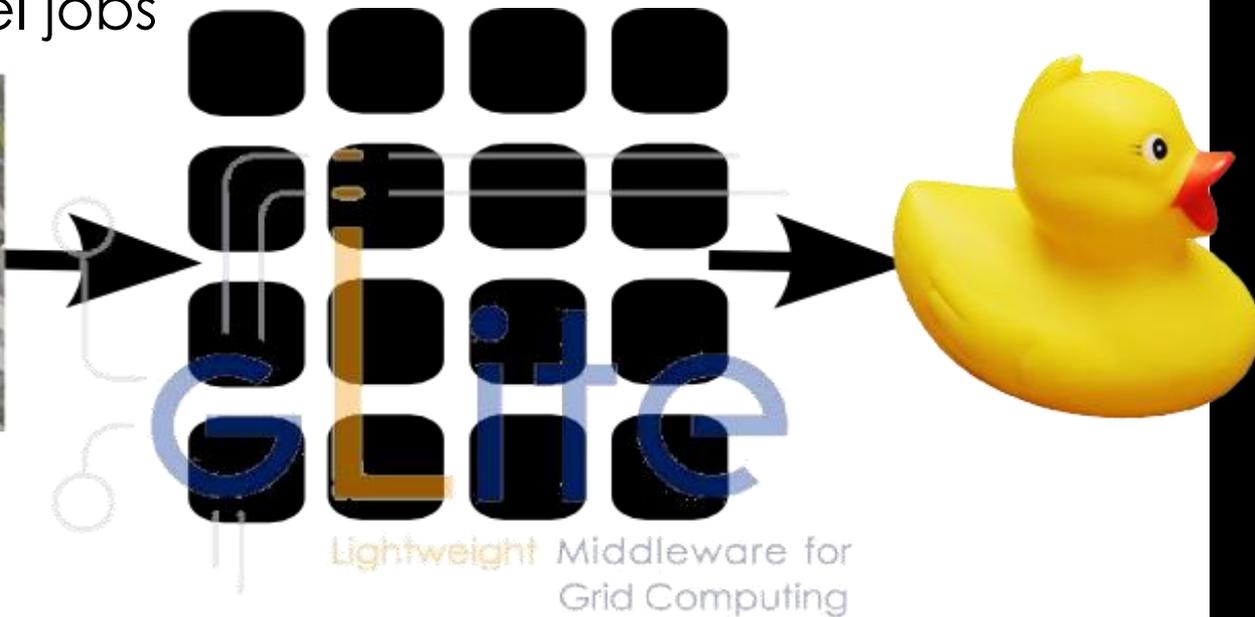
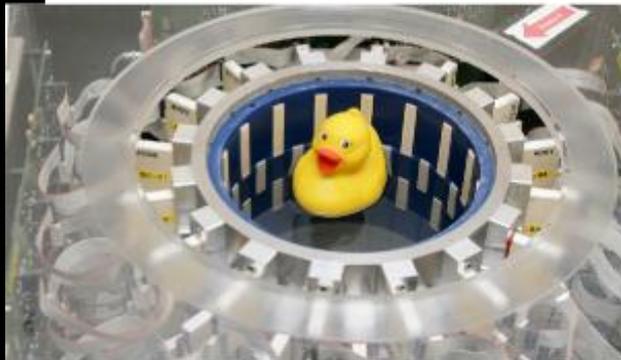


- Sites: 243 (in 49 countries)
- CPUS: 42798 (176 per site)
- RAM: 19TB
- RAM/CPU: 468MB
- DISK [Tot / Avail]: [8042TB / 5408TB] ([33892GB / 22792GB] per site)

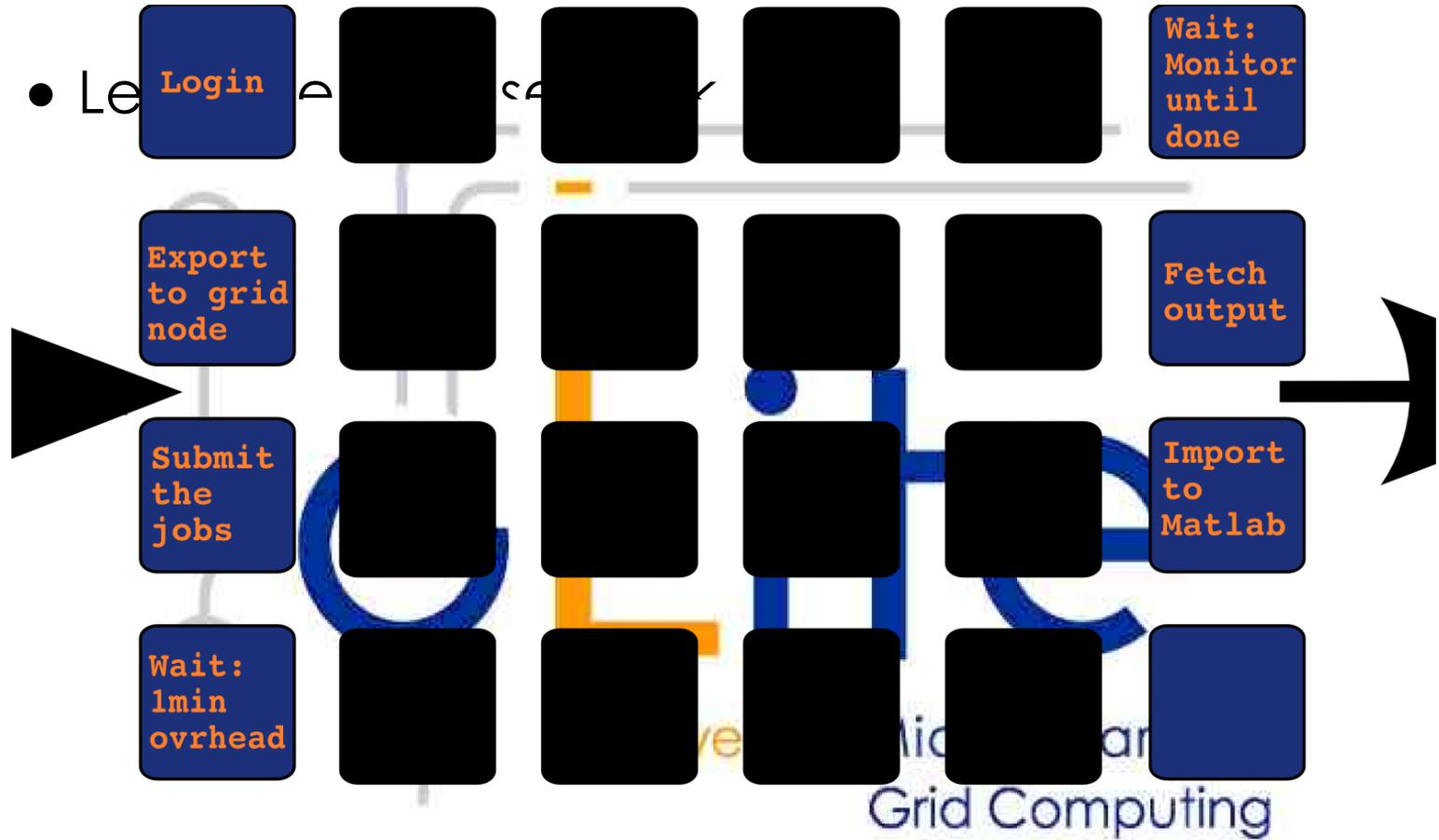


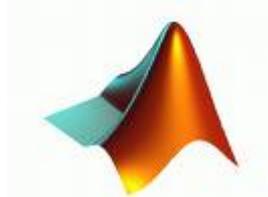
# Using gLite

- Initial approach to parallel execution:
  - Partitioning of data
  - Many parallel jobs



# Using gLite





# Parallel Matlab

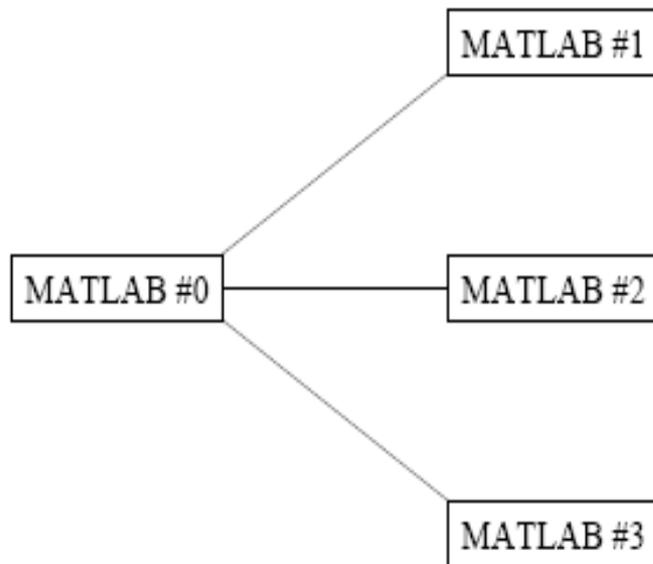
Elias Houstis

# About MATLAB

- MATLAB (with Simulink) – programming language for science and engineering
- Over 1 million users, over 3500 universities and colleges
- Engineering in industry + biotech, medical, financial
- Toolboxes for different fields –
  - Engineering, Bioinformatics, Economics etc.
- Parallel computing support
  - Job execution on multicore/cluster systems
  - MPI support

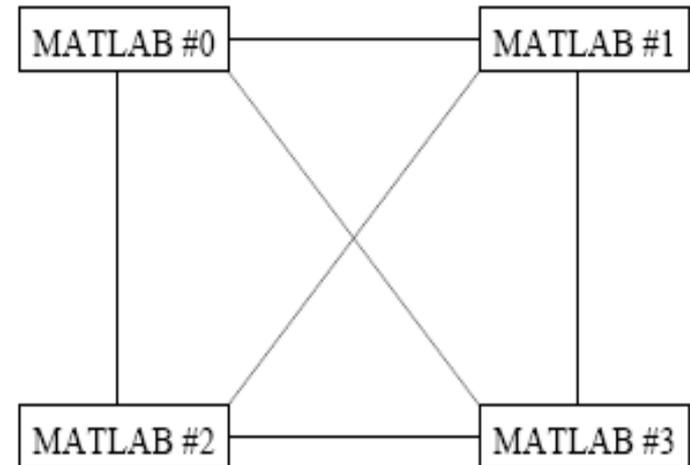
# Matlab in Parallel

Beautifully parallel



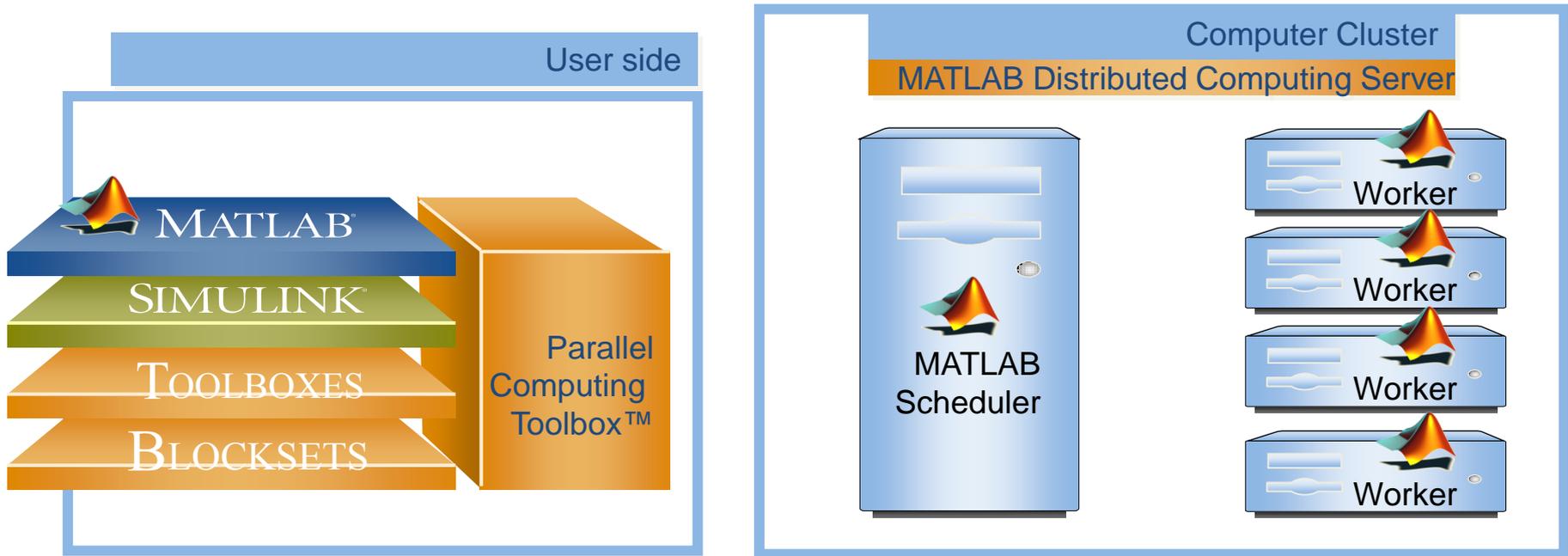
e.g., Multi, paralyze, Plab, ParMatlab

Message Passing



e.g., MultiMatlab, CMTM,  
DPTtoolbox, MatlabMPI, pMatlab

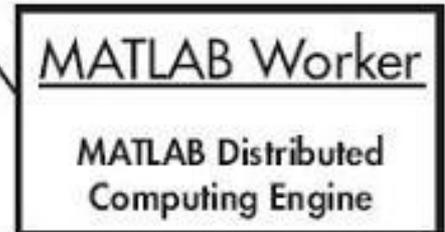
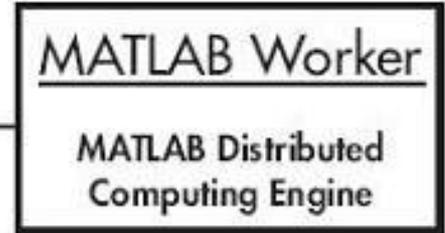
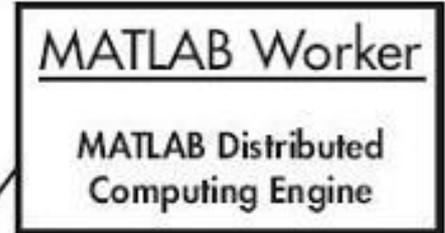
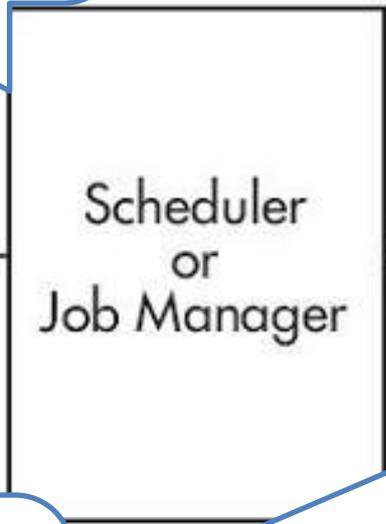
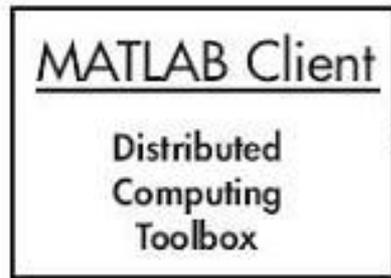
# Parallel Computing With MATLAB



- Support for third party schedulers

# Architecture of PCT

Coordinates the execution of jobs and the evaluation of their tasks, distributes the tasks for evaluation to the individual Matlab sessions called workers



Normal Matlab session in which a job and its tasks are created using the functions provided. Often, it is on the machine where user programs Matlab.

Matlab sessions which evaluates the task distributed by scheduler

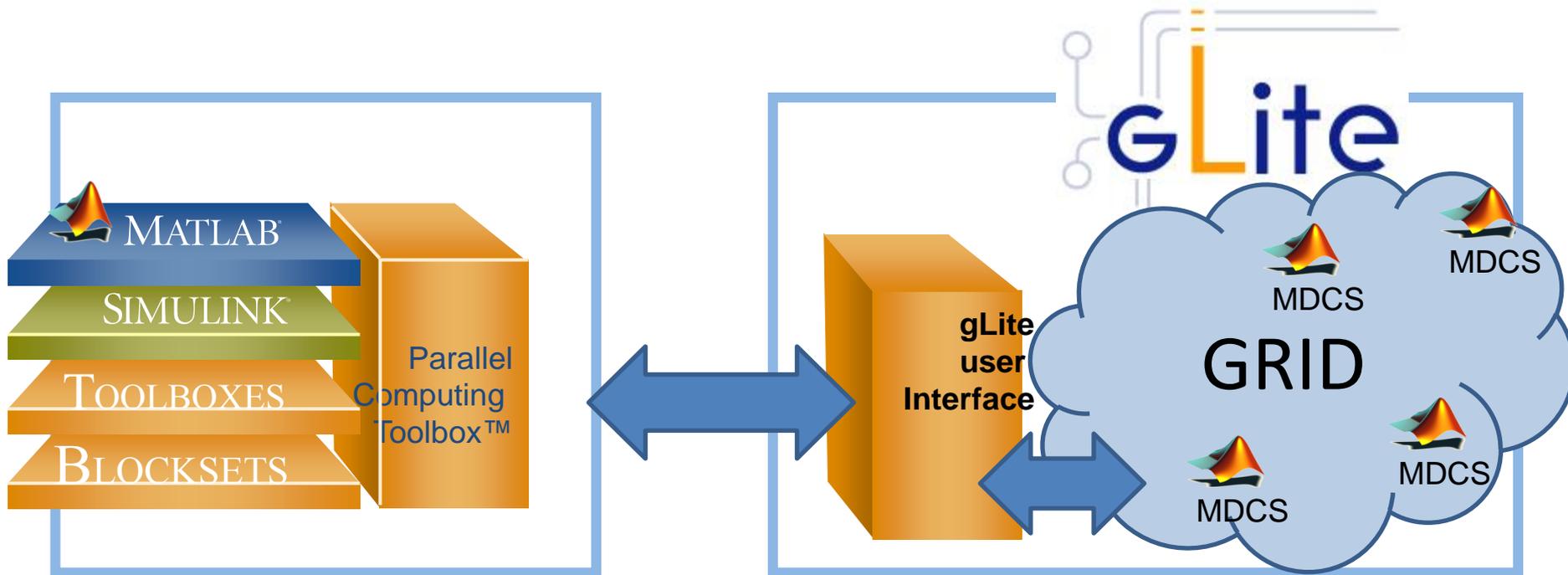
# Distributed Computing Server (DCS)



