Parallel Computing with MATLAB

Shiran Golan

Application Engineer

shirang@systematics.co.il

Outline

- Parallel Computing Overview
- Parallel Computing Paradigm
  - Multicore Desktops
  - Cluster Hardware
- Programming Parallel Applications
- Using parfor Loops
- Parallel Computing Beyond Parfor

Parallel Computing and Acceleration

Approach	Options	Learn more on MATLAB Documentation
Apply best coding practices	• Preallocation • Vectorization	Performance and Memory
Use explicit parallelism	• CPU • GPU • Cluster	MATLAB Parallel Computing
Integrate with other languages	• C • C++ • Fortran	MATLAB API for Other Languages

Why Parallel Computing?

- **Why parallel computing?**
  - Need faster insight into your data
  - Computing infrastructure is broadly available (multicore desktops, clusters, gpus)
- **Why parallel computing with MATLAB?**
  - Accelerate workflows with minimal code changes
  - Focus on engineering and research, not computation

Utilizing Additional Processing Power

- **Built-in multithreading**
  – Automatically enabled in MATLAB since R2008a
  – Functions such as : fft, eig, svd, sort, etc.
  – Leverage multicore CPUs

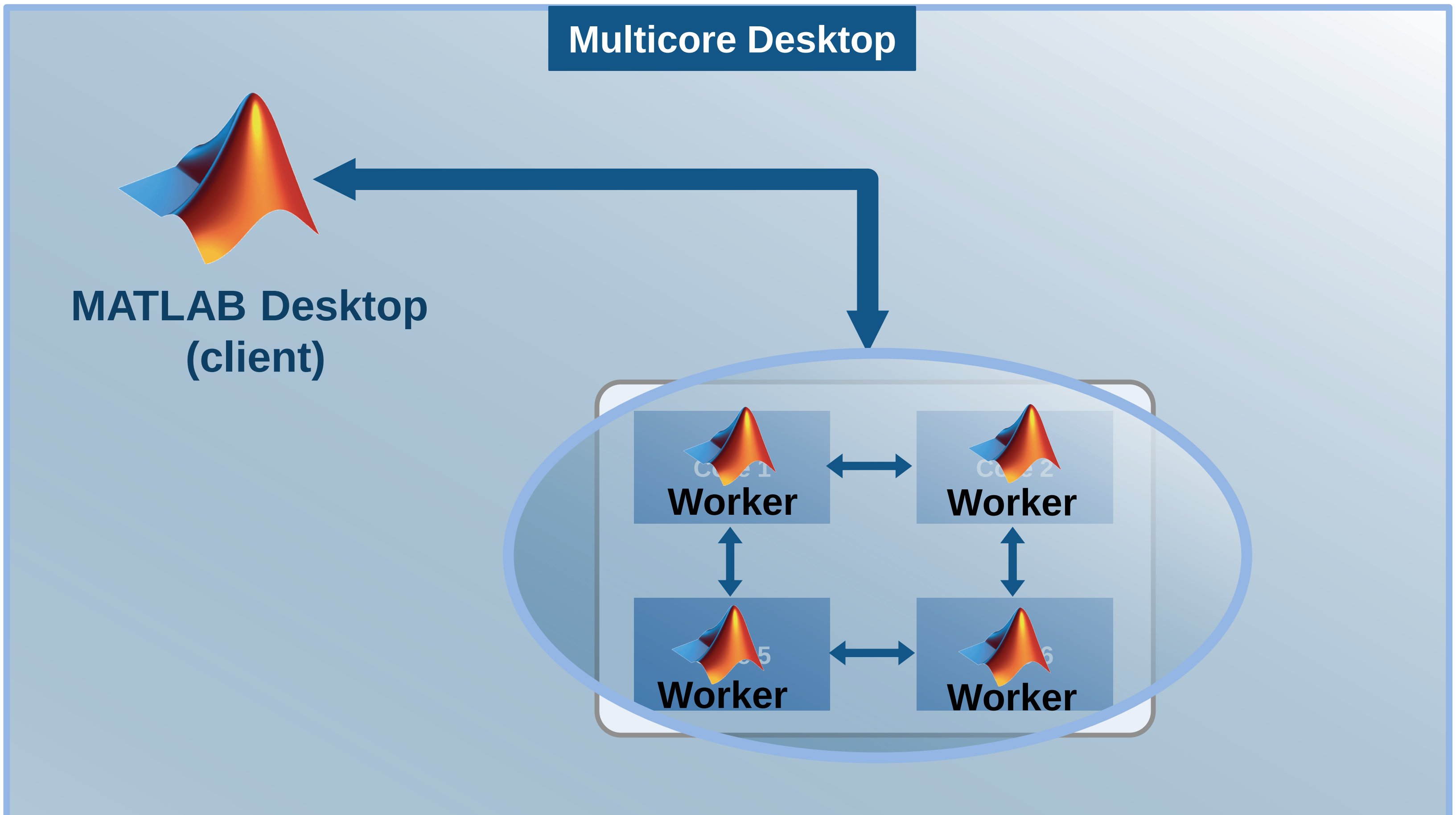
- **Parallel Computing Tools**
  – High-level constructs let you parallelize MATLAB applications
  – For a variety of applications
  – Leverage CPUs, GPUs, and scale to clusters and clouds

Outline

- Parallel Computing Overview
- Parallel Computing Paradigm
  - Multicore Desktops
  - Cluster Hardware
- Programming Parallel Applications
- Using parfor Loops
- Parallel Computing Beyond Parfor