- Parallel Computing Toolbox
  - Only four local workers on a multicore or multiprocessor computer
- PCT + DCS -> Cluster-based applications
- Coordinate and execute independent MATLAB operations simultaneously on a cluster of computers

# MATLAB gLite integration

- API for generic scheduler

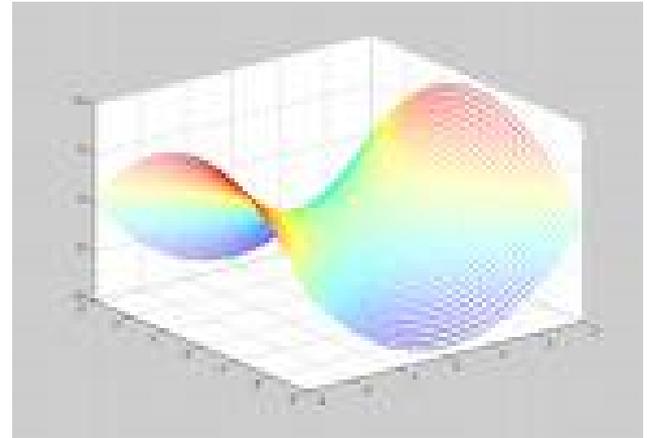


Developed by EGEE & Mathworks

# Parallel Computing Toolbox (PCT)

- Data and task parallelism using

- Parallel-for loops
- Distributed arrays
- Parallel numerical algorithms
- Message Passing functions



- Easy transition between serial and parallel

<http://www.mathworks.com/products/parallel-computing/description1.html>

# Mathworks – Parallel Computing toolbox

- The toolbox provides eight workers (MATLAB computational engines) to execute applications locally on a multicore desktop
- Parallel for-loops (`parfor`) for running task-parallel algorithms on multiple processors
- Computer cluster and grid support (with MATLAB Distributed Computing Server)

# Vectorization

- `>> clear all;`

```
tic;
```

```
for i=1:50000
```

```
  a(i) = sin(i);
```

```
end
```

```
toc
```

- Elapsed time is  
3.211070 seconds.

```
>> clear all;
```

```
tic;
```

```
i = [1:50000];
```

```
a = sin(i);
```

```
toc
```

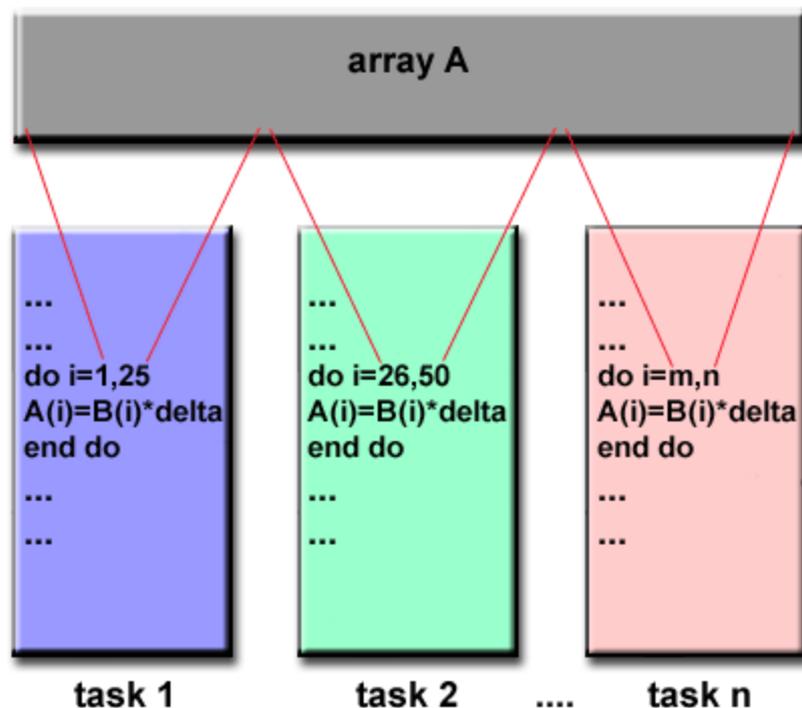
```
Elapsed time is 0.016062  
seconds.
```

```
>> speedup=3.211070 /0.016062  
=199.9172
```

# Data Parallel

The data parallel model demonstrates the following characteristics:

- Most of the parallel work performs operations on a data set, organized into a common structure, such as an array
- A set of tasks works collectively on the same data structure, with each task working on a different partition
- Tasks perform the same operation on their partition



On shared memory architectures, all tasks may have access to the data structure through global memory. On distributed memory architectures the data structure is split up and resides as "chunks" in the local memory of each task.

# parfor - Parallel for loop

## parfor - Parallel for loop

### Syntax

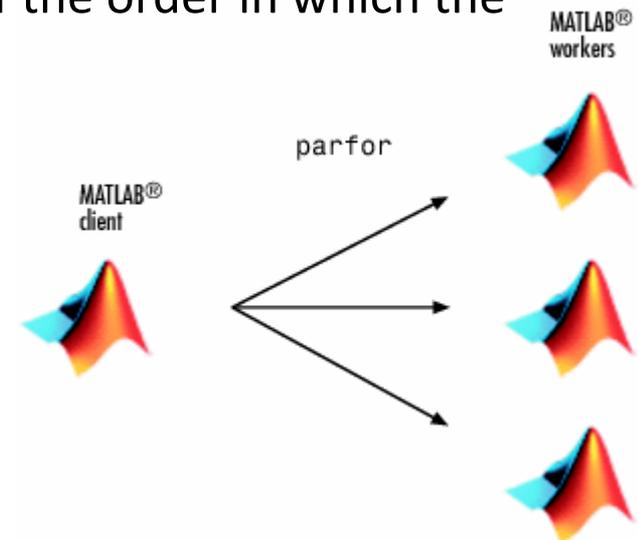
```
parfor loopvar = initval:endval; statements; end  
parfor (loopvar = initval:endval, M); statements; end
```

### Description

`parfor loopvar = initval:endval; statements; end` executes a series of MATLAB commands denoted here as statements for values of `loopvar` between `initval` and `endval`, inclusive, which specify a vector of increasing integer values. Unlike a traditional for-loop, there is no guarantee of the order in which the loop iterations are executed.

The general format of a `parfor` statement is:

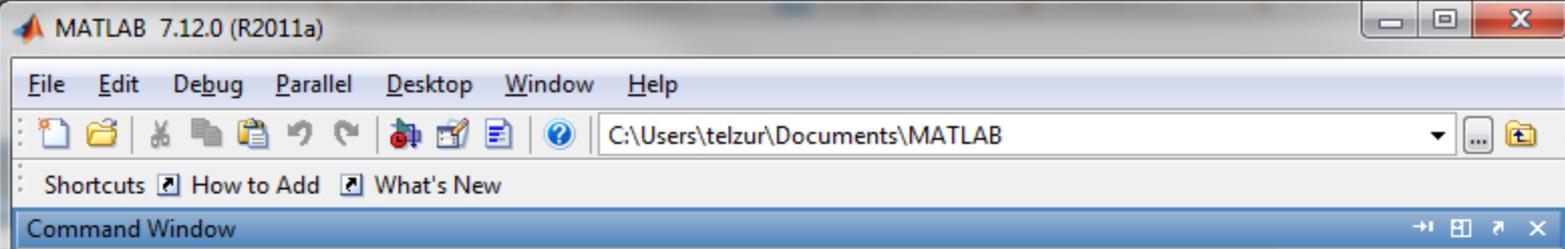
```
parfor Loopvar = initval:endval  
    <statements>  
end
```



# parfor – Perform three large eigenvalue computations using four computers or cores

```
>> clear all;
>> ntasks=4;tic;for
i=1:ntasks
c(:,i)=eig(rand(1000));
end;toc
Elapsed time is
8.575804 seconds.
```

- >> clear all;
- matlabpool open
- ntasks=4;tic;parfor  
i=1:ntasks
- c(:,i)=eig(rand(1000));
- end;toc
- Starting matlabpool using the 'local' configuration ... connected to 4 labs.
- Elapsed time is 5.198244 seconds.
- **Speedup= 1.6522**



```
>> clear all;  
>> ntasks=4;tic;parfor i=1:ntasks  
c(:,i)=eig(rand(2000));  
end;toc  
Elapsed time is 42.475697 seconds.  
>> clear all;  
>> matlabpool open  
Starting matlabpool using the 'local' configuration ...  
connected to 4 labs.  
>> ntasks=4;tic;parfor i=1:ntasks  
c(:,i)=eig(rand(2000));  
end;toc  
Elapsed time is 26.891233 seconds.  
>> matlabpool close  
speedup=1.5794  
Sending a stop signal to all the labs ... stopped.
```

# Parallel mode-I: matlabpool

- Open or close a pool of MATLAB sessions for parallel computation
- Syntax:
  - MATLABPOOL
  - MATLABPOOL OPEN
  - MATLABPOOL OPEN <poolsize>
  - MATLABPOOL CLOSE
  - MATLABPOOL CLOSE FORCE
  - .....
- Work on local client PC
- Without open matlabpool, parallel code will still run but runs sequentially

- %% Parameter Sweep of ODEs
- % This is a parameter sweep study of a 2nd order ODE system.
- %
- %  $m\ddot{x} + b\dot{x} + kx = 0$
- %
- % We solve the ODE for a time span of 0 to 25 seconds, with initial
- % conditions  $x(0) = 0$  and  $\dot{x}(0) = 1$ . We sweep the parameters  $b$
- % and  $k$  and record the peak values of  $x$  for each condition. At the end,
- % we plot a surface of the results.
- Computing in serial...
- Elapsed time is 27.64 seconds.
- Computing in parallel...
- Starting matlabpool using the 'local' configuration ... connected to 4 labs.
- Elapsed time is 12.39 seconds.
- Sending a stop signal to all the labs ... stopped.
- Speed up (time serial / time parallel): 2.23

```
%% Parameter Sweep (Parallel)
```

```
% Next, we convert the |for| loop to a |parfor| loop and start a pool or
```

```
% MATLAB workers.
```

```
disp('Computing in parallel...');drawnow;
```

```
matlabpool open
```

```
tic;
```

```
parfor idx = 1:numel(kGrid)
```

```
    % Solve ODE
```

```
    [T,Y] = ode45(@(t,y) odesystem(t, y, m, bGrid(idx), kGrid(idx)), ...
```

```
        [0, 25], ... % simulate for 25 seconds
```

```
        [0, 1]) ;    % initial conditions
```

```
    % Determine peak value
```

```
    peakVals(idx) = max(Y(:,1));
```

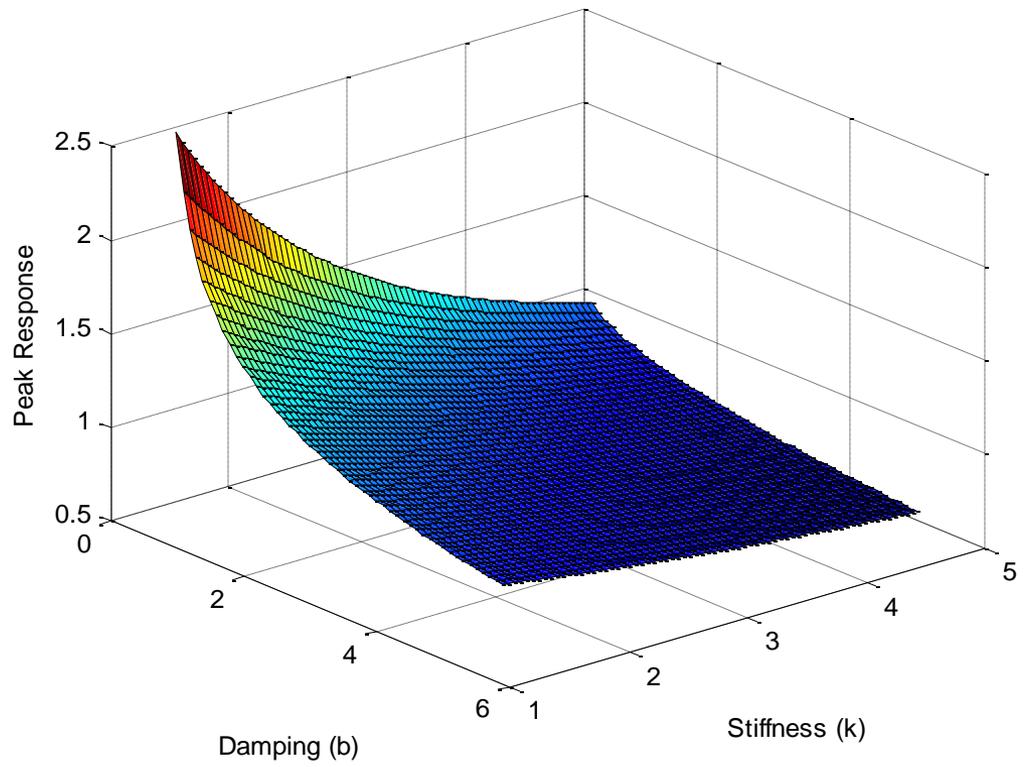
```
end
```

```
t2 = toc;
```

```
fprintf('Elapsed time is %0.2f seconds.\n', t2);
```

```
% Close MATLAB Pool
```

```
matlabpool close
```



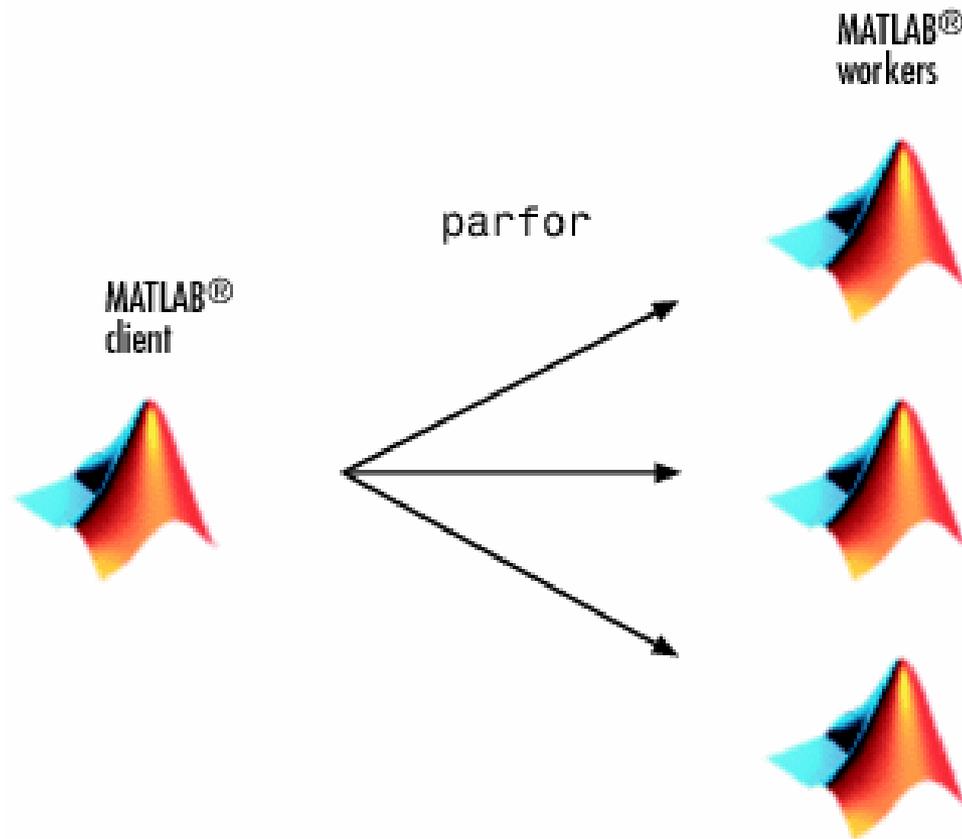
# Task Parallel applications

- **parallel problems by organizing them into independent *tasks* (units of work)**
  - parallelize Monte Carlo simulations
- **Parallel for-Loops (parfor)**

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

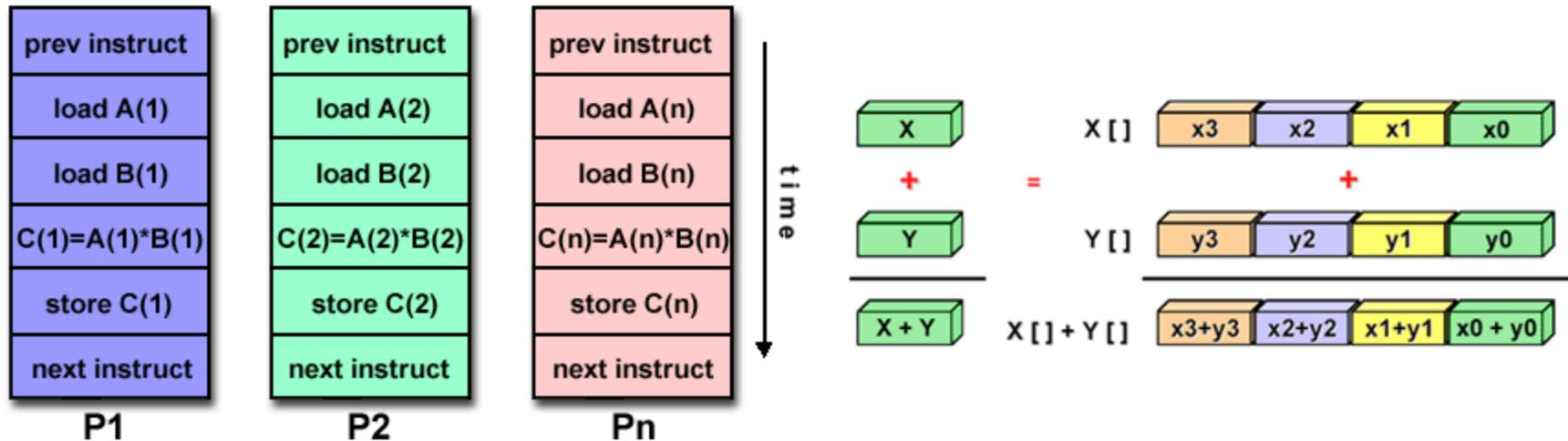
- Mix task parallel and serial code in the same function
- Run loops on a pool of MATLAB resources
- Iterations must be order-independent

Iterations run in parallel in the MATLAB pool  
(local workers)



# SIMD

- A type of parallel computer
- All processing units execute the same instruction at any given clock cycle
- Each processing unit can operate on a different data element
- Two varieties: Processor Arrays and Vector Pipelines
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.



# Data Parallel applications

- Single Program Multiple Data (spmd)

```
spmd (n)
    <statements>
end
```

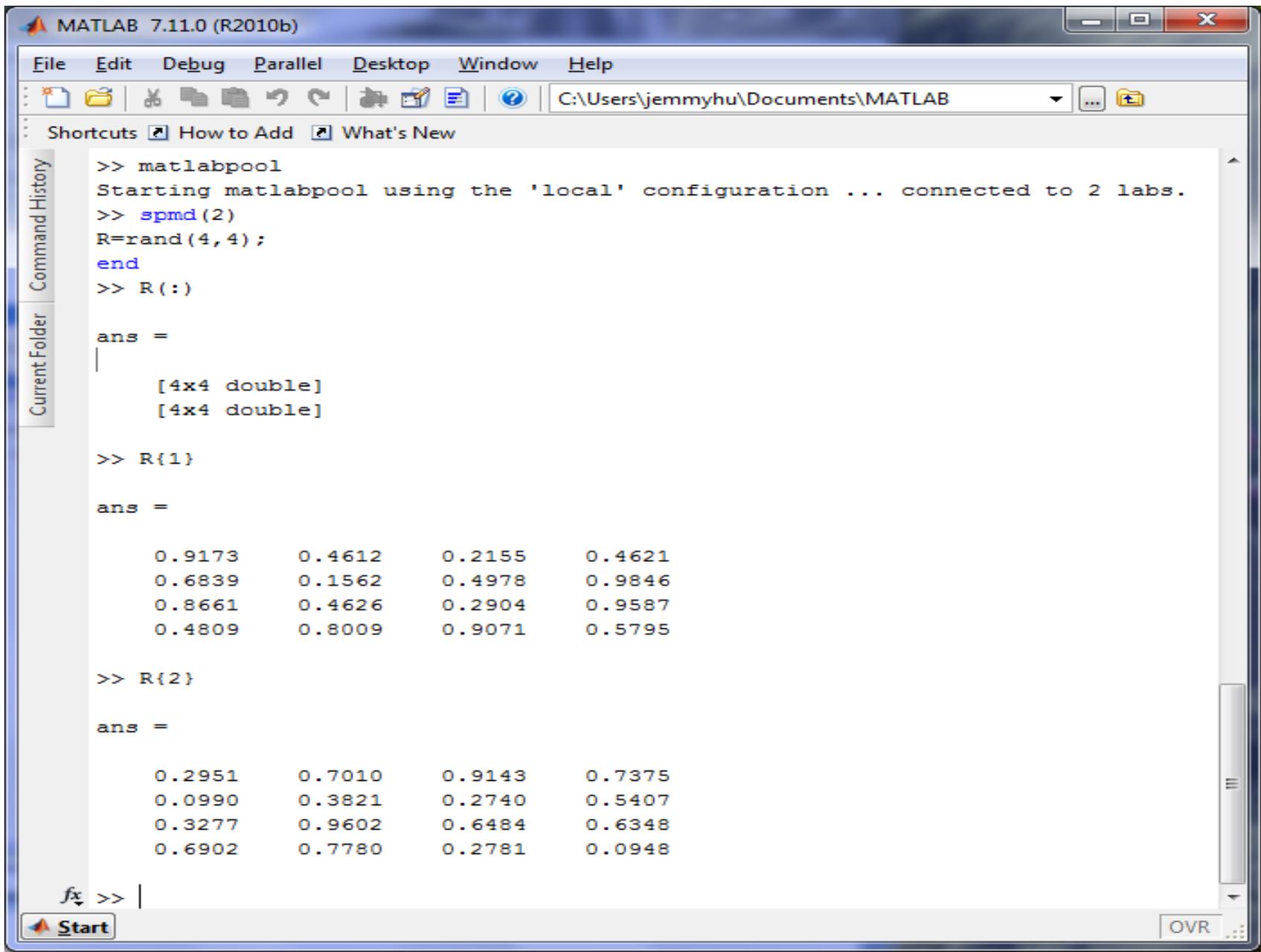
For example, create a random matrix on four labs:

```
matlabpool open
    spmd (2)
        R = rand(4,4);
    end
matlabpool close
```

create different sized arrays depending on labindex:

```
matlabpool open
    spmd (2)
        if labindex==1
            R = rand(4,4);
        else
            R = rand(2,2);
        end
    end
matlabpool close
```

# Demo



The image shows a screenshot of the MATLAB 7.11.0 (R2010b) Command Window. The window title is "MATLAB 7.11.0 (R2010b)". The menu bar includes "File", "Edit", "Debug", "Parallel", "Desktop", "Window", and "Help". The current folder is "C:\Users\jemmyhu\Documents\MATLAB". The Command History pane on the left shows the following commands: `>> matlabpool`, `>> spmd(2)`, `R=rand(4,4);`, `end`, `>> R(:)`, `>> R{1}`, and `>> R{2}`. The Command Window displays the output of these commands. The output of `matlabpool` is "Starting matlabpool using the 'local' configuration ... connected to 2 labs." The output of `spmd(2)` is "R=rand(4,4);". The output of `end` is "ans =". The output of `>> R(:)` is a 4x4 matrix of random numbers. The output of `>> R{1}` is a 4x4 matrix of random numbers. The output of `>> R{2}` is a 4x4 matrix of random numbers. The Command Window prompt is `fx >> |`.

```
MATLAB 7.11.0 (R2010b)
File Edit Debug Parallel Desktop Window Help
C:\Users\jemmyhu\Documents\MATLAB
Shortcuts How to Add What's New
Command History
Current Folder
>> matlabpool
Starting matlabpool using the 'local' configuration ... connected to 2 labs.
>> spmd(2)
R=rand(4,4);
end
>> R(:)

ans =

    [4x4 double]
    [4x4 double]

>> R{1}

ans =

    0.9173    0.4612    0.2155    0.4621
    0.6839    0.1562    0.4978    0.9846
    0.8661    0.4626    0.2904    0.9587
    0.4809    0.8009    0.9071    0.5795

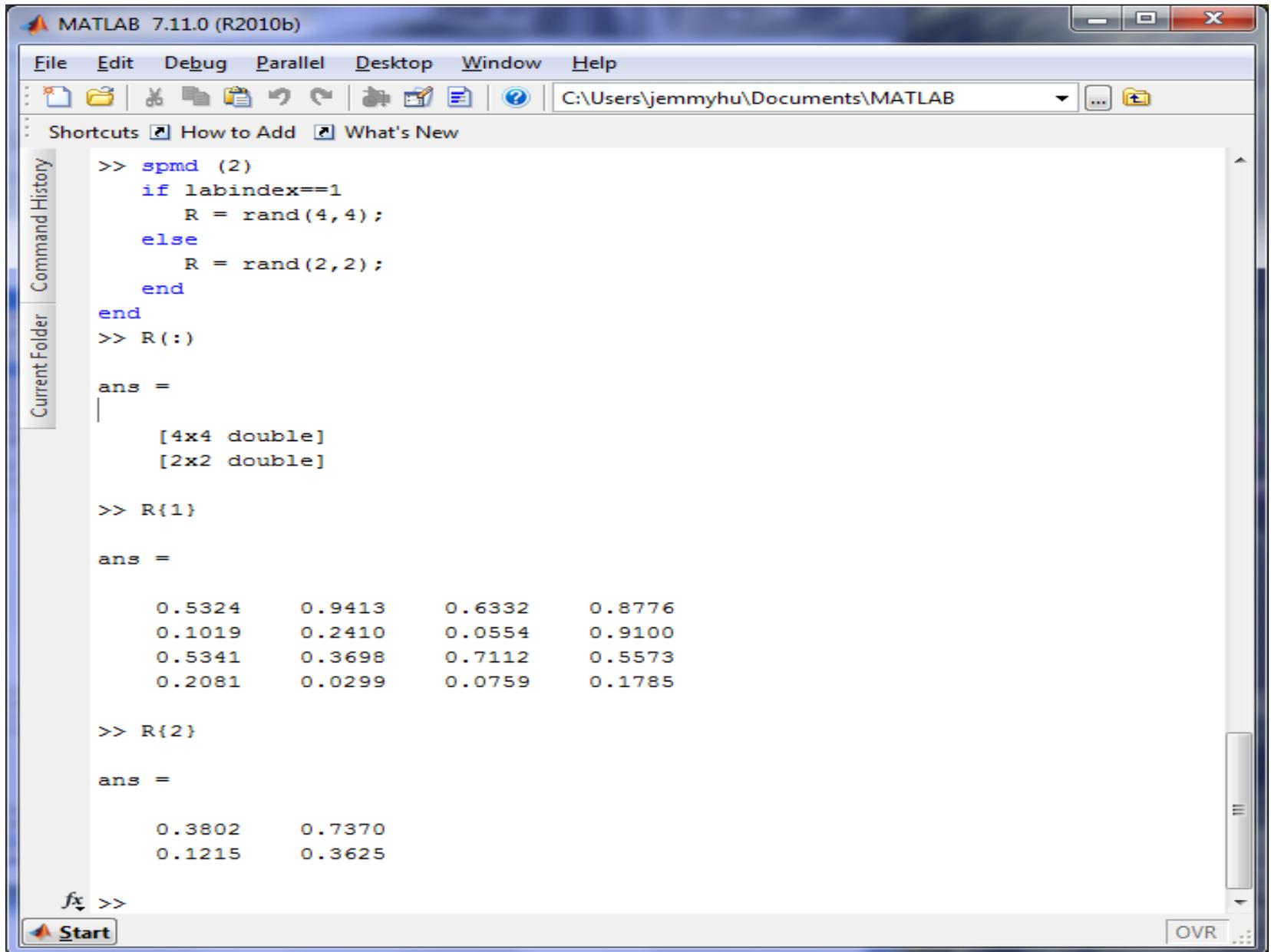
>> R{2}

ans =

    0.2951    0.7010    0.9143    0.7375
    0.0990    0.3821    0.2740    0.5407
    0.3277    0.9602    0.6484    0.6348
    0.6902    0.7780    0.2781    0.0948

fx >> |
Start OVR
```

# Demo



The image shows a screenshot of the MATLAB 7.11.0 (R2010b) Command Window. The window title is "MATLAB 7.11.0 (R2010b)". The menu bar includes "File", "Edit", "Debug", "Parallel", "Desktop", "Window", and "Help". The current directory is "C:\Users\jemmyhu\Documents\MATLAB". The Command Window contains the following code and output:

```
>> spmd (2)
    if labindex==1
        R = rand(4,4);
    else
        R = rand(2,2);
    end
end
>> R{:}
```

ans =

[4x4 double]
[2x2 double]

```
>> R{1}
```

ans =

0.5324	0.9413	0.6332	0.8776
0.1019	0.2410	0.0554	0.9100
0.5341	0.3698	0.7112	0.5573
0.2081	0.0299	0.0759	0.1785

```
>> R{2}
```

ans =

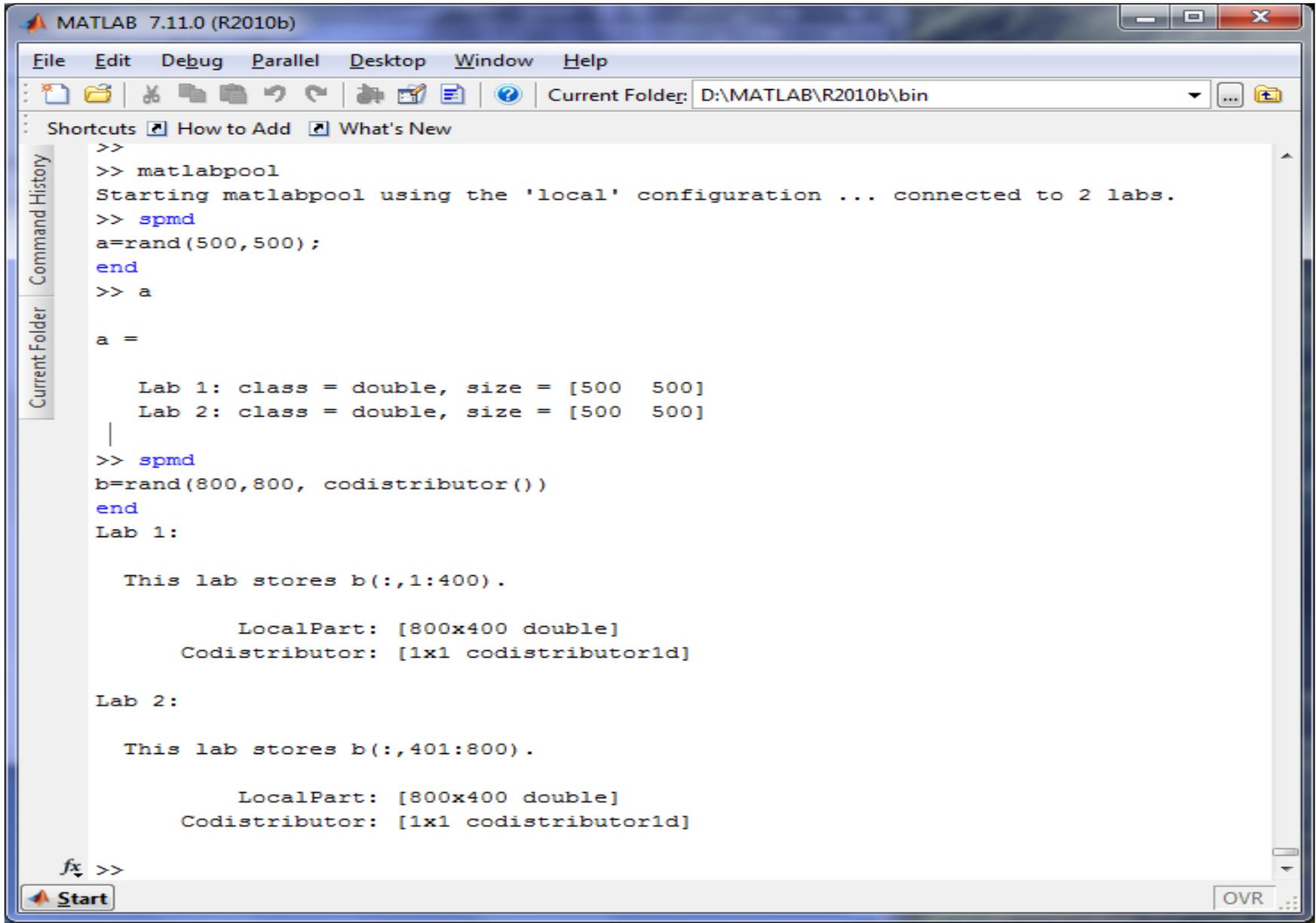
0.3802	0.7370
0.1215	0.3625

fx >>

Start OVR

# Distributed arrays and operations (matlabpool mode)

## codistributor()



```
MATLAB 7.11.0 (R2010b)
File Edit Debug Parallel Desktop Window Help
Current Folder: D:\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command History
>>
>> matlabpool
Starting matlabpool using the 'local' configuration ... connected to 2 labs.
>> spmd
a=rand(500,500);
end
>> a

a =

    Lab 1: class = double, size = [500 500]
    Lab 2: class = double, size = [500 500]

>> spmd
b=rand(800,800, codistributor())
end
Lab 1:

    This lab stores b(:,1:400).

        LocalPart: [800x400 double]
        Codistributor: [1x1 codistributor1d]

Lab 2:

    This lab stores b(:,401:800).

        LocalPart: [800x400 double]
        Codistributor: [1x1 codistributor1d]

fx >>
```

```
>> spmd
c=b'
end
Lab 1:

This lab stores c(1:400,:).

      LocalPart: [400x800 double]
      Codistributor: [1x1 codistributor1d]

Lab 2:

This lab stores c(401:800,:).

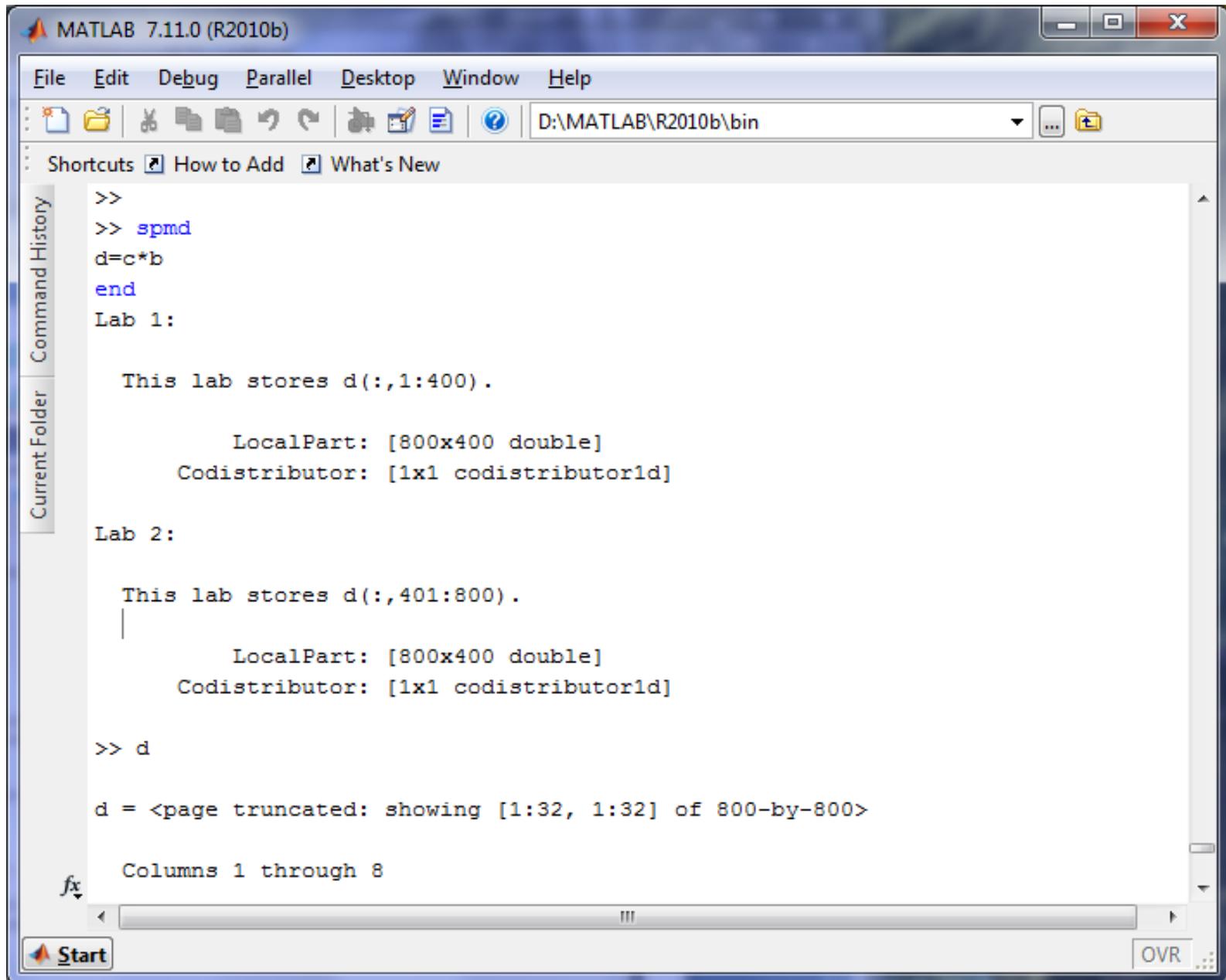
      LocalPart: [400x800 double]
      Codistributor: [1x1 codistributor1d]

>> c

c = <page truncated: showing [1:32, 1:32] of 800-by-800>

Columns 1 through 8

    0.2272    0.5522    0.9372    0.8589    0.8029    0.4314    0.2116    0.5545
    0.0658    0.7758    0.0901    0.3114    0.0746    0.5446    0.5271    0.7786
    0.0811    0.2854    0.2272    0.1215    0.3944    0.9134    0.6010    0.7143
    ...      ...      ...      ...      ...      ...      ...      ...
```