Parallel Computing Paradigm

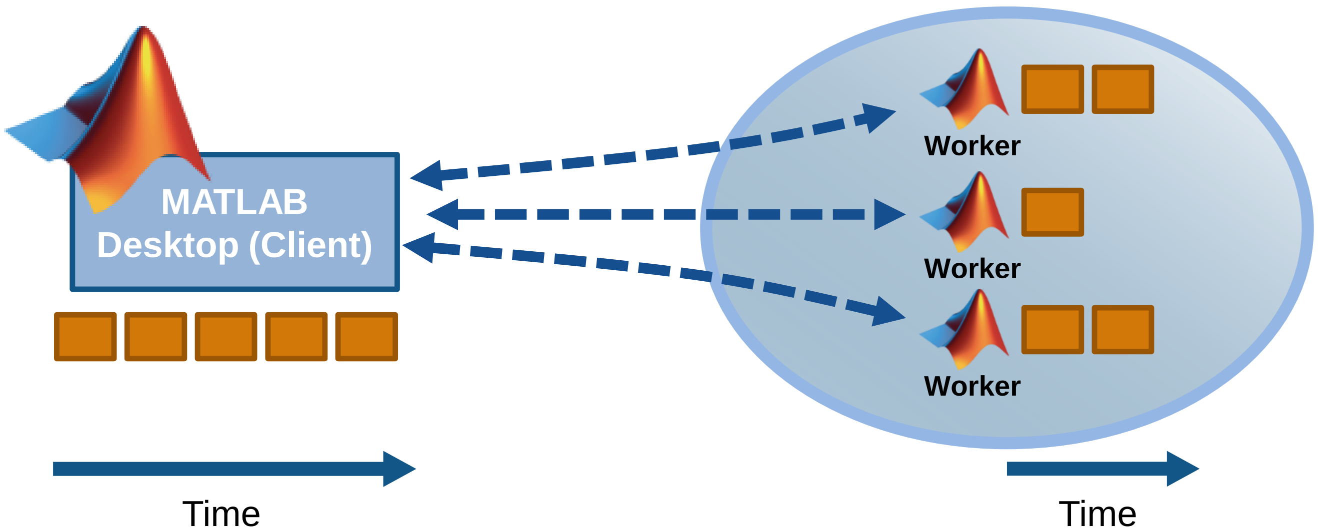
Multicore Desktops



Embarrassingly or Explicit Parallelism: Independent Tasks or Iterations

Parallel for loops

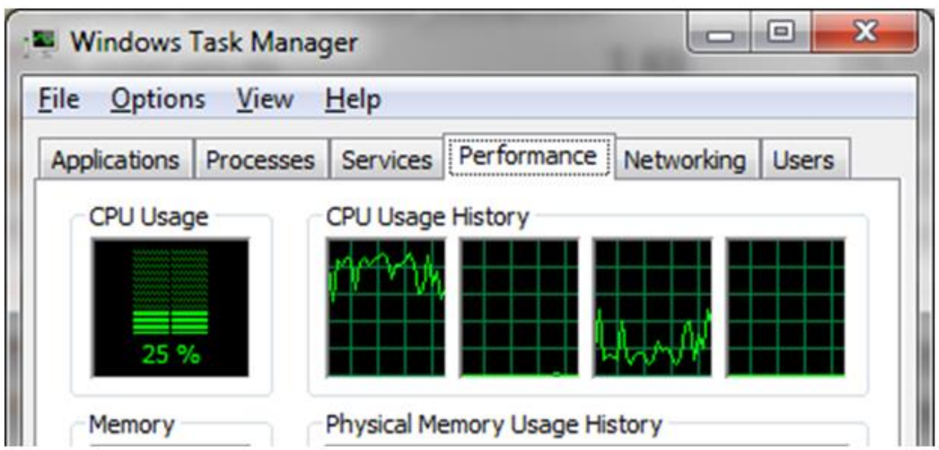
- Ideal problem for parallel computing
- No dependencies or communications between tasks
- Examples: parameter sweeps, Monte Carlo simulations

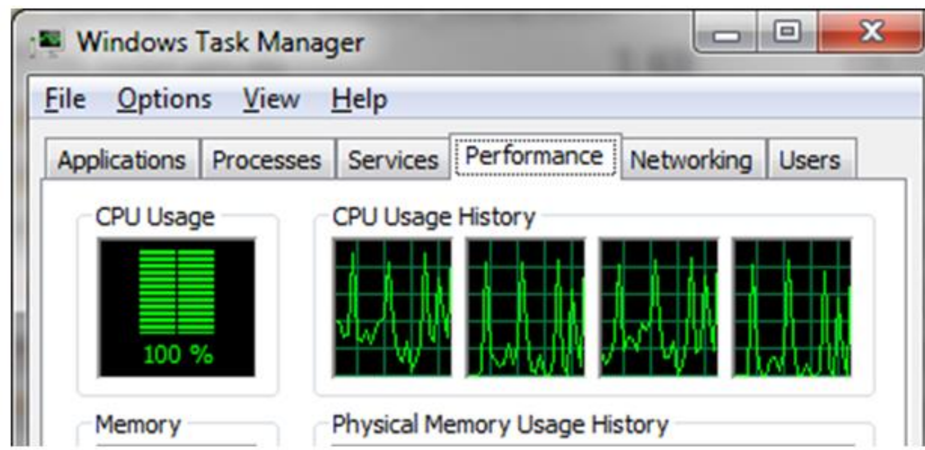


Parallel Computing Toolbox

Solve computationally and data-intensive problems using:

- multicore processors

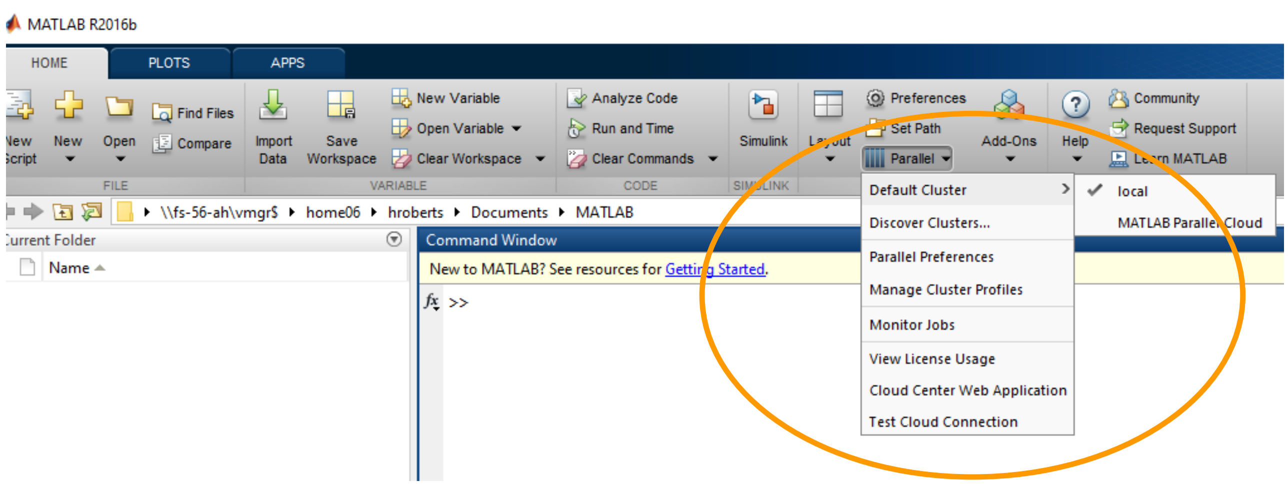




- computer clusters (requires MDCS)
- GPUs

Parallel Computing Toolbox

- Enables **explicit** use of multi-core processors
- Provides high-level tools and infrastructure for:
  - **creation** and **monitoring** of jobs
  - **allocating** hardware resources
- Provides a **bridge** to cluster hardware