The image shows a screenshot of the MATLAB 7.11.0 (R2010b) Command Window. The window title is "MATLAB 7.11.0 (R2010b)". The menu bar includes File, Edit, Debug, Parallel, Desktop, Window, and Help. The address bar shows the current directory as "D:\MATLAB\R2010b\bin". The Command History pane on the left is active, showing the commands entered. The main Command Window area displays the following text:

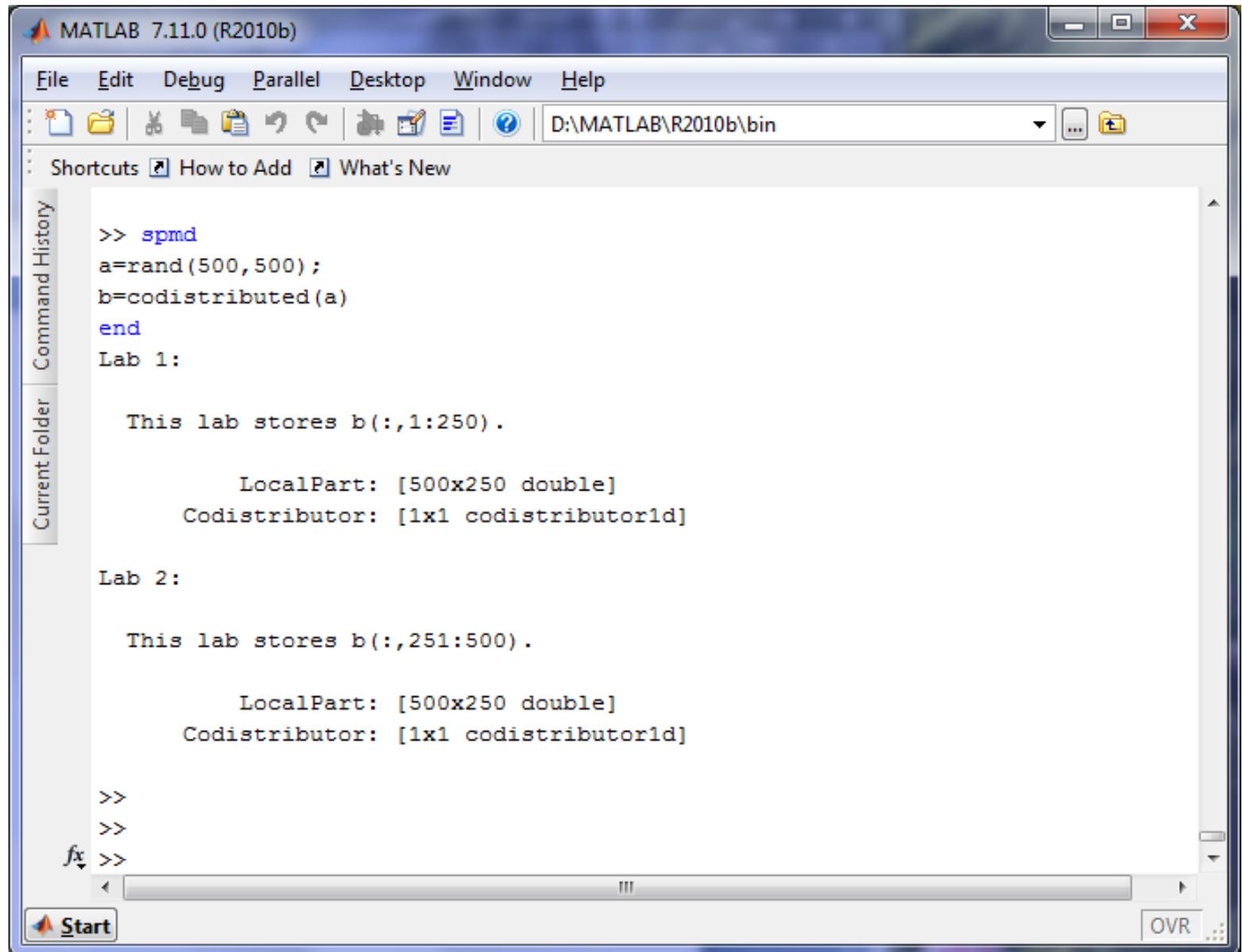
```
>>  
>> spmd  
d=c*b  
end  
Lab 1:  
  
This lab stores d(:,1:400).  
  
LocalPart: [800x400 double]  
Codistributor: [1x1 codistributor1d]  
  
Lab 2:  
  
This lab stores d(:,401:800).  
|  
LocalPart: [800x400 double]  
Codistributor: [1x1 codistributor1d]  
  
>> d  
  
d = <page truncated: showing [1:32, 1:32] of 800-by-800>  
  
Columns 1 through 8
```

The Command History pane on the left shows the following commands:

```
>>  
>> spmd  
d=c*b  
end  
Lab 1:  
  
This lab stores d(:,1:400).  
  
LocalPart: [800x400 double]  
Codistributor: [1x1 codistributor1d]  
  
Lab 2:  
  
This lab stores d(:,401:800).  
|  
LocalPart: [800x400 double]  
Codistributor: [1x1 codistributor1d]  
  
>> d
```

The Command Window also shows a scroll bar at the bottom and a "Start" button in the bottom-left corner. The "OVR" button is visible in the bottom-right corner.

# codistributed()



The image shows a MATLAB 7.11.0 (R2010b) Command Window. The window title is "MATLAB 7.11.0 (R2010b)". The menu bar includes "File", "Edit", "Debug", "Parallel", "Desktop", "Window", and "Help". The address bar shows the path "D:\MATLAB\R2010b\bin". The Command Window contains the following text:

```
>> spmd
a=rand(500,500);
b=codistributed(a)
end
Lab 1:

    This lab stores b(:,1:250).

        LocalPart: [500x250 double]
        Codistributor: [1x1 codistributor1d]

Lab 2:

    This lab stores b(:,251:500).

        LocalPart: [500x250 double]
        Codistributor: [1x1 codistributor1d]

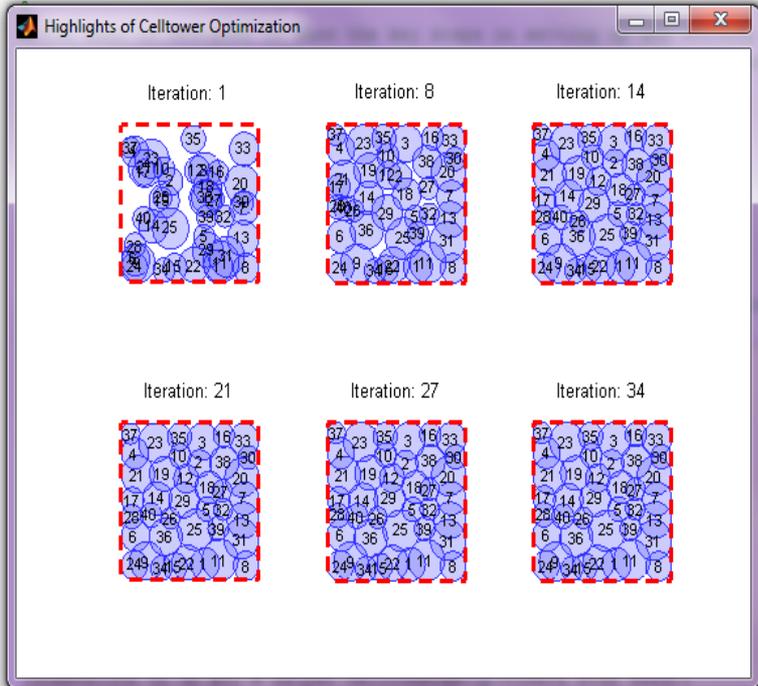
>>
>>
fx >>
```

The Command Window also shows a "Command History" sidebar on the left and a "Current Folder" sidebar on the right. The status bar at the bottom left shows the "Start" button and the "OVR" button.

```

Editor - C:\Users\elias\Downloads\ParallelComputingDemos\ParallelOptim_Example\PhoneTowerScript.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Base fx
- 1.0 + ÷ 1.1 x %%% %%%
1 %% Demo script for Cell Tower Optimization Problem
2 %% This demo illustrates how you would set up and solve an optimization
3 %% problem (constrained non-linear minimization).
4 %%
5 %% The problem is to determine the locations of # cell towers, each having
6 %% various coverage radius, so that there is maximum coverage, or minimum
7 %% overlap. It is a constrained optimization problem because the cell towers
8 %% are constrained to lie within a boundary. It is a non-linear optimization
9 %% problem because of the non-linear objective function (overlap area as a
10 %% function of # cell tower locations).
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
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```



MATLAB 7.11.0 (R2010b)

File Edit Debug Parallel Desktop Window Help

C:\Users\elias

Shortcuts How to Add What's New

Elapsed time: 7.21 sec  
 Elapsed time: 52.99 sec  
 Elapsed time is 67.55 seconds.  
 Starting matlabpool using the 'local' configuration ... connected to 4 labs.  
 Elapsed time is 44.09 seconds.  
 Speed up (time serial / time parallel): 1.53  
 Sending a stop signal to all the labs ... stopped.

fx >>

Optimization PlotFnc

File Edit View Insert Tools Desktop Window Help

Current Function Value: 67.0909

Function value

Iteration

Iteration: 33

Stop Pause

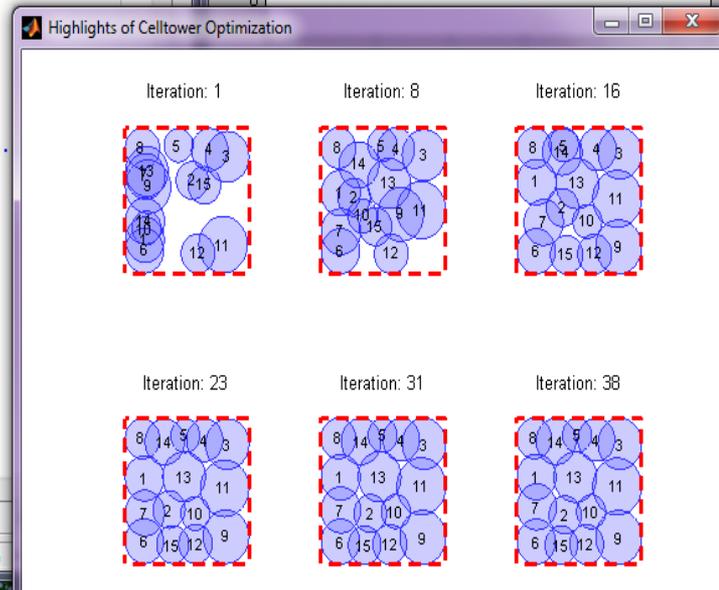
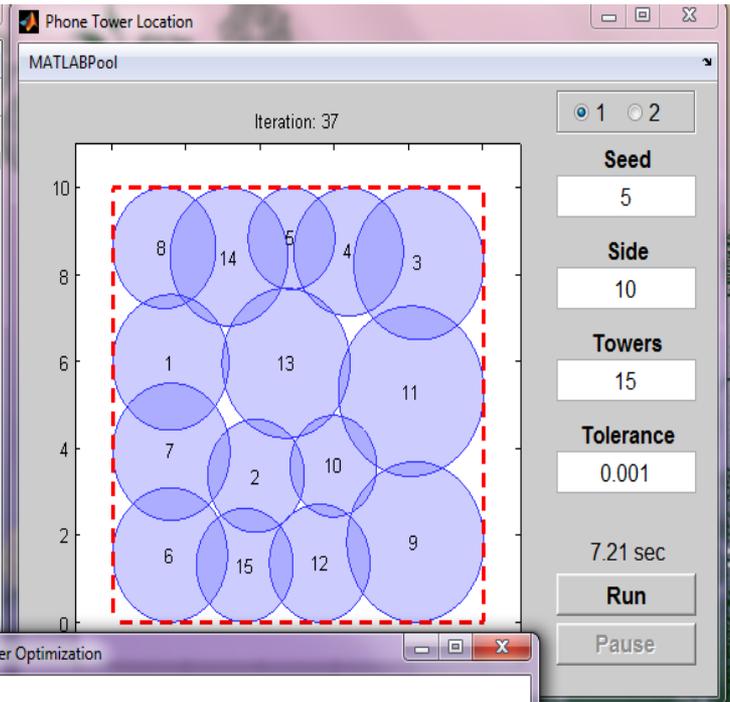
script Ln 30 Col 42 OVR

Start

```

Editor - C:\Users\elias\Downloads\ParallelComputingDemos\ParallelOptim_Example\celltowerGUI.m
File Edit Text Go Cell Tools Debug Desktop Window Help
- 1.0 + ÷ 1.1 x
1 function celltowerGUI()
2     % CELLTOWERGUI
3
4     % Copyright 2006-2011 The MathWorks, Inc.
5
6     % Default Parameter Values
7     default.towers = [15 40];
8     default.side   = [10 15];
9     default.seed   = [5 5];
10    default.tol    = [0.001 0.1];
11
12    % Initial Parameters
13    param.towers = default.towers(1);
14    param.side   = default.side(1);
15    param.seed   = default.seed(1);
16    param.tol    = default.tol(1);
17
18    % Generate Figure
19    figH = findobj('type', 'figure', 'tag', 'phonetowerGUI');
20    if ishandle(figH)
21        close(figH);
22    end
23    figH = figure(...
24        'name'                , 'Phone Tower Location',
25        'numbertitle'        , 'off', ...
26        'tag'                 , 'phonetowerGUI', ...
27        'color'               , [.8 .8 .8], ...
28        'menubar'             , 'none', ...
29        'defaultuicontrolunits', 'normalized', ...
30        'defaultuicontrolbackgroundcolor', [.8 .8 .8], ...
31        'defaultuicontrolfontunits', 'points', ...
32        'defaultuicontrolfontsize', 12);
33
34    menuH = uimenu(...
35        'Parent'   , figH, ...
36        'Label'    , 'MATLABPool');

```



- **Parallel mode on a MATLAB Pool (1)**

[matlabpool](#) Open or close pool of MATLAB sessions for parallel computation

[parfor](#) Execute code loop in parallel

[spmd](#) Execute code in parallel on MATLAB pool

- **Interactive Functions**

[help](#) Help for toolbox functions in Command Window

[pmode](#) Interactive Parallel Command Window

- **Parallel mode on a MATLAB Pool (2)**

[batch](#) Run MATLAB script as batch job

```
File Edit Text Go Cell Tools Debug Desktop
+ - 1.0 + + 1.1 x
This file uses Cell Mode. For information, see the rapid code
1 function [time1,time2,time3] = new
2 n=10^8; step = 1/n;
3 % Serial Version
4 tic; s=0;
5 for i=1:n-1
6 x=(i-0.5)*step; s=s+4./(1+x^2);
7 end
8 time1 = toc;
9
10 % Parallel Version
11 matlabpool open 4
12 tic;
13 spmd
14 slocal = myTestSum(n,step);
15 end
16 time2 = toc;
17 % Parallel Version parfor
18 nstep=10^8; step = 1/nstep;
19 tic;s=0;
20 parfor i=1:nstep-1
21 x=(i-0.5)*step;
22 s=s+4./(1+x^2);
23 end
24 time3 = toc;matlabpool close
25 end
26 %%
27 function slocal = myTestSum(n,step
28 nlo = ( n * ( labindex - 1 ) ) / n
29 nhi = ( n * labindex) / numlabs;
30 slocal = 0;
31 for i=nlo:nhi
32 x=(i-0.5)*step; slocal=slocal+4./
33 end
34 end
```

```
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\elias
Shortcuts How to Add What's New
>> [time1,time2,time3] = newSPMDVersion
Starting matlabpool using the 'local' configuration ... connected to 4 labs.
Sending a stop signal to all the labs ... stopped.

time1 =

    2.7594

time2 =

    1.0683

time3 =

    0.9888

A >> |
```

```

function [time1,time2,time3] =
newSPMDVersion
n=10^8; step = 1/n;
% Serial Version
tic; s=0;
for i=1:n-1
x=(i-0.5)*step; s=s+4./(1+x^2);
end
time1 = toc;

% Parallel Version
matlabpool open 4
tic;
spmd
slocal = myTestSum(n,step);
end
time2 = toc;

```