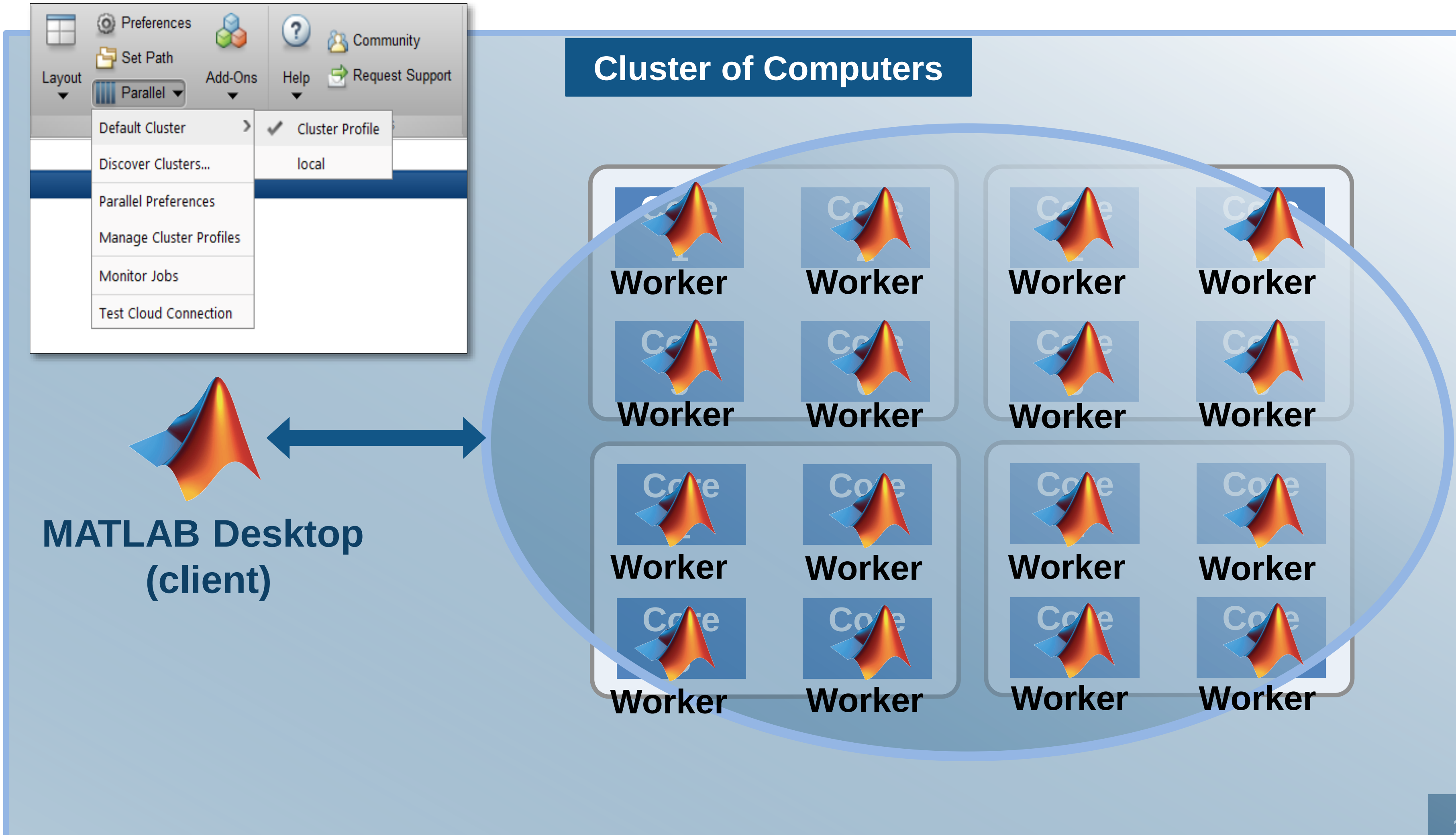


Outline

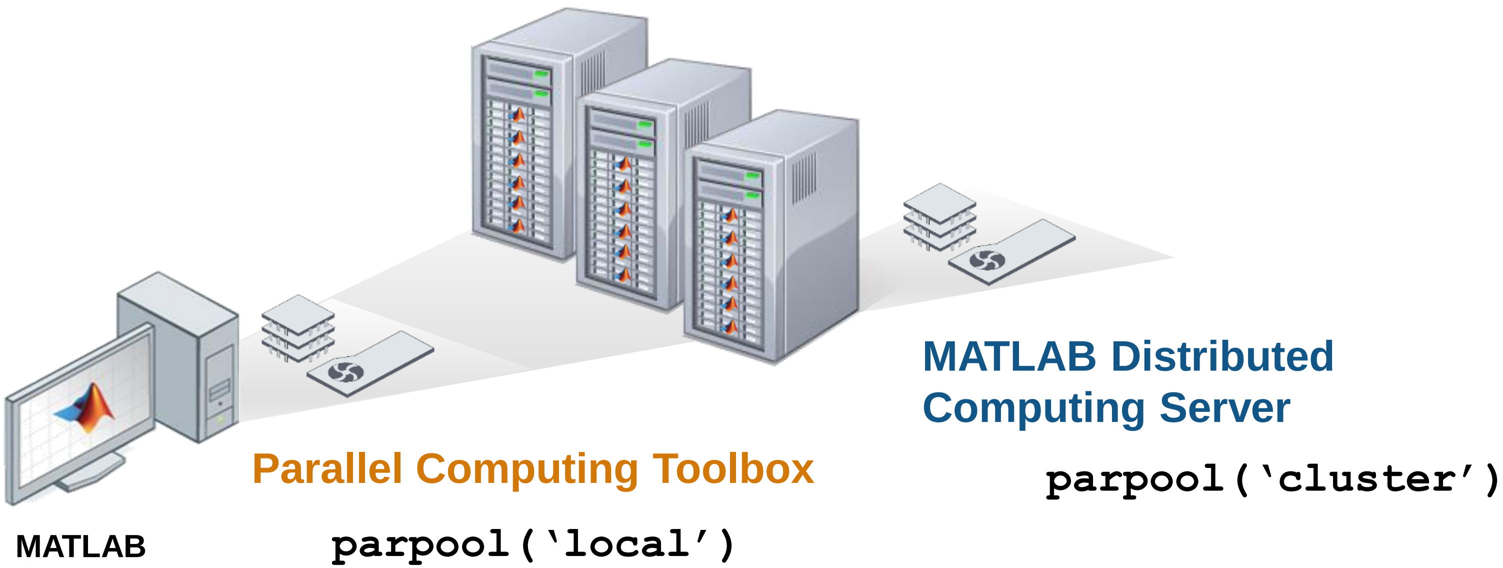
- Parallel Computing Overview
- Parallel Computing Paradigm
  - Multicore Desktops
  - Cluster Hardware
- Programming Parallel Applications + Example
- Using parfor Loops + Example
- Parallel Computing Beyond Parfor

Parallel Computing Paradigm

Cluster Hardware



Scale parfor

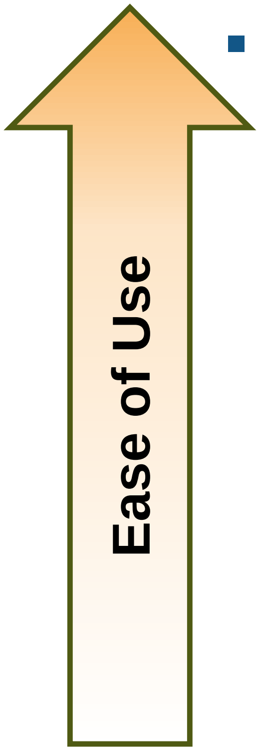


- Develop your application once
- Change run environment by changing the profile

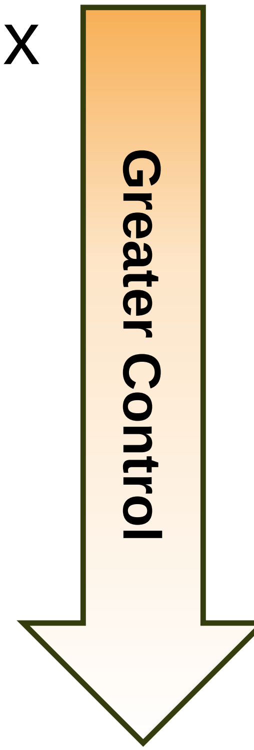
Outline

- Parallel Computing Overview
- Parallel Computing Paradigm
  - Multicore Desktops
  - Cluster Hardware
- Programming Parallel Applications
- Using parfor Loops
- Parallel Computing Beyond Parfor

Programming Parallel Applications (1 of 3)



- Enable with Parallel Computing Toolbox
  - Use same code with core MATLAB
  - Examples
    - Specific functions in many `toolboxes`
    - `parfor`
    - `batch`
    - `mapreduce`



Parallel-enabled Toolboxes

Enable parallel computing support by setting a flag or preference

Image Processing

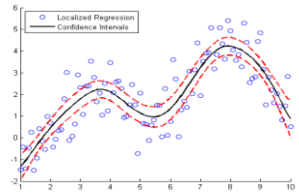
Batch Image Processor, Block Processing, GPU-enabled functions


Original Image of Peppers


Recolored Image of Peppers

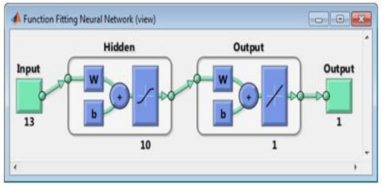
Statistics & Machine Learning

Resampling Methods, k-Means clustering, GPU-enabled functions



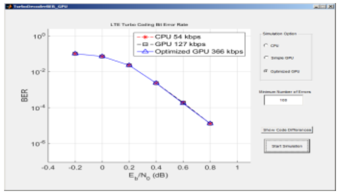
Neural Networks

Deep Learning, Neural Network training and simulation



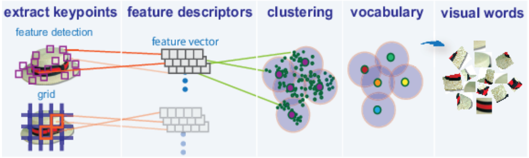
Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