```

% Parallel Version parfor
nstep=10^8; step = 1/nstep;
tic;s=0;
parfor i=1:nstep-1
x=(i-0.5)*step;
s=s+4./(1+x^2);
end
time3 = toc;matlabpool close
end

function slocal = myTestSum(n,step)
nlo = ( n * ( labindex - 1 ) ) / numlabs + 1;
nhi = ( n * labindex ) / numlabs;
slocal = 0;
for i=nlo:nhi
x=(i-0.5)*step; slocal=slocal+4./(1+x^2);
end
end

```

# Profiling (profile on...profile viewer)

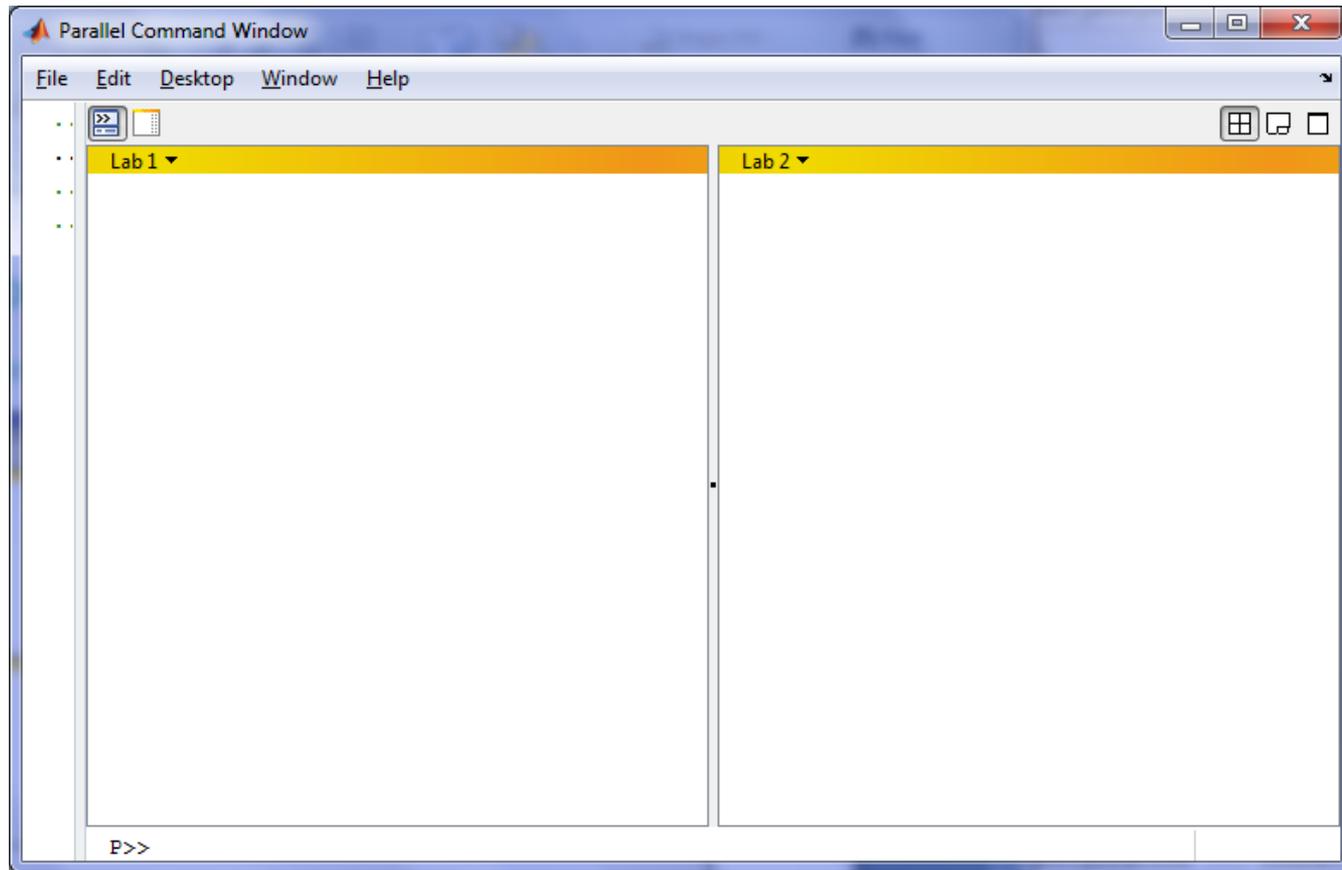
- profile on
- computations
- [profile viewer](#)

## Function listing

time	calls	line
		1 function [time1,time2,time3] = newSPMDVersion
	1	2 n=10^8; step = 1/n;
		3 % Serial Version
	1	4 tic; s=0;
< 0.01	1	5 for i=1:n-1
8.24	99999999	6 x=(i-0.5)*step; s=s+4./(1+x^2);
10.49	99999999	7 end
	1	8 time1 = toc;
		9
		10 % Parallel Version
6.08	1	11 matlabpool open 4
	1	12 tic;
1.07	1	13 spmd
		14 slocal = myTestSum(n,step);
		15 end
	1	16 time2 = toc;
		17 % Parallel Version parfor
	1	18 nstep=10^8; step = 1/nstep;
	1	19 tic;s=0;
1.04	1	20 parfor i=1:nstep-1
		21 x=(i-0.5)*step;
		22 s=s+4./(1+x^2);
		23 end
3.32	1	24 time3 = toc;matlabpool close
	1	25 end

# Parallel mode-II: pmode

>> pmode start



P>> pmode exit

# pmode demo

```
P>> help magic % ask for help on a function
```

```
P>> PI = pi % set a variable on all the labs
```

```
P>> myid = labindex % lab ID
```

```
P>> all = numlabs % total No. of labs
```

```
P>> segment = [1 2; 3 4; 5 6] % create a replicated array on all the labs
```

```
P>> segment = segment + 10*labindex % perform on different labs
```

```
    P>> x = magic(4) % replicated on every lab
```

```
P>> y=codistributed(x) % partitioned among the lab
```

```
P>> z = y + 10*labindex % operate on the distributed array whole
```

```
P>> combined = gather(y) % entire array in one workspace
```

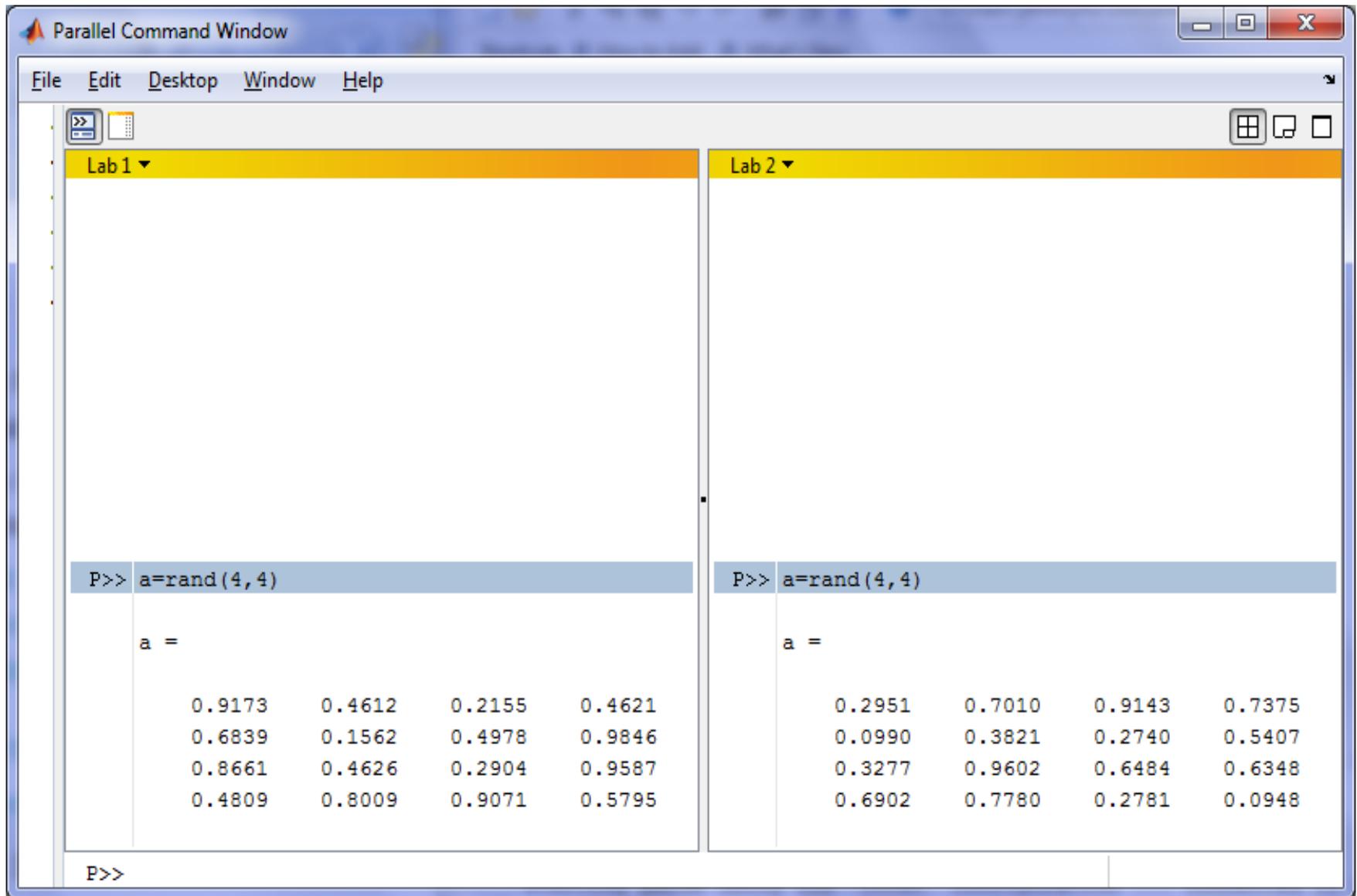
The combined is now a 4-by-4 array in the **client workspace**.

```
whos combined
```

To see the array, type its name.

```
combined
```

# Demo: distributed array operations (repeat)



The screenshot shows a 'Parallel Command Window' with two panes, 'Lab 1' and 'Lab 2'. Both panes show the execution of the command `a=rand(4,4)` and the resulting 4x4 matrix of random values.

**Lab 1**

```
P>> a=rand(4,4)
```

a =

0.9173	0.4612	0.2155	0.4621
0.6839	0.1562	0.4978	0.9846
0.8661	0.4626	0.2904	0.9587
0.4809	0.8009	0.9071	0.5795

P>>

**Lab 2**

```
P>> a=rand(4,4)
```

a =

0.2951	0.7010	0.9143	0.7375
0.0990	0.3821	0.2740	0.5407
0.3277	0.9602	0.6484	0.6348
0.6902	0.7780	0.2781	0.0948

Parallel Command Window

File Edit Desktop Window Help

Lab 1

```
P>> b=rand(2,8,codistributor())
```

This lab stores  $b(:,1:4)$ .

LocalPart: [2x4 double]  
Codistributor: [1x1 codistributor1d]

```
P>> getLocalPart(b)
```

ans =

0.4267	0.8379	0.6662	0.4703
0.5679	0.5489	0.7862	0.5058

Lab 2

```
P>> b=rand(2,8,codistributor())
```

This lab stores  $b(:,5:8)$ .

LocalPart: [2x4 double]  
Codistributor: [1x1 codistributor1d]

```
P>> getLocalPart(b)
```

ans =

0.9787	0.1247	0.0250	0.1717
0.4276	0.9818	0.6535	0.4670

P>>

Parallel Command Window

File Edit Desktop Window Help



Lab 1 ▾

P>> gather(b)

ans =

Columns 1 through 7

0.4267	0.8379	0.6662	0.4703	0.9787
0.5679	0.5489	0.7862	0.5058	0.4276

Column 8

0.1717  
0.4670

Lab 2 ▾

P>> gather(b)

ans =

Columns 1 through 7

0.4267	0.8379	0.6662	0.4703	0.9787
0.5679	0.5489	0.7862	0.5058	0.4276

Column 8

0.1717  
0.4670

P>>

Parallel Command Window

File Edit Desktop Window Help

Lab 1

0.4267	0.8379	0.6662	0.4703
0.5679	0.5489	0.7862	0.5058

P>> c=b'

This lab stores c(1:4,:).

LocalPart: [4x2 double]  
Codistributor: [1x1 codistributor1d]

P>> getLocalPart(c)

ans =

0.4267	0.5679
0.8379	0.5489
0.6662	0.7862
0.4703	0.5058

P>>

Lab 2

0.9787	0.1247	0.0250	0.1717
0.4276	0.9818	0.6535	0.4670

P>> c=b'

This lab stores c(5:8,:).

LocalPart: [4x2 double]  
Codistributor: [1x1 codistributor1d]

P>> getLocalPart(c)

ans =

0.9787	0.4276
0.1247	0.9818
0.0250	0.6535
0.1717	0.4670

P>>

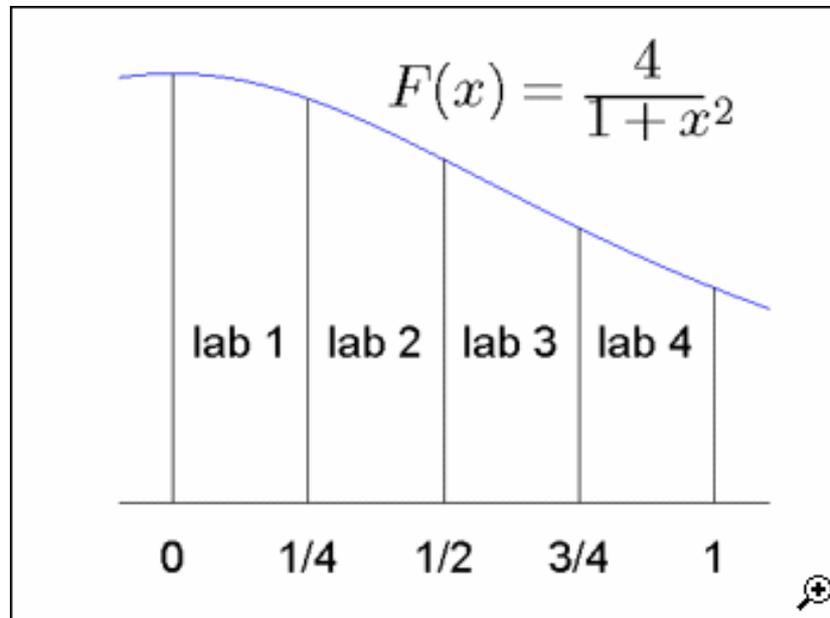
# Parallel pi in pmode

use the fact that

$$\int_0^1 \frac{4}{1+x^2} dx = 4(\operatorname{atan}(1) - \operatorname{atan}(0)) = \pi$$

to approximate pi by approximating the integral on the left.

divide the work between the labs by having each lab calculate the integral the function over a subinterval of  $[0, 1]$  as shown in the picture



# Steps

- All labs/workers will compute the same function:  $F=4/(1+x^2)$
- Each worker/lab will calculate over a subinterval  $[a,b]$  of  $[0, 1]$ ,  
for 2 labs, the subinterval will be:  
[0, 0.50]  
[0.50, 1.0]  
 $a = (\text{labindex}-1)/\text{numlabs}$   
 $b = \text{labindex}/\text{numlabs}$
- Use a MATLAB quadrature method to compute the integral  
 $\text{myIntegral} = \text{quadl}(F, a, b)$
- Add together to form the entire integral over  $[0,1]$   
 $\text{piApprox} = \text{gplus}(\text{myIntegral})$



Lab 1 ▾

```
P>> clear
P>> F=@(x) 4./(1+x.^2)
```

F =

$$@(x) 4./(1+x.^2)$$

```
P>> a=(labindex-1)/numlabs;
P>> b=labindex/numlabs;
P>> [a, b]
```

ans =

0 0.5000

```
P>> myIntegral=quadl(F,a,b)
```

myIntegral =

1.8546

```
P>> piApprox=gplus(myIntegral)
```

piApprox =

3.1416

```
P>> abs(pi-piApprox)
```

ans =

2.4865e-010

P&gt;&gt;



Lab 2 ▾

```
P>> clear
P>> F=@(x) 4./(1+x.^2)
```

F =

$$@(x) 4./(1+x.^2)$$

```
P>> a=(labindex-1)/numlabs;
P>> b=labindex/numlabs;
P>> [a, b]
```

ans =

0.5000 1.0000

```
P>> myIntegral=quadl(F,a,b)
```

myIntegral =

1.2870

```
P>> piApprox=gplus(myIntegral)
```

piApprox =

3.1416

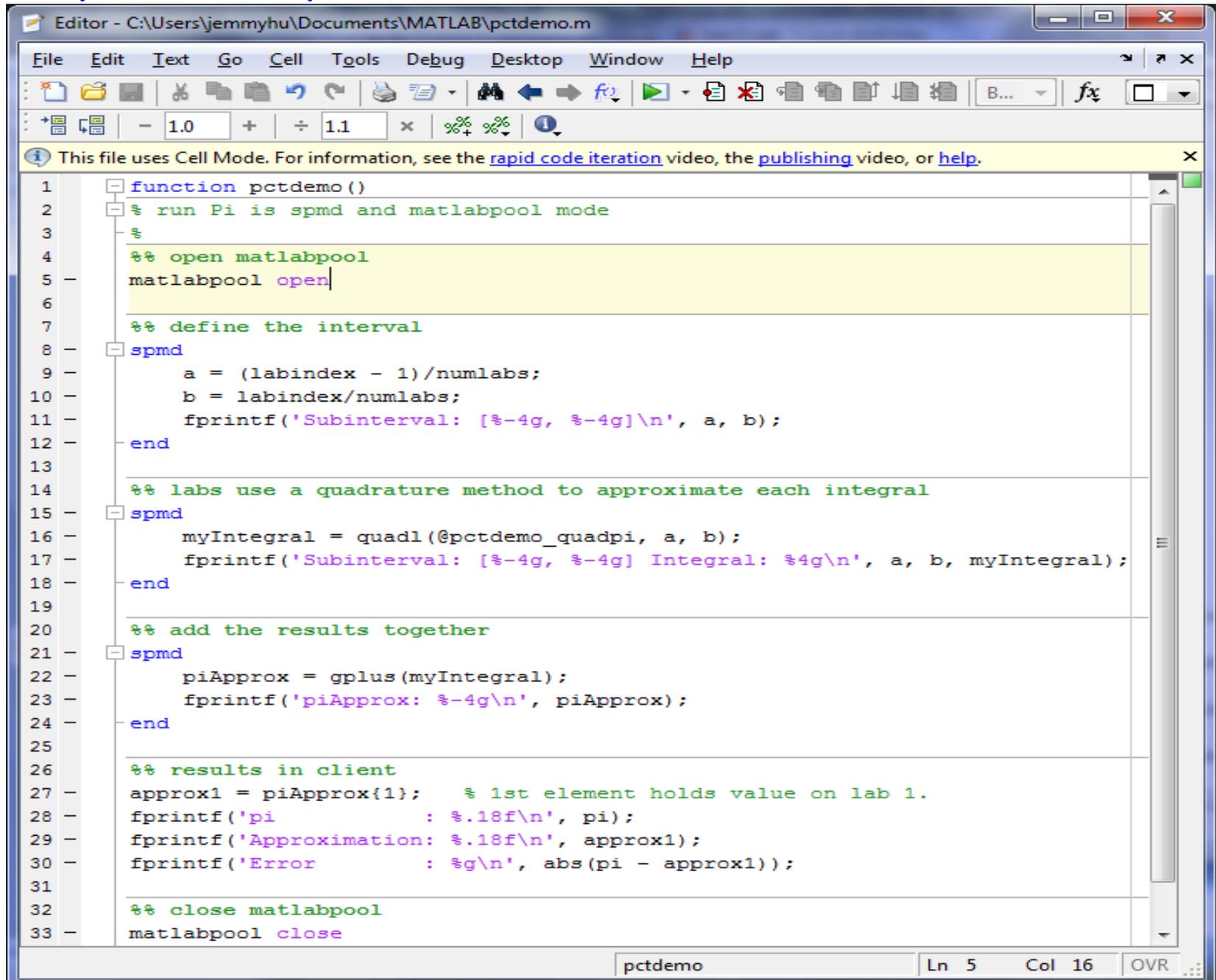
```
P>> abs(pi-piApprox)
```

ans =

2.4865e-010

P&gt;&gt;

# Parallel pi in matlabpool-mode



```
Editor - C:\Users\jemmyhu\Documents\MATLAB\pctdemo.m
File Edit Text Go Cell Tools Debug Desktop Window Help
This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.
1 function pctdemo()
2 % run Pi is spmd and matlabpool mode
3 %
4 %% open matlabpool
5 matlabpool open
6
7 %% define the interval
8 spmd
9     a = (labindex - 1)/numlabs;
10    b = labindex/numlabs;
11    fprintf('Subinterval: [%-4g, %-4g]\n', a, b);
12 end
13
14 %% labs use a quadrature method to approximate each integral
15 spmd
16    myIntegral = quadl(@pctdemo_quadpi, a, b);
17    fprintf('Subinterval: [%-4g, %-4g] Integral: %4g\n', a, b, myIntegral);
18 end
19
20 %% add the results together
21 spmd
22    piApprox = gplus(myIntegral);
23    fprintf('piApprox: %-4g\n', piApprox);
24 end
25
26 %% results in client
27 approx1 = piApprox{1}; % 1st element holds value on lab 1.
28 fprintf('pi          : %.18f\n', pi);
29 fprintf('Approximation: %.18f\n', approx1);
30 fprintf('Error         : %g\n', abs(pi - approx1));
31
32 %% close matlabpool
33 matlabpool close
pctdemo Ln 5 Col 16 OVR
```

```
>> pmode start
Starting pmode using the 'local' configuration ... connected to 2 labs.
>> pmode exit
Sending a stop signal to all the labs ... stopped.
>> run('C:\Users\jemmyhu\Documents\MATLAB\pctdemo.m')
Starting matlabpool using the 'local' configuration ... connected to 2 labs.
Lab 1:
  Subinterval: [0 , 0.5 ]
Lab 2:
  Subinterval: [0.5 , 1 ]
Lab 1:
  Subinterval: [0 , 0.5 ] Integral: 1.85459
Lab 2:
  Subinterval: [0.5 , 1 ] Integral: 1.287
Lab 1:
  piApprox: 3.14159
Lab 2:
  piApprox: 3.14159
pi          : 3.141592653589793100
Approximation: 3.141592653838447500
Error       : 2.48654e-010
Sending a stop signal to all the labs ... stopped.
```

fx &gt;&gt; |

# Batch mode

MATLAB®  
client

MATLAB®  
worker

batch



Name a .m file as 'mybatch' with

```
for i=1:1024  
A(i) = sin(i*2*pi/1024);  
end
```

run in batch mode

```
job = batch('mybatch')
```

The batch command does not block MATLAB, so you must wait for the job to finish before you can retrieve and view its results:

```
wait(job)
```

The load command transfers variables from the workspace of the worker to the workspace of the client, where you can view the

results:

```
load(job, 'A')  
plot(A)
```

When the job is complete, permanently remove its data:

```
destroy(job)
```

```
MATLAB 7.11.0 (R2010b)
File Edit Debug Parallel Desktop Window Help
C:\Users\jemmyhu\Documents\MATLAB
Shortcuts How to Add What's New
Command History
Current Folder
>>
>> job=batch('mybatch')

job =

Job ID 13 Information
=====

        UserName : jemmyhu
        State     : running
SubmitTime : Thu Jun 02 11:01:55 EDT 2011
StartTime  :
Running Duration :

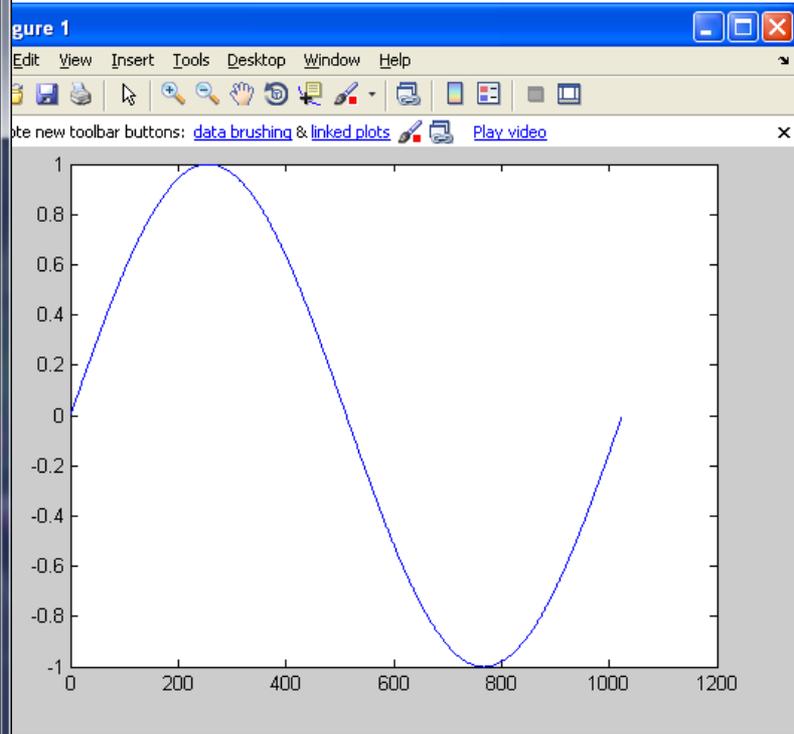
- Data Dependencies

FileDependencies : C:\Users\jemmyhu\Documents\MATLAB\mybatch.m
PathDependencies : {}

- Associated Task(s)

Number Pending : 1
Number Running : 0
Number Finished : 0
TaskID of errors :

>> wait(job)
>> load(job, 'A')
>> plot(A)
fx >> |
```



# A batch parallel loop

```
% mybatch
parfor i=1:1024
A(i) = sin(i*2*pi/1024);
end
```

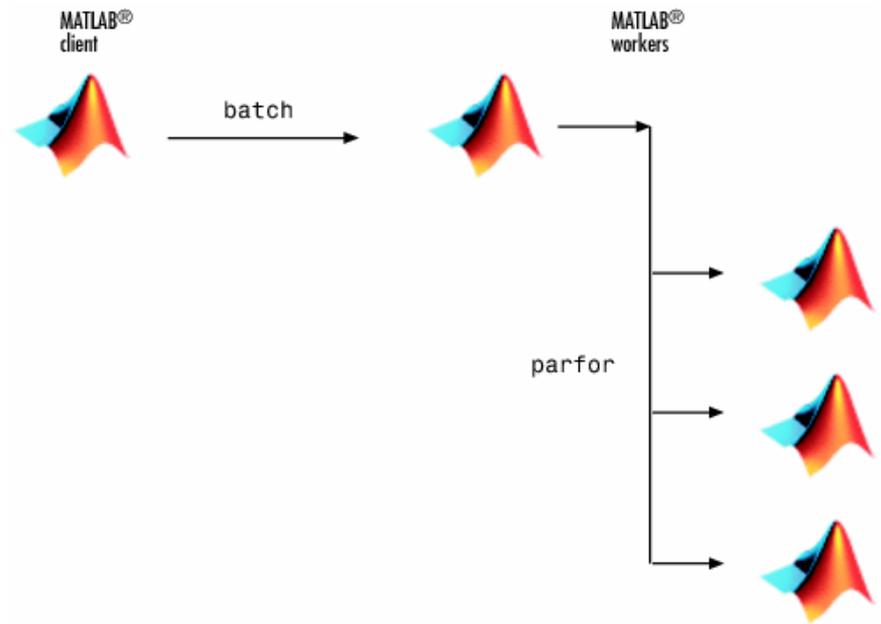
```
% run job in batch
job = batch('mybatch', 'configuration', 'local', 'matlabpool', 2)
```

% To view the results:

```
wait(job)
load(job, 'A')
plot(A)
```

% remove its data:

```
destroy(job)
```



MATLAB 7.11.0 (R2010b)

Editor - C:\Users\elias\Dropbox\S3\Mathima26.11.12\TALK\run\_pmybatch.m

```
File Edit Debug Parallel Desktop Window Help
C:\Users\elias

Shortcuts How to Add What's New

job =

Job ID 22 Information
=====
                UserName : elias
                  State : running
        SubmitTime : Fri Nov 23 07:23:52 EET 2012
          StartTime :
        Running Duration :

- Data Dependencies

        FileDependencies : C:\Users\elias\Dropbox\S3\Math
        PathDependencies : {}

- Associated Task(s)

        Number Pending : 1
        Number Running : 0
        Number Finished : 0
        TaskID of errors :

fx >>
```

```
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: ... fx

1 %run in batch mode
2 job = batch('pmybatch')
3 %The batch command does not block MATLAB, so you must wait
4 %for the job to finish before you can retrieve and view its results:
5 wait(job)
6 %The load command transfers variables from the workspace of the
7 %worker to the workspace of the client, where you can view the
8 %results:
9 load(job, 'A')
10 plot(A)
11 %When the job is complete, permanently remove its data:
12 destroy(job)
13
```

Figure 1

Col 7 OVR

```
>> job=batch('mybatch_par', 'configuration', 'local', 'matlabpool', 1)
```

```
job =
```

```
MatlabPool Job ID 17 Information
```

```
=====
                        UserName : jemmyhu
                          State   : queued
                    SubmitTime : Thu Jun 02 11:09:39 EDT 2011
                          StartTime :
                    Running Duration :
```

```
- Data Dependencies
```

```
FileDependencies : C:\Users\jemmyhu\Documents\MATLAB\mybatch_par.m
PathDependencies  : {}
```

```
- Associated Task(s)
```

```
Number Pending   : 2
Number Running   : 0
Number Finished  : 0
TaskID of errors :
```

```
- Scheduler Dependent (MatlabPool Job)
```

```
MaximumNumberOfWorkers : 2
MinimumNumberOfWorkers  : 2
```

```
>> wait(job)
>> load(job, 'A')
>> plot(A)
>> destroy(job)
```

```
fx >>
```

This file uses Cell Mode. For information, see the [rapid code iteration](#) video, the [publishing](#) video, or [help](#).

```

1 function interactiveToBatch()
2     % interactive to batch
3
4     %% for loop
5     A = cell(1,4);
6     for(i=1:4)
7         A{i} = magic(i);
8     end
9     A{:}
10
11    %% interactive parallel for loop
12    A = cell(1,4);
13    parfor(i=1:4)
14        A{i} = magic(i);
15    end
16    A{:}
17
18    %% batch job
19    jm = findResource('scheduler', 'configuration', 'local');
20    job = createJob(jm);
21    for(i=1:4)
22        createTask(job, @magic, 1, {i});
23    end
24    submit(job);
25    waitForState(job);
26    A = getAllOutputArguments(job);
27    A{:}
28    end
    