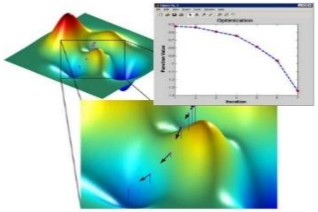
Computer Vision

Parallel-enabled functions in bag-of-words workflow



Optimization

Parallel estimation of gradients



Other Parallel-enabled Toolboxs:

www.mathworks.com/products/parallel-computing/parallel-support.html

Programming Parallel Applications (2 of 3)

Ease of Use

- Enable by changing input data type
  - Use same algorithmic code with core MATLAB
  - Examples
    - Functions overloaded for **`gpuArray`**
    - Functions overloaded for **`codistributed`**

Greater Control

www.mathworks.com/help/distcomp/using-matlab-functions-on-codistributed-arrays.html
www.mathworks.com/help/distcomp/run-built-in-functions-on-a-gpu.html

Programming Parallel Applications (3 of 3)

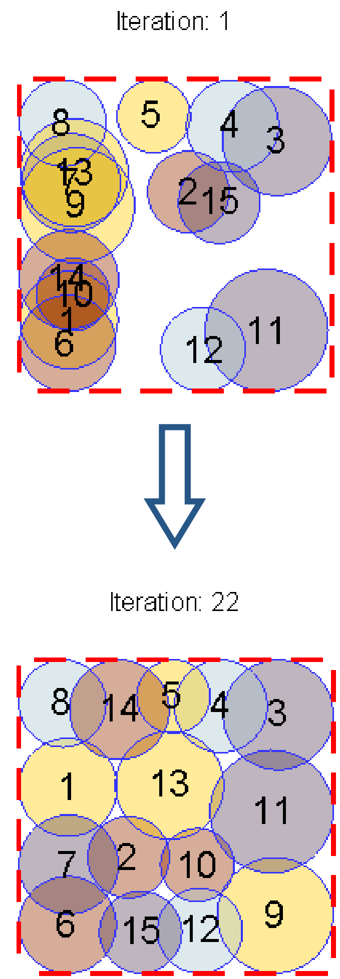
Ease of Use

- Build code with Parallel Computing API
  - Applications require Parallel Computing Toolbox
  - Examples:
    - `spmd, gop`
    - Message passing
    - Job submission functions
    - CUDA kernels

Greater Control

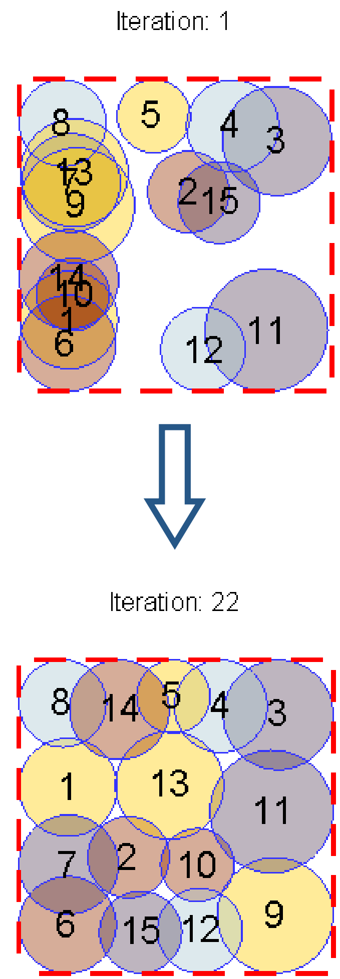
Example: Optimizing Tower Placement

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



Summary of Example

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



Parallel Computing Support in Optimization Toolbox

- Functions:
  - **`fmincon`**

    Finds a constrained minimum of a function of several variables
  - **`fminimax`**

    Finds a minimax solution of a function of several variables
  - **`fgoalattain`**

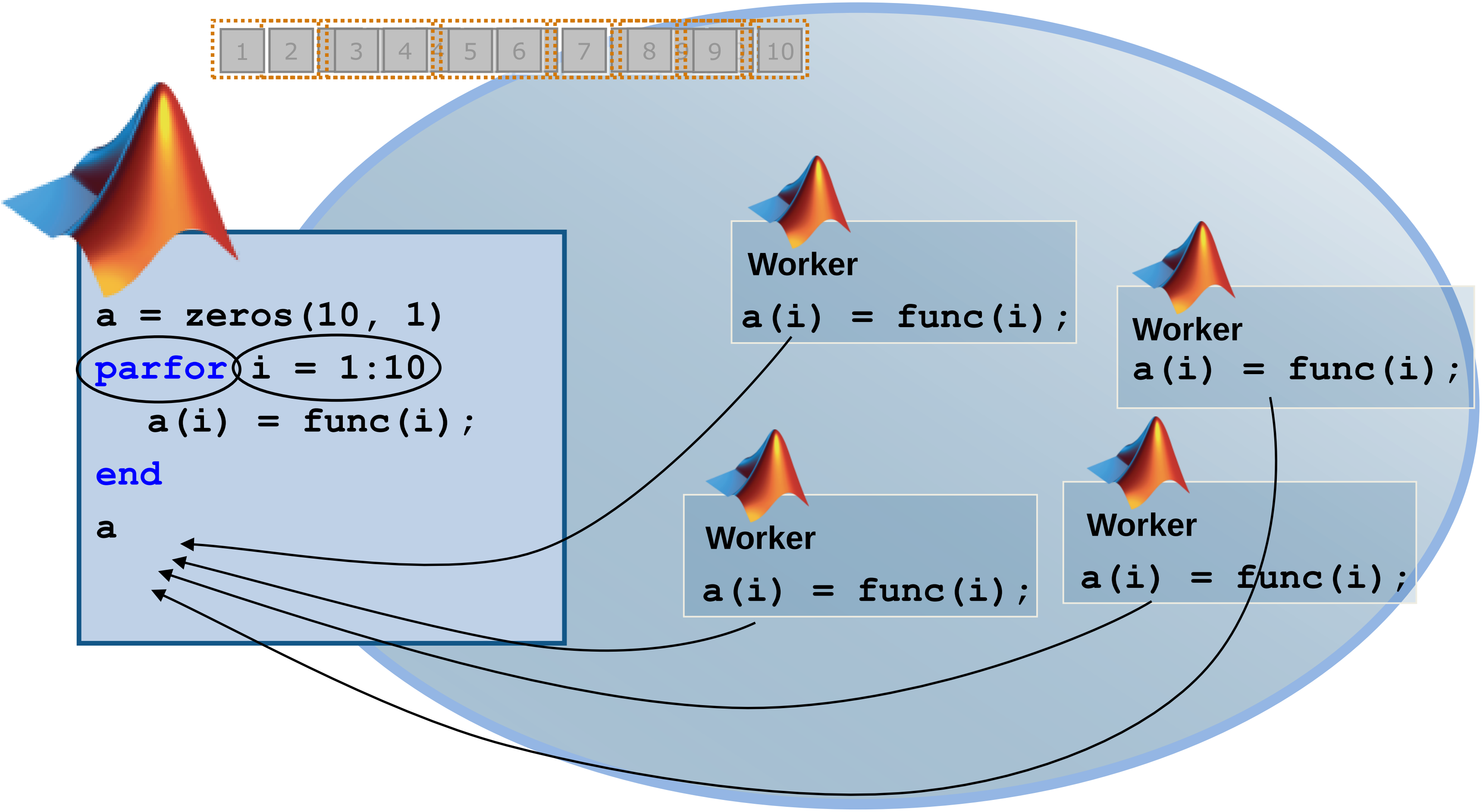
    Solves the multiobjective goal attainment optimization problem