```

# Key Function List

- **Job Creation**

[createJob](#) Create job object in scheduler and client

[createTask](#) Create new task in job

[dfeval](#) Evaluate function using cluster

- **Interlab Communication Within a Parallel Job**

[labBarrier](#) Block execution until all labs reach this call

[labBroadcast](#) Send data to all labs or receive data sent to all labs

[labindex](#) Index of this lab

[labReceive](#) Receive data from another lab

[labSend](#) Send data to another lab

[numlabs](#) Total number of labs operating in parallel on current job

- **Job Management**

[cancel](#) Cancel job or task

[destroy](#) Remove job or task object from parent and memory

[getAllOutputArguments](#) Output arguments from evaluation of all tasks in job object

[submit](#) Queue job in scheduler

[wait](#) Wait for job to finish or change states

# Typical Use Cases

- **Parallel for-Loops (parfor)**

allowing several MATLAB workers to execute individual loop iterations simultaneously

restriction on parallel loops is that no iterations be allowed to depend on any other iterations.

- **Large Data Sets**

allows you to distribute that array among multiple MATLAB workers, so that each worker contains only a part of the array

Each worker operates only on its part of the array, and workers automatically transfer data between themselves when necessary

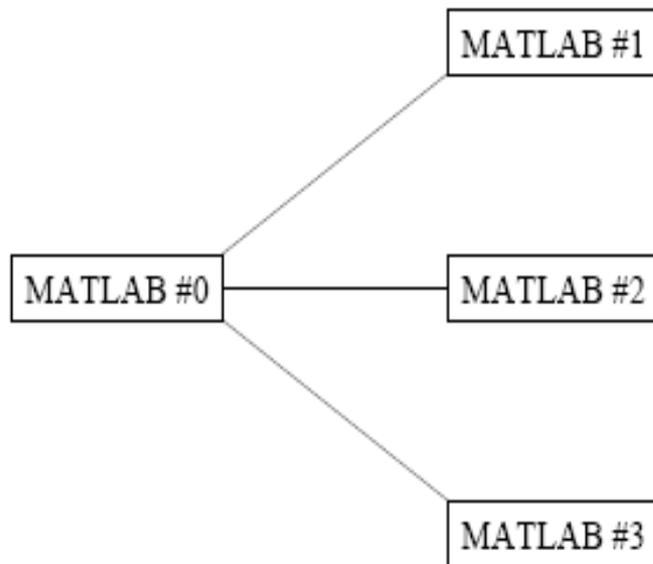
- **Batch Jobs**

offload work to a MATLAB worker session to run as a batch job.

the MATLAB worker can run either on the same machine as the client, or if using MATLAB Distributed Computing Server, on a remote cluster machine.

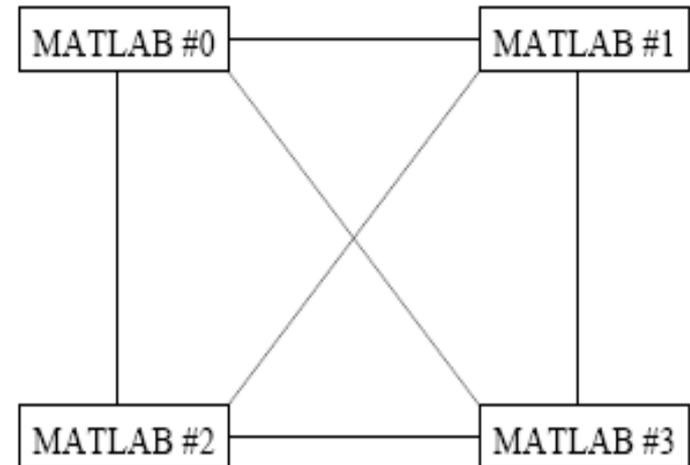
# Matlab in Parallel

Beautifully parallel



e.g., Multi, paralyze, Plab, ParMatlab

Message Passing



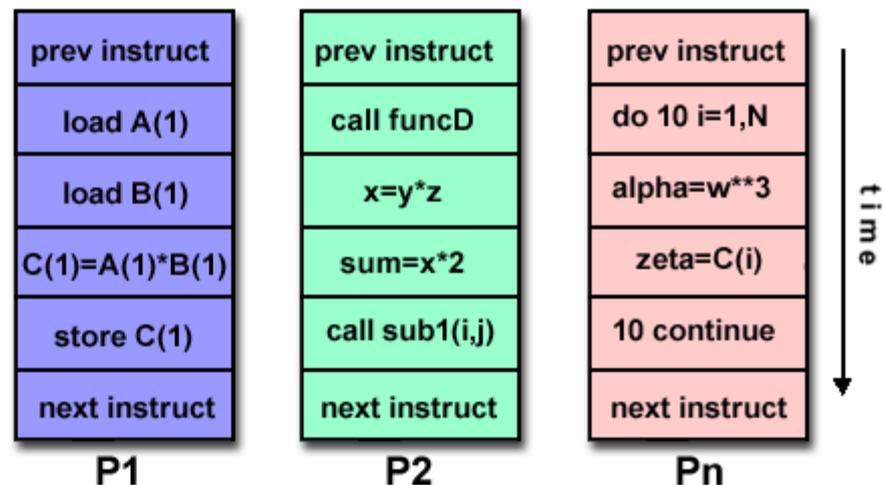
e.g., MultiMatlab, CMTM,  
DPTtoolbox, MatlabMPI, pMatlab

# MPI+SMP Parallel Paradigms

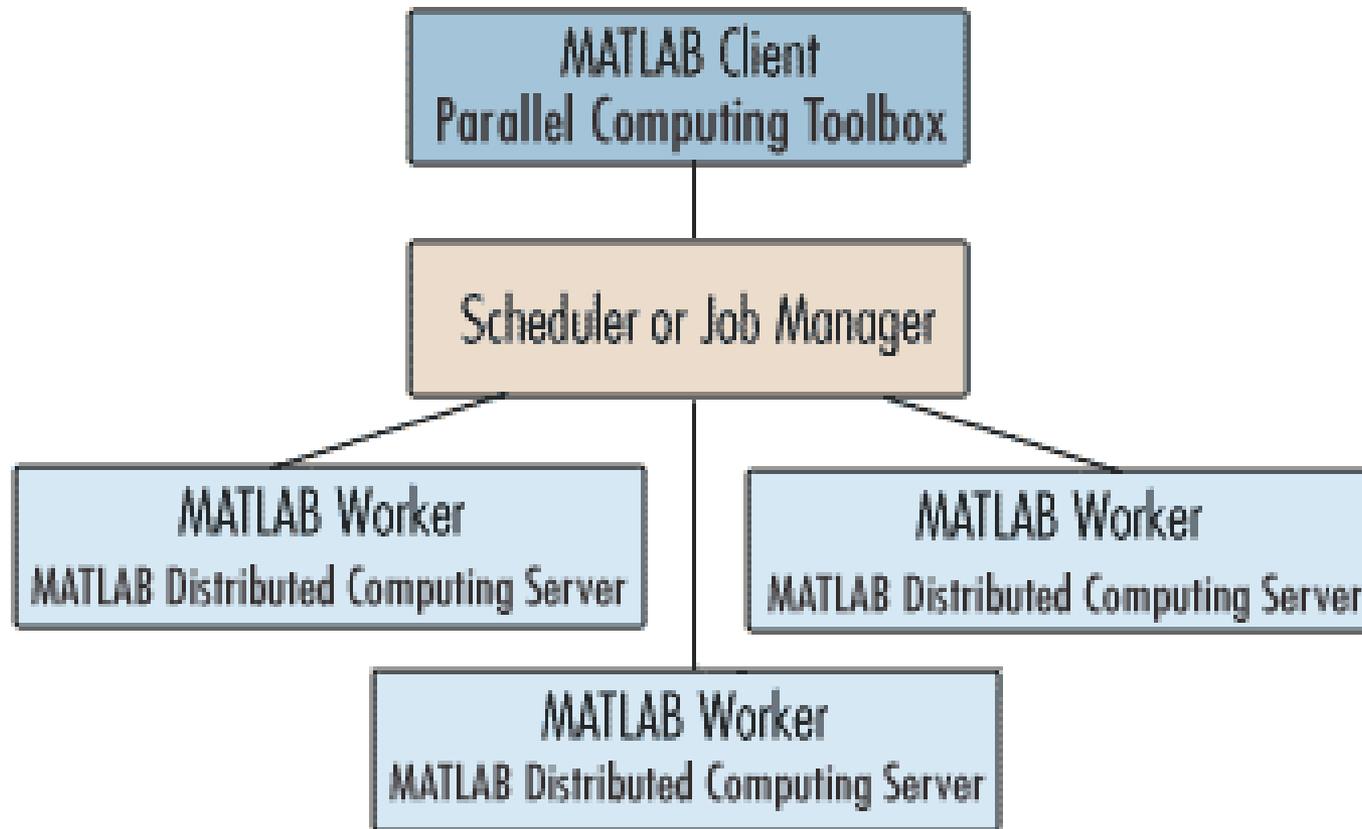
- <http://ist.uwaterloo.ca/ew/saw/parallel/FLASH/swf/mpi+smp.swf>

# MIMD

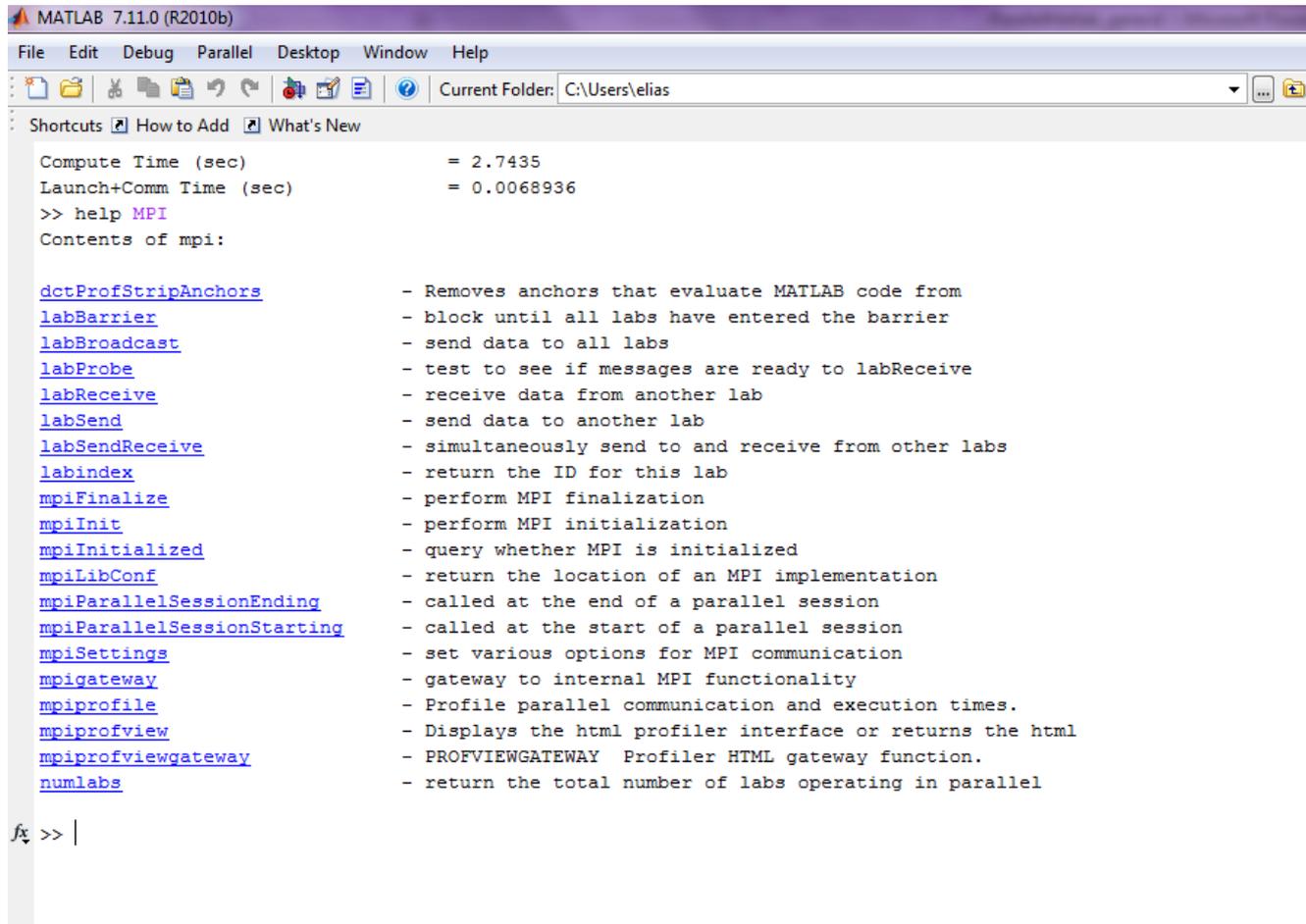
- Currently, most common type of parallel computer
- Every processor may be executing a different instruction stream
- Every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components



# Parallel Computing Toolbox and MATLAB Distributed Computing



# MPI Library

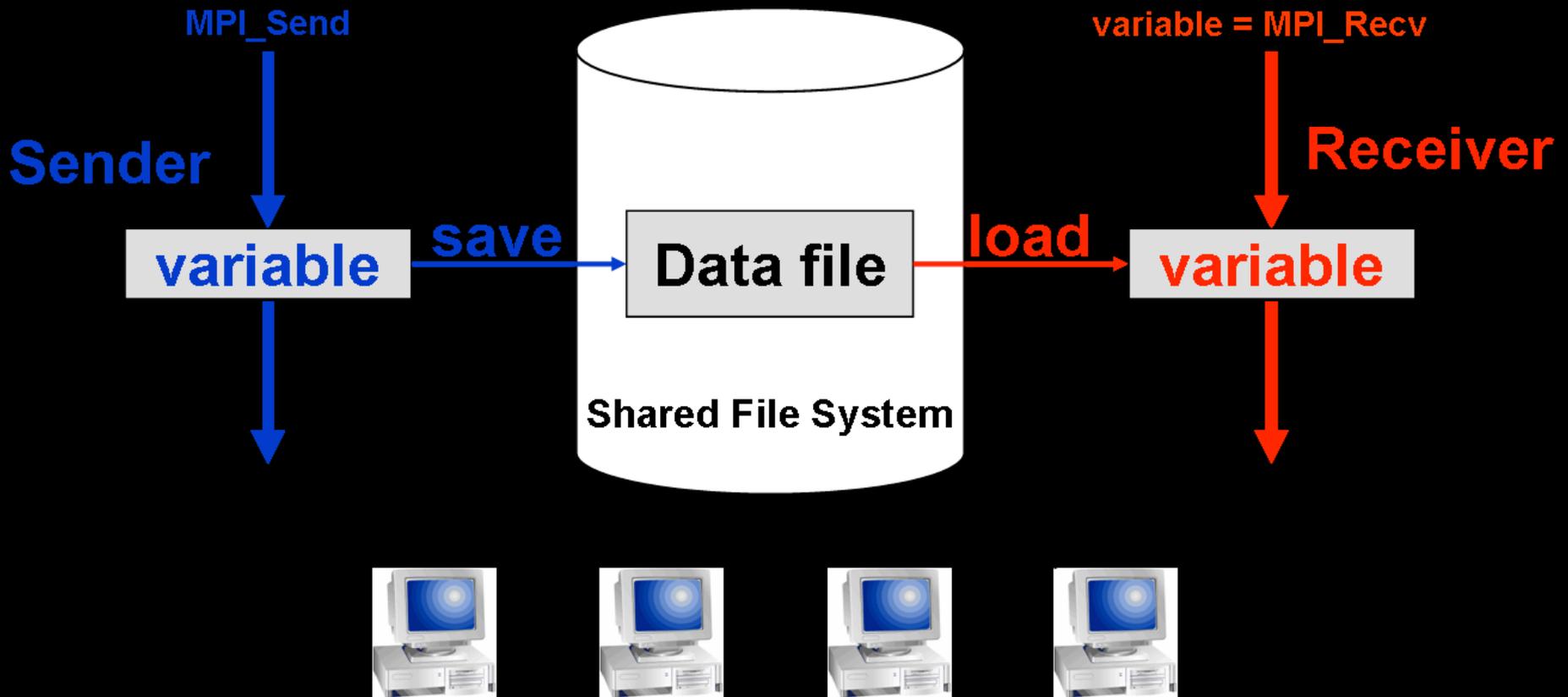


```
MATLAB 7.11.0 (R2010b)
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\elias
Shortcuts How to Add What's New
Compute Time (sec) = 2.7435
Launch+Comm Time (sec) = 0.0068936
>> help MPI
Contents of mpi:

dctProfStripAnchors - Removes anchors that evaluate MATLAB code from
labBarrier - block until all labs have entered the barrier
labBroadcast - send data to all labs
labProbe - test to see if messages are ready to labReceive
labReceive - receive data from another lab
labSend - send data to another lab
labSendReceive - simultaneously send to and receive from other labs
labindex - return the ID for this lab
mpiFinalize - perform MPI finalization
mpiInit - perform MPI initialization
mpiInitialized - query whether MPI is initialized
mpiLibConf - return the location of an MPI implementation
mpiParallelSessionEnding - called at the end of a parallel session
mpiParallelSessionStarting - called at the start of a parallel session
mpiSettings - set various options for MPI communication
mpigateway - gateway to internal MPI functionality
mpiprofile - Profile parallel communication and execution times.
mpiprofview - Displays the html profiler interface or returns the html
mpiprofviewgateway - PROFVIEWGATEWAY Profiler HTML gateway function.
numlabs - return the total number of labs operating in parallel

fx >> |
```

**MatlabMPI implements the fundamental communication operations in MPI using MATLAB's file I/O functions.**



# MatlabMPI

<http://www.ll.mit.edu/mission/isr/matlabmpi/matlabmpi.html#introduction>

Exit full screen (F11)

Home > Mission Areas > ISR Systems and Technology > MatlabMPI

## MATLABMPI

Space Control >

Air and Missile Defense  
Technology >

Communications and  
Information Technology >

ISR Systems and  
Technology >

- MatlabMPI
- pMatlab
- HPEC Challenge

Advanced Electronics  
Technology >

Tactical Systems >

Homeland Protection >

Air Traffic Control >

### Parallel Programming with MatlabMPI

*Dr. Jeremy Kepner*  
[kepner@ll.mit.edu](mailto:kepner@ll.mit.edu)

#### I. INTRODUCTION

Matlab is the dominant programming language for implementing numerical computations and is widely used for algorithm development, simulation, data reduction, testing and system evaluation. Many of these computations could benefit from faster execution on a parallel computer. There have been many previous attempts to provide an efficient mechanism for running Matlab programs on parallel computers. These efforts have faced numerous challenges and none have received widespread acceptance.

In the world of parallel computing the Message Passing Interface (MPI) is the de facto standard for implementing programs on multiple processors. MPI defines C and Fortran language functions for doing point-to-point communication in a parallel program. MPI has proven to be an effective model for implementing parallel programs and is used by many of the world's most demanding applications (weather modeling, weapons simulation, aircraft design, etc.).

MatlabMPI is set of Matlab scripts that implement a subset of MPI and allow any Matlab program to be run on a parallel computer. The key innovation of MatlabMPI is that it implements the widely used MPI "look and feel" on top of standard Matlab file i/o, resulting in a "pure" Matlab implementation that is exceedingly small (~300 lines of code). Thus, MatlabMPI will run on any combination of computers that Matlab supports. In addition, because of its small size, it is simple to download and use (and modify if you like).

### MatlabMPI Page Contents

- [Introduction](#)
- [Download](#)
- [Requirements](#)
- [Installing and Running](#)
- [Launching and File I/O](#)
- [Error Handling](#)
- [Running on Linux](#)
- [Running on MacOSX](#)
- [Running on PC](#)
- [Other Optimizations](#)
- [Running in Batch Mode](#)
- [Other Settings](#)
- [Diagnostics and Troubleshooting](#)
- [First-Time User's Rules of Thumb](#)
- [Files](#)

### pMatlab: Parallel Matlab Toolbox

pMatlab provides a set of Matlab data structures and functions that implement distributed Matlab arrays

[to pMatlab page >](#)

# pMatLab example

The image displays a MATLAB 7.11.0 (R2010b) environment. The main window shows the command prompt with the following output:

```
fx >>
Compute Time (sec)           = 2.7435
Launch+Comm Time (sec)      = 0.0068936
```

A figure window titled "Figure 1" displays a colorful fractal image of the Mandelbrot set, showing the characteristic bulbous shapes and intricate boundary details.

The Editor window shows the script `pMandelbrot.m` with the following code:

```
9 % At the Matlab prompt type
10 % pMandelbrot
11 % To run in parallel with distributed arrays
12 % at the Matlab prompt type
13 % eval(pRUN('pMandelbrot',2,{}))
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15
16 % Set number of iterations, mesh size and threshold.
17 N=2400; Niter=20; epsilon = 0.001;
18 N=800; % Debug
19 PARALLEL = 1; % Set control flag.
20 Wmap = 1; % Create serial map.
21 if (PARALLEL)
22     Wmap = map([Np 1], {}, 0:Np-1); % Create parallel map.
23 % dist(1).dist = 'c'; dist(2).dist = 'b';
24 % Wmap = map([Np 1], dist, 0:Np-1); % 1D Cyclic Distribution.
25 end
26 W = zeros(N,N,Wmap); % Create distributed array
27 Wloc = local(W); % Get local part.
28 myI = global_ind(W,1); % Get local i indices.
29 myJ = global_ind(W,2); % Get local j indices.
30 [ReC ImC] = meshgrid(myJ./(N/2) -1.6, myI./(N/2) -1);
31 Cloc = complex(ReC, ImC); % Initialize C.
32 Zloc = Cloc; % Initialize Z.
33 ieps = 1:numel(Wloc); % Initialize indices.
34 tic; % Start clock.
35 for i=1:Niter; % Compute Mandelbrot set.
36     Zloc(ieps) = Zloc(ieps) .* Zloc(ieps) + Cloc(ieps);
37     Wloc(ieps) = exp(-abs(Zloc(ieps)));
38     ieps = ieps( find(Wloc(ieps) > epsilon) );
39 end
40 W = put_local(W,Wloc); % Put back into W;
41 Tcompute = toc; % Stop clock.
42 disp(['Compute Time (sec) = ', num2str(Tcompute)]);
43
44 tic; % Start Clock.
45 W1 = agg(W); % Aggregate back to leader.
```

# FFT with pMatLab

**MATLAB 7.11.0 (R2010b)**

```
>> profile on
Np = 1
Pid = 0
Distributed vector size (words) = 1048576
Distributed vector size (bytes) = 16777216
Local vector size (bytes) = 16777216
Allocation Time (sec) = 0.11417
Launch Time (sec) = 0.01701
Begin 1st CornerTurn
Begin FFT of 2nd Dimension
Begin 2nd CornerTurn
Compute time (sec) = 0.13974
Communication time (sec) = 0.15662
Run time (sec) = 0.29636
Performance (Gigaflops) = 0.35382
Bandwidth (Gigabytes/sec) = 0.32137
>> profile viewer
```

**Editor - C:\pMatlab\Examples\FFT\pFFT.m**

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % This script implements a simple 1D FFT benchmark us
3 % the most common way of implementing a 1D FFT in par
4 % contains a parallel implementation of this algorithm
5 %
6 % Parameters:
7 % * PARALLEL - Enable the pMatlab library.
8 %
9 % * VALIDATE - Enable validation of FFT result.
10 %
11 % * ERROR_LIMIT - Error limit used in validation.
12 %
13 % * N - length of vector to FFT, must be divisible
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 % To run in serial without distributed arrays, set
16 % PARALLEL = 0
17 % At the Matlab prompt type
18 % pFFT
19 % To run in serial with distributed arrays, set
20 % PARALLEL = 1
21 % At the Matlab prompt type
22 % pFFT
23 % To run in parallel with distributed arrays
24 % at the Matlab prompt type
25 % eval(pRUN('pFFT',2,{}))
26 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
27
28 PARALLEL=1; % Turn parallelism on and
29 N = 2^25*Np; M = N/Np; % Set vector/matrix dim
30 N = 2^20*Np; M = N/Np; % Debug.
31 VALIDATE = 0; % Turn validation on or of
32 ErrorRate = eps; % Set error to machine pre
33
34 Xmap = 1; % Serial map.
35 if PARALLEL
36 Xmap = map([1 Np], {}, 0:Np-1); % Parallel map.
```

**Function details for pFFT**

Links are disabled because this is a static copy of a profile report

**pFFT (1 call, 0.451 sec)**  
Generated 25-Nov-2012 21:05:34 using cpu time.  
script in file C:\pMatlab\Examples\FFT\pFFT.m  
Copy to new window for comparing multiple runs

Parents (calling functions)  
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
86	Xloc = fft(Xloc,[],2);	1	0.123 s	27.3%	
45	X = complex(rand(1,N,Xmap),ran...	1	0.077 s	17.1%	
81	X = transpose_grid(X);	1	0.063 s	14.0%	
92	X = transpose_grid(X);	1	0.043 s	9.5%	
100	X = transpose_grid(X);	1	0.041 s	9.1%	
All other lines			0.104 s	23.1%	
Totals			0.451 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
transpose_grid	function	3	0.147 s	32.6%	
map.rand	function	2	0.065 s	14.4%	
num2str	function	12	0.012 s	2.7%	
dmat.agg	function	1	0.011 s	2.4%	
dmat.complex	function	1	0.010 s	2.2%	
map.zeros	function	2	0.006 s	1.3%	
map.map	function	1	0.003 s	0.7%	
Pid	function	2	0.001 s	0.2%	

## Available examples:

- xbasic.m Extremely simple MatlabMPI program that prints out the rank of each processor.
- basic.m Simple MatlabMPI program that sends data from processor 1 to processor 0.
- multi\_basic.m Simple MatlabMPI program that sends data from processor 1 to processor 0 a few times.
- probe.m Simple MatlabMPI program that demonstrates the using MPI\_Probe to check for incoming messages.
- broadcast.m Tests MatlabMPI broadcast command.
- basic\_app.m Examples of the most common usages of MatlabMPI.
- basic\_app2.m Examples of the most common usages of MatlabMPI.
- basic\_app3.m Examples of the most common usages of MatlabMPI.
- basic\_app4.m Examples of the most common usages of MatlabMPI.
- blurimage.m MatlabMPI test parallel image processing application.
- speedtest.m Times MatlabMPI for a variety of messages.
- synch\_start.m Function for synchronizing starts.
- machines.m Example script for creating a machine description.
- unit\_test.m Wrapper for using an example as a unit test.
- unit\_test\_all.m Calls all of the examples as way of testing the entire library.
- unit\_test\_mcc.m Wrapper for using an example as a mcc unit test.
- unit\_test\_all\_mcc.m Calls all of the examples using MPI\_cc as way of testing the entire library.



```
% Initialize MPI.
MPI_Init;

% Create communicator.
comm = MPI_COMM_WORLD;

% Modify common directory from default for better performance.
% comm = MatMPI_Comm_dir(comm, '/tmp');

% Get size and rank.
comm_size = MPI_Comm_size(comm);
my_rank = MPI_Comm_rank(comm);

% Print rank.
disp(['my_rank: ', num2str(my_rank)]);

% Wait momentarily.
pause(2.0);

% Finalize Matlab MPI.
MPI_Finalize;
disp('SUCCESS');
if (my_rank ~= MatMPI_Host_rank(comm))
    exit;
end
```

# Demo folder ~/matlab/, watch top at the other machine

```
vdwarf2.ee.bgu.ac.il - PuTTY
vdwarf2.ee.bgu.ac.il> matlab -nodesktop -nodisplay -nojvm

      < M A T L A B >
  Copyright 1984-2007 The MathWorks, Inc.
    Version 7.5.0.338 (R2007b)
      August 9, 2007

-----
Your MATLAB license will expire in 11 days.
Please contact your system administrator or
The MathWorks to renew this license.
-----

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> eval( MPI_Run('xbasic', 2, {'vdwarf3', 'vdwarf4'}) );
Launching MPI rank: 1 on: vdwarf4
Launching MPI rank: 0 on: vdwarf3

unix_launch =

  rsh vdwarf4 -n 'cd /users/agnon/misc/tel-zur/matlab; /bin/sh ./MatMPI/Unix_Comm
ands.vdwarf4.1.sh &' &
  rsh vdwarf3 -n 'cd /users/agnon/misc/tel-zur/matlab; /bin/sh ./MatMPI/Unix_Comm
ands.vdwarf3.0.sh &' &

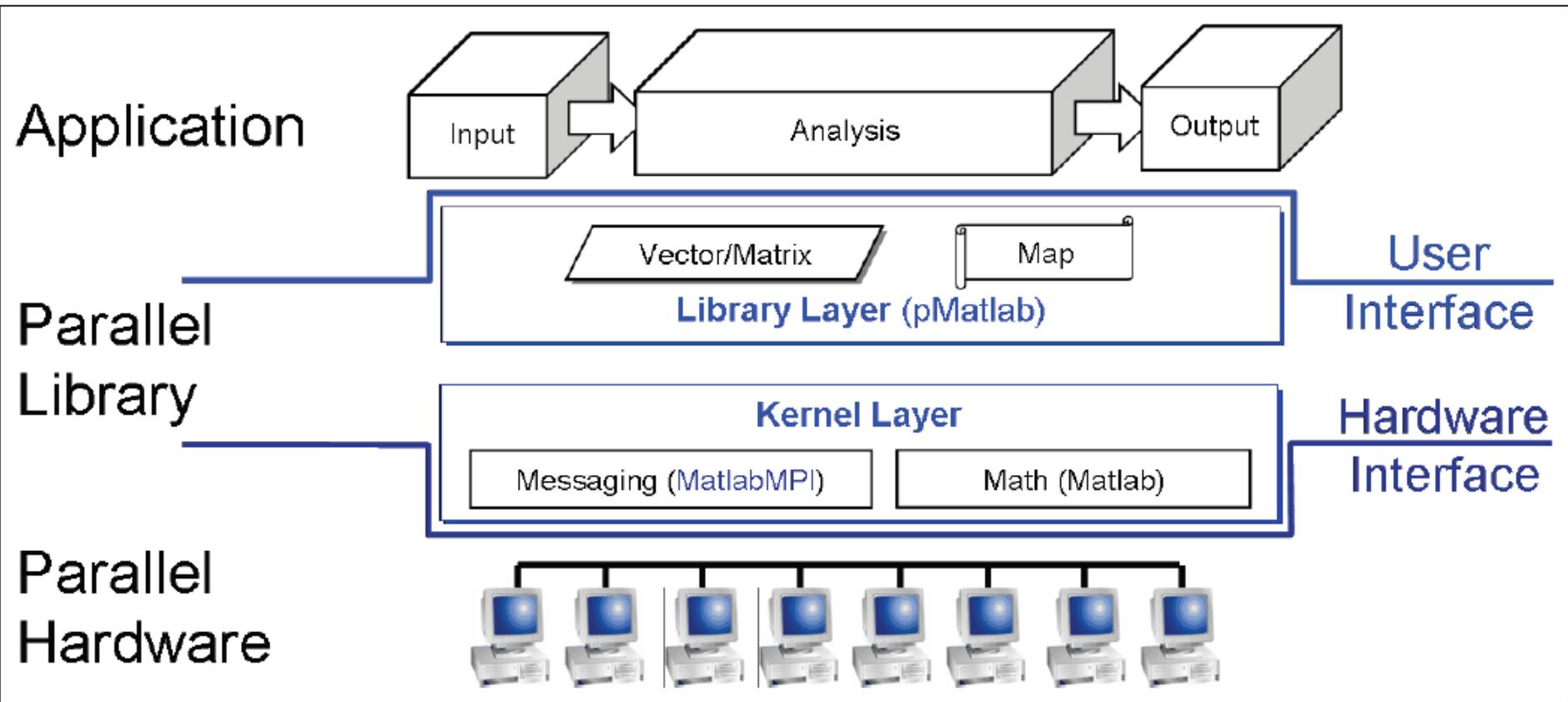
>>
>>
>>
>>
>>
>>
>>
>>
>>
```

# Parallel Matlab (Octave) using pMatlab

Global arrays – “...Communication is hidden from the programmer; arrays are automatically redistributed when necessary, without the knowledge of the programmer...”

“...The ultimate goal of pMatlab is to move beyond basic messaging (and its inherent programming complexity) towards higher level parallel data structures and functions, allowing any MATLAB user to parallelize their existing program by simply changing and adding a few lines,

Source: [http://www.ll.mit.edu/mission/isr/pmatlab/pMatlab\\_intro.pdf](http://www.ll.mit.edu/mission/isr/pmatlab/pMatlab_intro.pdf)



**Figure 11 – Parallel MATLAB consists of two layers. pMatlab provides parallel data structures and library functions. MatlabMPI provides messaging capability.**

# Parallel Computing with Matlab on Amazon Cloud

## MATLAB Parallel Computing Tools: Basic Setup and Requirements

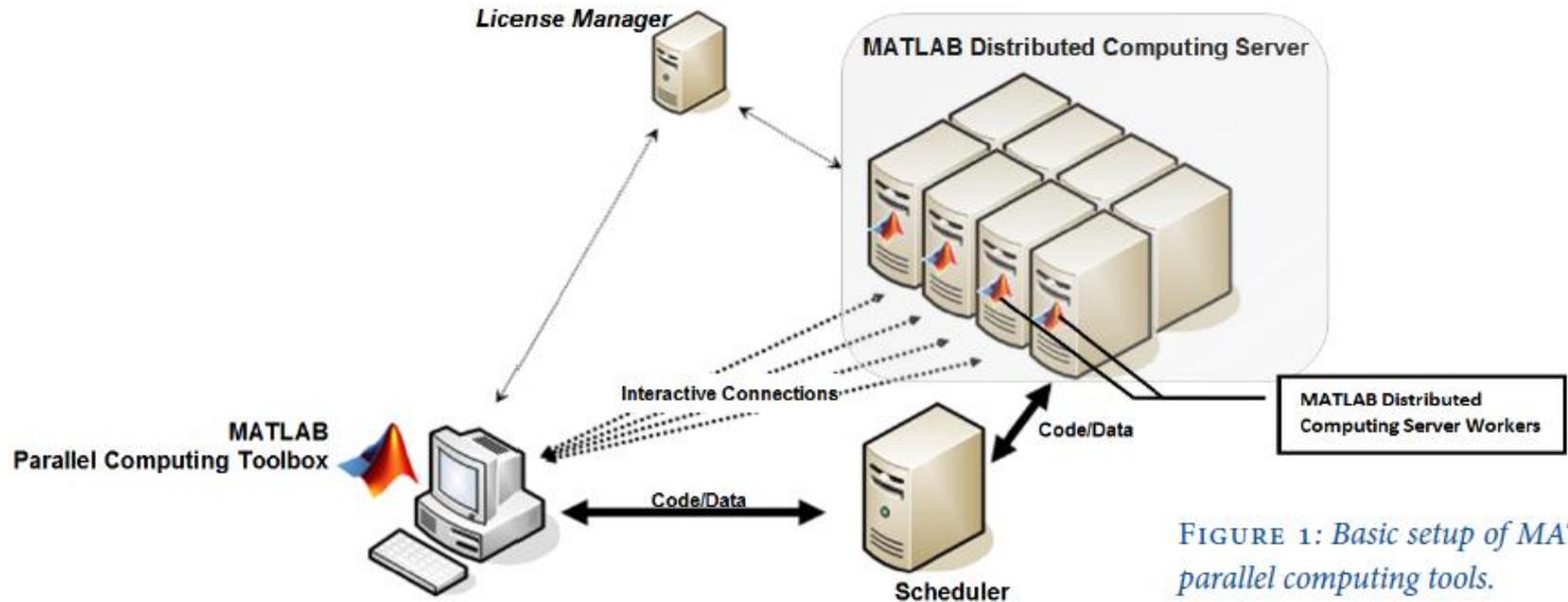


FIGURE 1: Basic setup of MATLAB parallel computing tools.

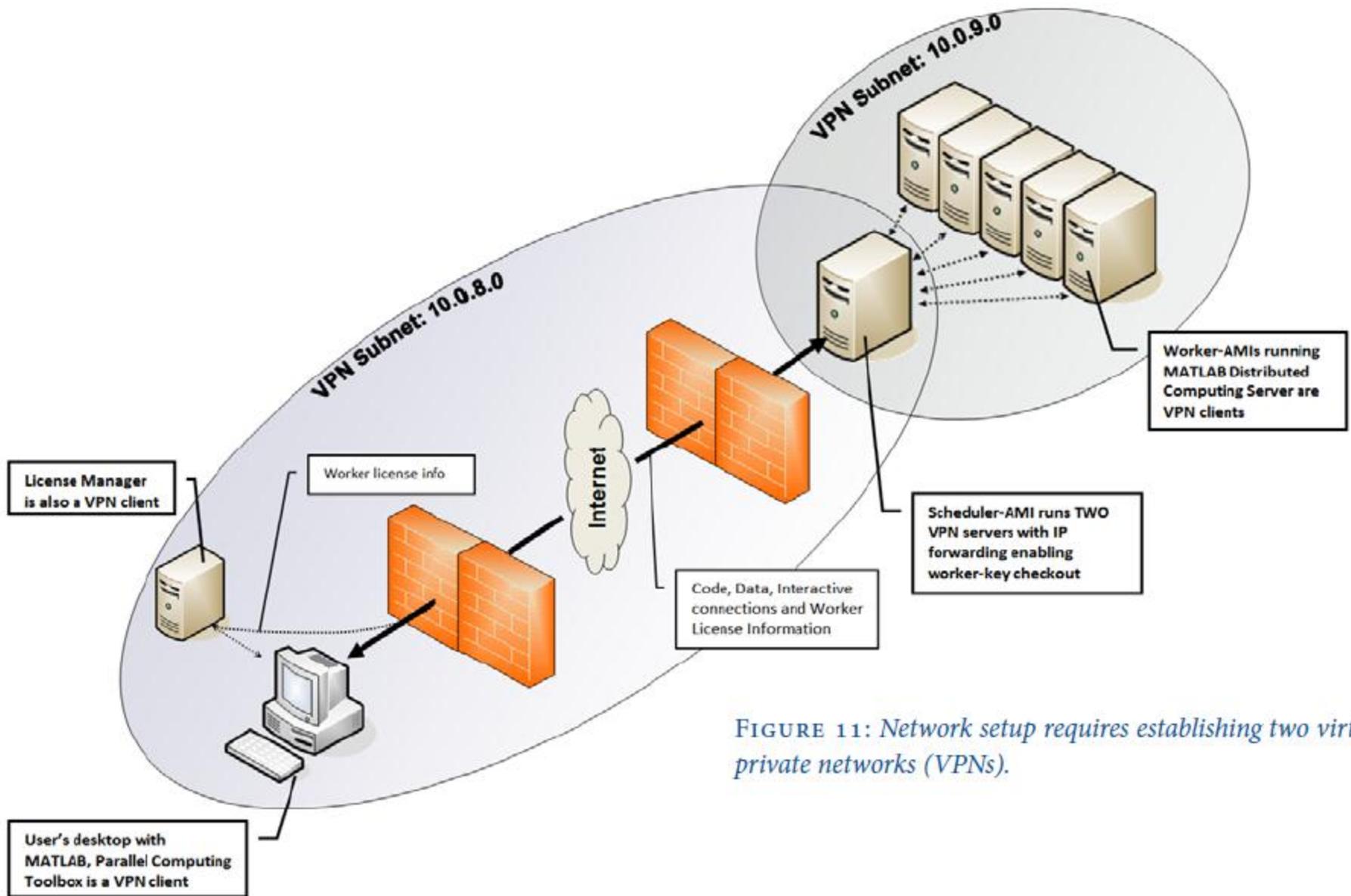


FIGURE 11: Network setup requires establishing two virtual private networks (VPNs).

# Example -Problem Description

- System of  $2^6$  (=64) square matrices
  - Each matrix  $\rightarrow$  Sparse, square,  $2^{17}$  (=131072) dimension
  - Matrix is generated by 'spdiags' using a 'random' array
- To extract first 100 eigenvectors
  - 'eigs' function is used
- See handout for the code
- Each matrix calculation is distributed

# Example - Serial Matlab

- 'eigen' is a function
  - Input : (vector of random numbers, dimension of the matrix)
  - Output : eigenvectors

```
        n = 2^16;  
        p = 2^7;  
        e = rand (n,1,p);  
  
        for i = 1 : p  
            a = e(:, :, i);  
            ans(i) = eigen(a, n);  
        end
```

} Independent calculations

# Example-Parallel Matlab

## 1. Find available distributed computing resources (`findResource` function)

```
nprocs = [ getenv('DMATLAB_NPROCS') ]  
np = sscanf( nprocs, '%d' )  
mgr_name = [ getenv('JOBMANAGER') ]  
mgr_host = [ getenv('JOBMANAGERHOST') ]  
  
jm =findResource('jobmanager','Name',mgr_name,'LookupURL',mgr_host);
```



## 2. Create distributed job

```
j = createJob(jm,'FileDependencies',{<path>});
```



Path to additional files

# Example – Parallel Matlab

## 3. Create Tasks for each worker

```
n = 2^16;  
p = 2^7;  
e = rand (n,1,p);
```

```
for i = 1 : p  
    a = e(:, :, i);
```

```
createTask(j, @eigen, 1, {a,n}); ← Creating tasks
```

```
end
```

Name of  
the job

Parallel Task  
function

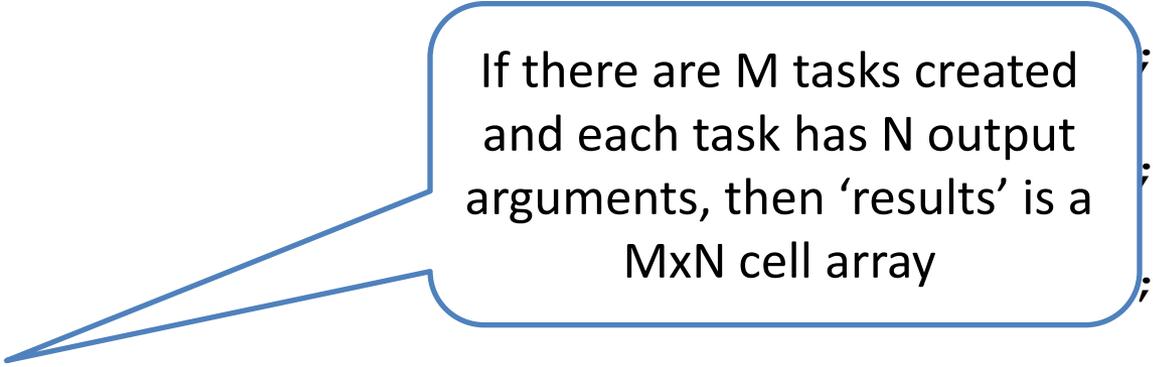
Number of  
output  
arguments

Input arguments  
to each task

Same as Serial Code

# Example – Parallel Matlab

## 4. Submit the job and wait for the results



If there are M tasks created and each task has N output arguments, then 'results' is a MxN cell array

```
results = getAllOutputArguments(j)
```

## 5. Remove the individual task or parent job object

```
destroy(j);
```