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

Outline

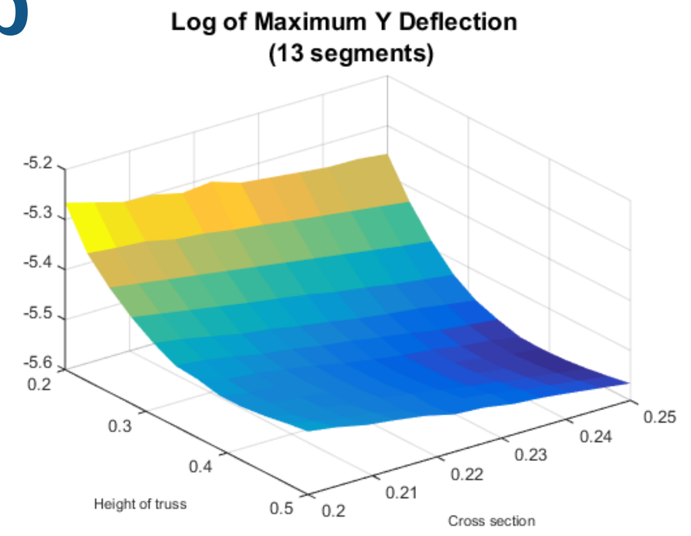
- Parallel Computing Overview
- Parallel Computing Paradigm
  - Multicore Desktops
  - Cluster Hardware
- Programming Parallel Applications
- Using parfor Loops
- Parallel Computing Beyond Parfor

The Mechanics of `parfor` Loops



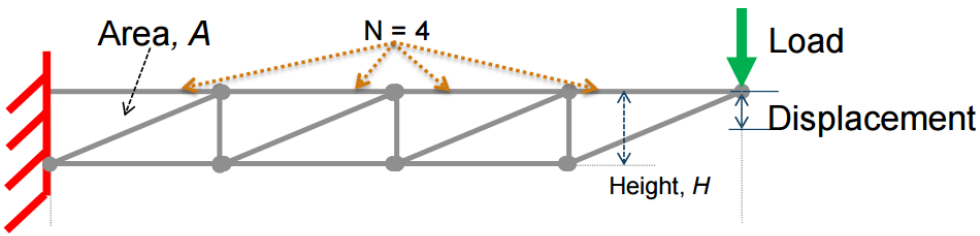
Pool of MATLAB Workers

Example: Parameter Sweep Parallel for-loops



- **Deflection** of customizable truss
  Parameters investigated:
  - Height of truss
  - Cross sectional area of truss elements

- Convert `for` to `parfor`



- Use pool of MATLAB workers

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

Parfeval: N=3*p.NumWorkers; -similar to what parfor does

Converting `for` to `parfor`

- Requirements for `parfor` loops:
  - Task independent
  - Order independent
- Constraints on the loop body:
  - Cannot “introduce” variables (e.g. `load`, etc.)
  - Cannot contain `break` or `return` statements
  - Cannot contain `evalc, eval, evalin, assignin`
  - Cannot contain a call to scripts directly
  - Cannot contain another `parfor` loop

Advice for Converting `for` to `parfor`

- Use **Code Analyzer** to diagnose `parfor` issues
- If your `for` loop cannot be converted to a `parfor`, consider **wrapping** a subset of the body to a function
- Read the section in the documentation on **classification** of variables
  www.mathworks.com/help/distcomp/classification-of-variables-in-parfor-loops.html
- **Blog Post**: Using parfor Loops
  blogs.mathworks.com/loren/2009/10/02/using-parfor-loops-getting-up-and-running/

Outline

- Parallel Computing Overview
- Parallel Computing Paradigm
  - Multicore Desktops
  - Cluster Hardware
- Programming Parallel Applications
- Using parfor Loops
- Parallel Computing Beyond Parfor

Parallel Computing with MATLAB – Beyond PARFOR

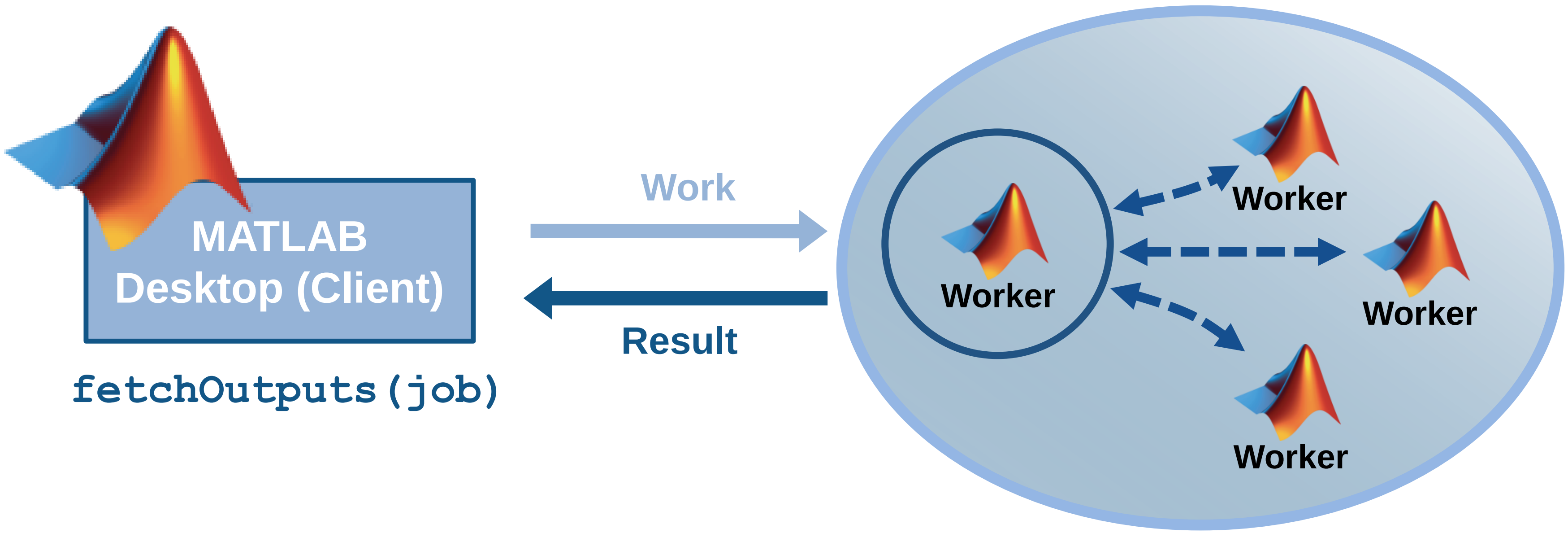
Well-known features

- parallel-enabled toolboxes
- `parfor`
- `gpuArray`

Advanced support

- batch submission, jobs and tasks
  `batch, createJob, createTask`
- asynchronous execution on pool
  `parfeval`
- parallel support for big data
  `tall, mapreduce`
  distributed arrays ("global arrays") -
  `distributed, codistributed`
- Single Program Multiple Data - `SPMD`
- message passing
  `labSend, labReceive`

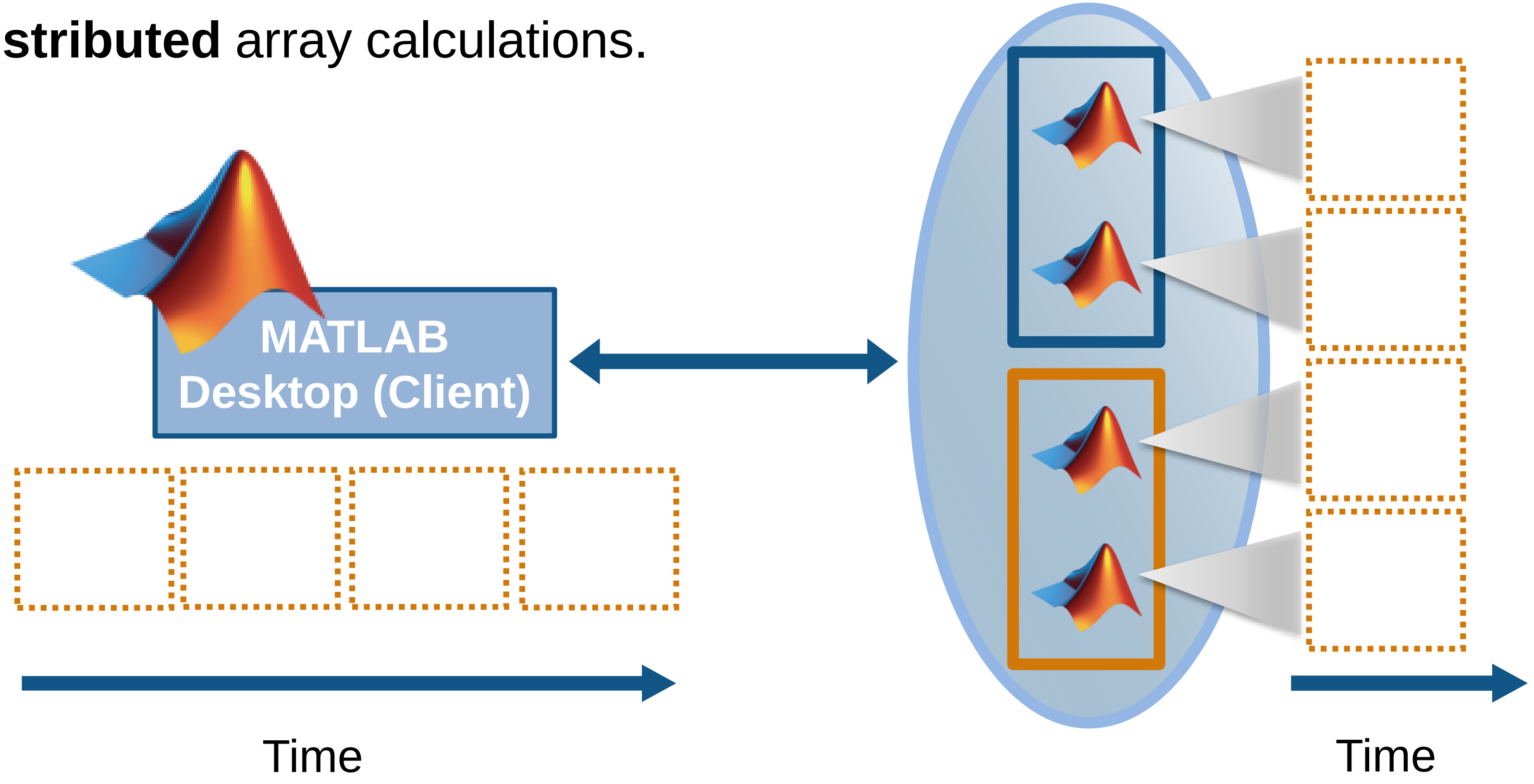
Offload and Scale Computations with `batch`



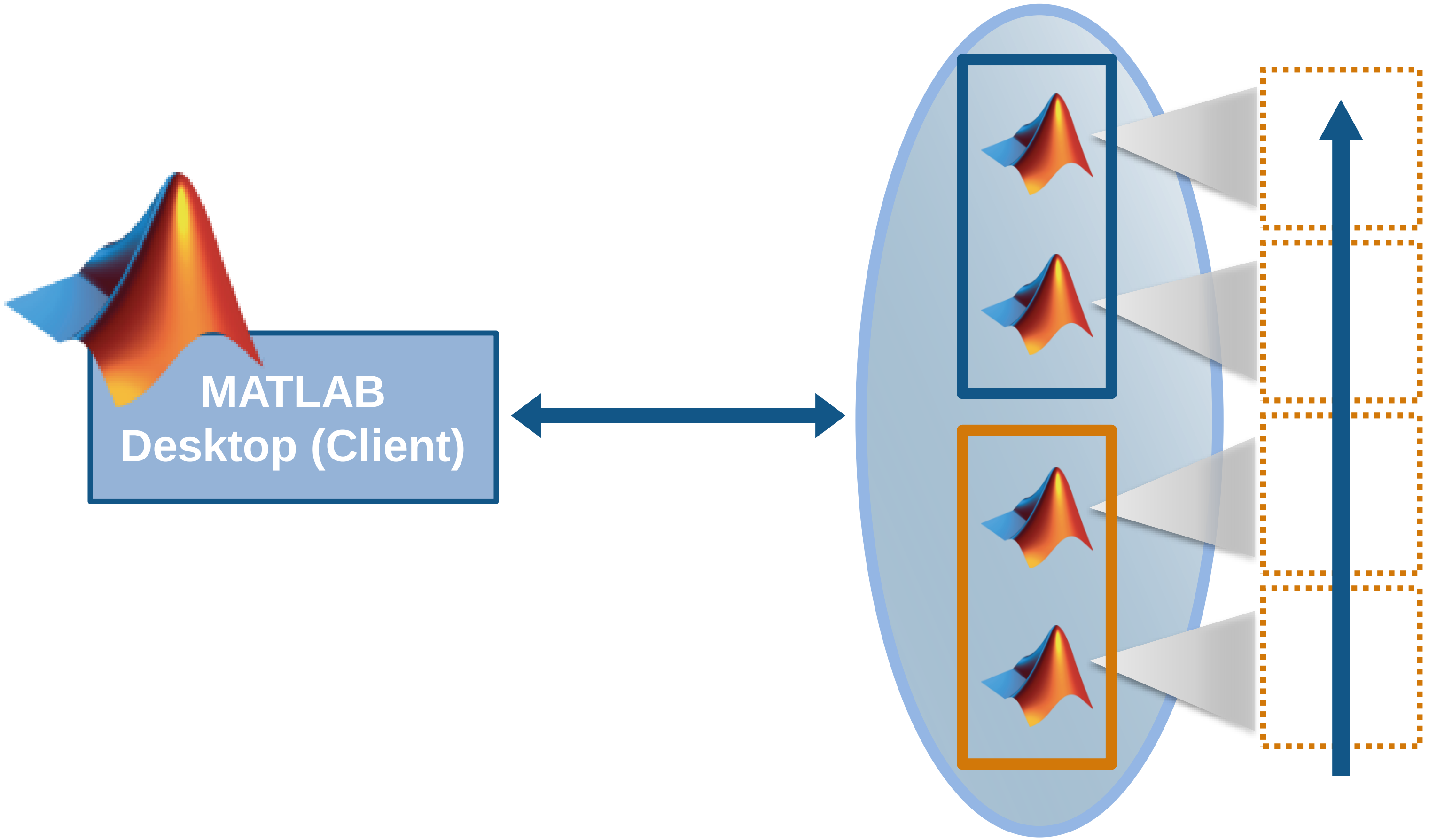
fetchOutputs(job)

Execution with `spmd`

- A way to execute a block of code once on each worker in a pool.
- Can provide some time savings due to parallel execution, but it is most often used either to **set worker environments** or as a **building block for distributed** array calculations.



Execution with gop



Summary

- Parallel computing with MATLAB is **explicit parallelism** that uses **multiple MATLAB computation engines**
- Once you’ve written an application with **supported functions**, you can execute in parallel by **opening a parallel pool**
- Develop parallel applications on the desktop and then **scale** to clusters as needed

Speeding up MATLAB using the GPU

What is a GPU?

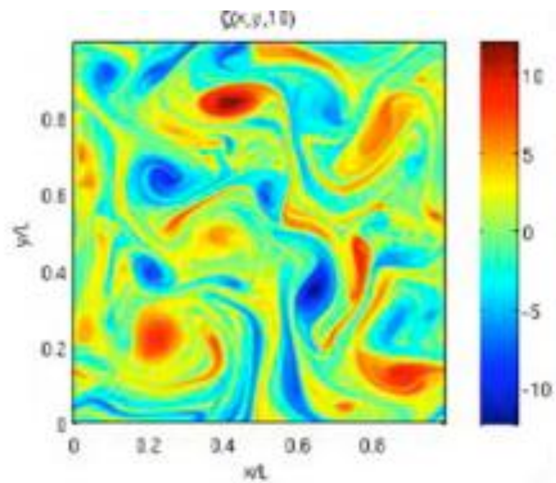
Graphical Processing Unit

NVIDIA GeForce GT 730M	
# Cores	384, @719MHz
Memory	2 GB, DDR 3

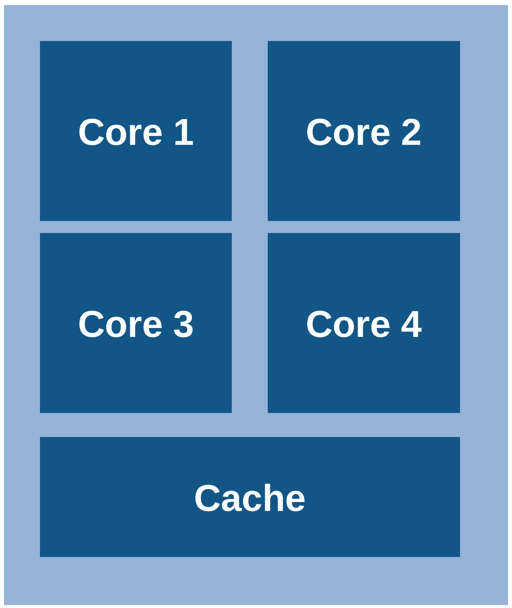
History of GPUs

3D Gaming & CAD → Scientific Computing

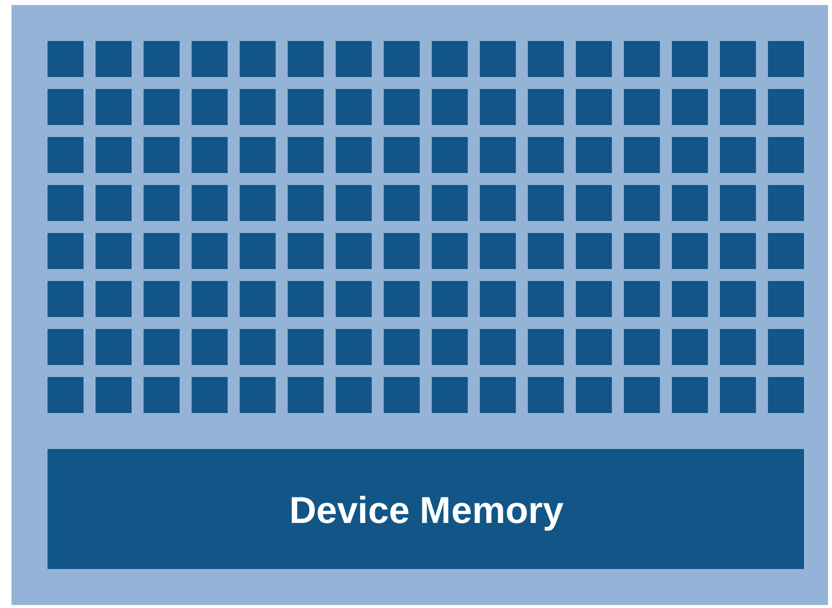
→



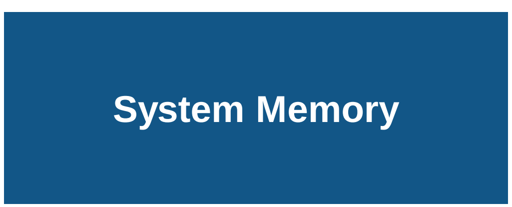
CPUs vs. GPUs



CPU (Multiple Cores)



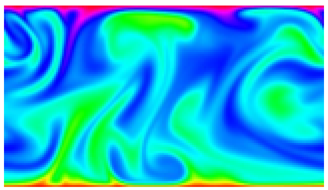
GPU (Hundreds of Cores)



CPU	GPU
Sequential code, handles branching well	Simpler computational code
Large, low-latency memory	Small, high-latency memory
Several cores	Hundreds of cores
Can work on all data types	Strictly numerical data

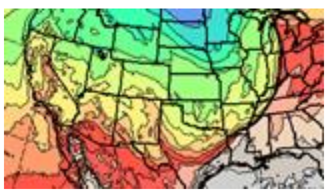
Problems for Running on the GPU

A selection of problems from the CUDA Community Showcase:

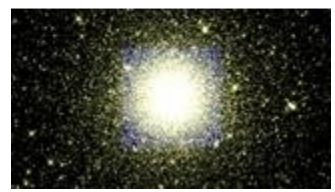


Computational Fluid Dynamics

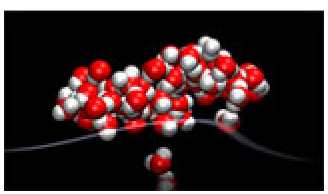
Computational Finance



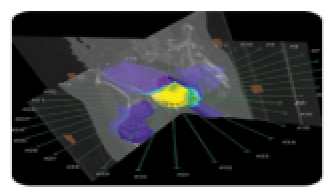
Weather Modeling



N-Body Simulations



Molecular Modeling



Digital Signal Processing

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

The Language of GPUs

Kernel	Code running on the GPU
Device	The card containing the GPU and memory
Host	The CPU and system memory
CUDA	Compute Unified Device Architecture – Language designed for general purpose use of GPU

Three Levels of Support

1) Invoking built-in functions on arrays existing on the GPU

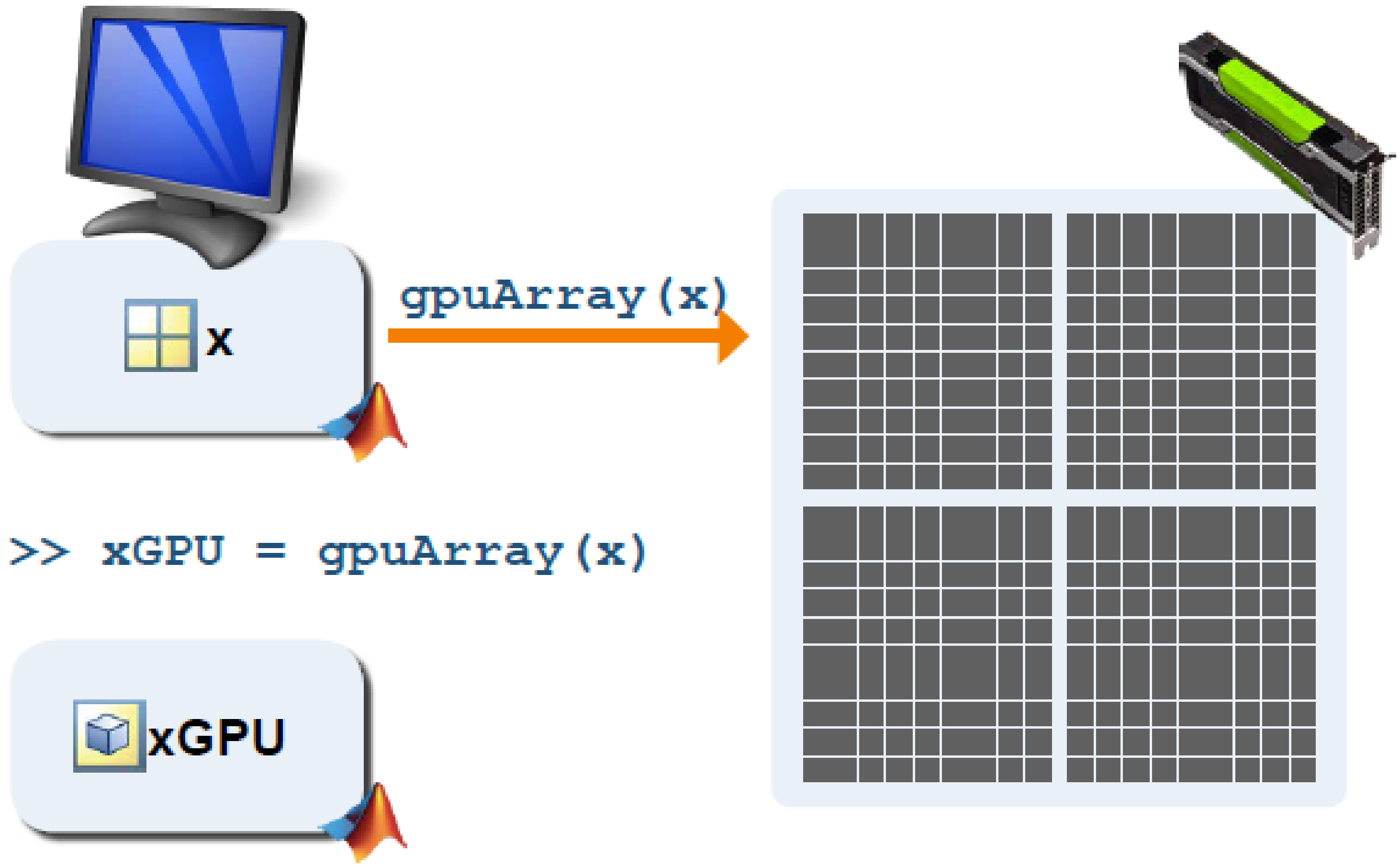
2) Translating simple MATLAB functions to run on the GPU

3) Invoking CUDA code from MATLAB

nVIDIA Solutions

Compute capability = 2.0 or higher (as of R2014b)

1. Invoking Built-In Functions

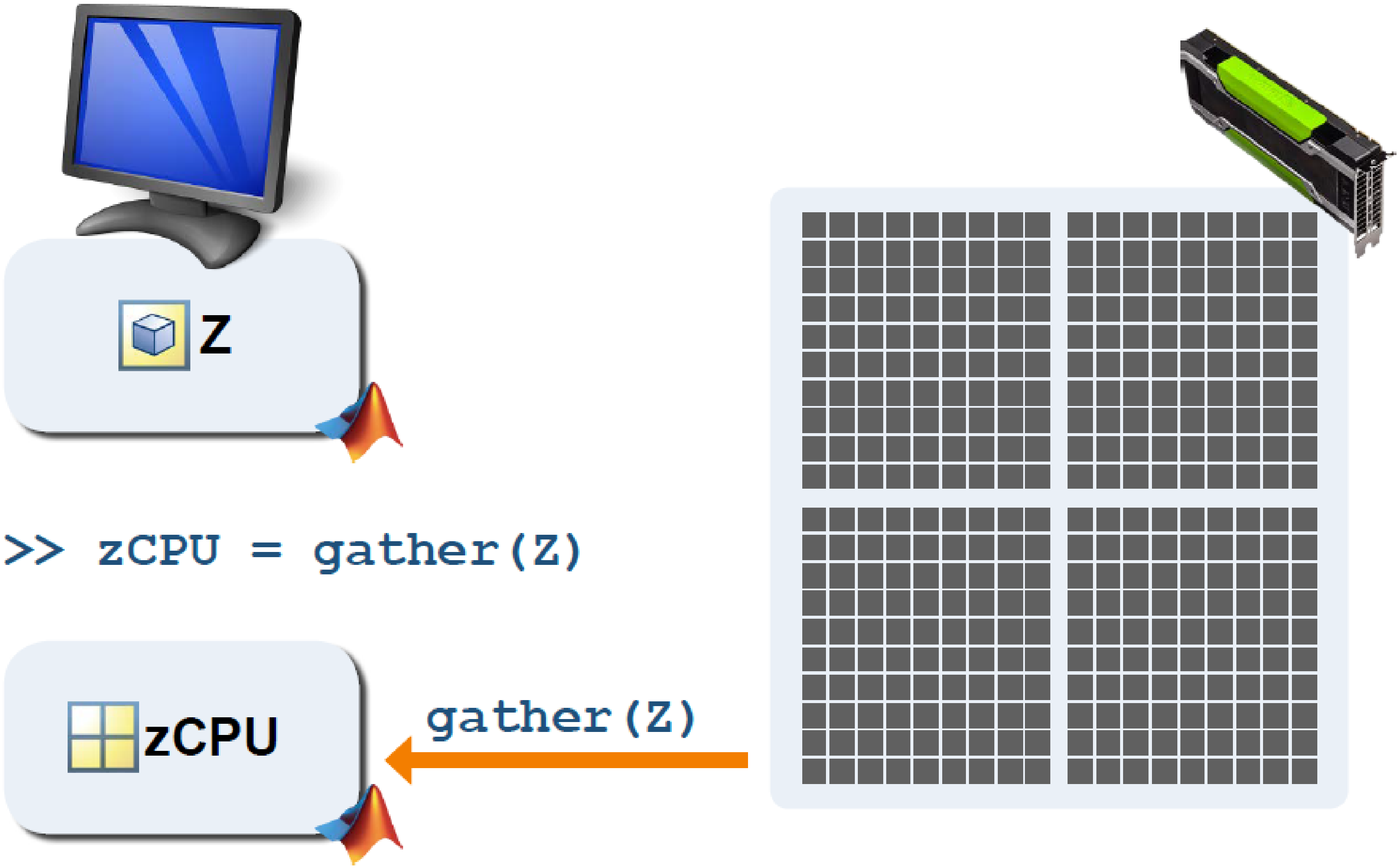


Built-In Functions That Support gpuArray

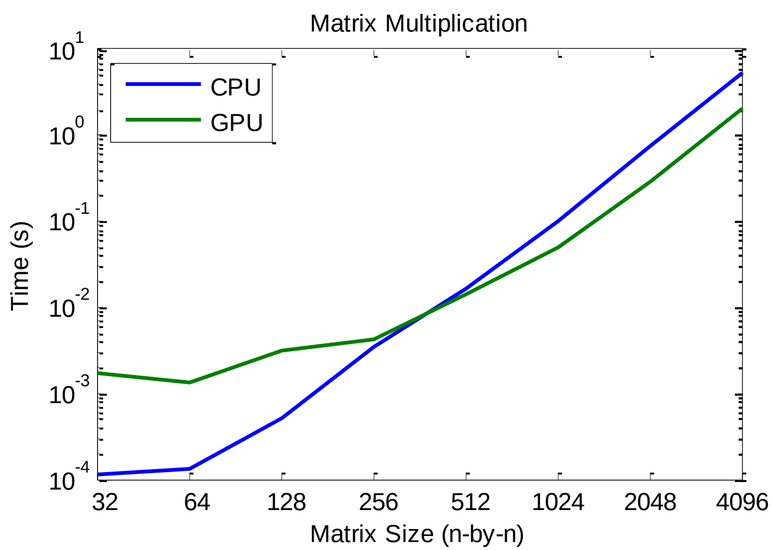
abs	compan	find	ismatrix	pagefun	spconvert
acos	complex	fix	ismember	pcg	sph2cart
acosd	cond	flip	ismembertol	perms	sprand
acosh	conj	fliplr	isnan	permute	sprandn
acot	conv	flipud	isnumeric	pinv	sprandsym
acotd	conv2	floor	isreal	planerot	sprintf
acoth	convn	fprintf	isrow	plot (and related)	sqrt
acsc	corrcoef	full	issorted	plus	squeeze
acscd	cos	gamma	issparse	pol2cart	std
acsch	cosd	gammainc	issymmetric	poly	sub2ind
accumarray	cosh	gammaincinv	istril	polyarea	subsasgn
all	cot	gammaln	istriu	polyder	subsindex
and	cotd	gather	isvector	polyfit	subspace
angle	coth	ge	kron	polyint	subsref
any	cov	gmres	ldivide	polyval	sum
arrayfun	cross	gradient	le	polyvalm	superiorfloat
asec	csc	gt	legendre	pow2	svd
asecd	cscd	hankel	length	power	swapbytes
asech	csch	histcounts	log	prod	tan
asin	ctranspose	horzcat	log10	psi	tand
asind	cummax	hsv2rgb	log1p	qmr	tanh
asinh	cummin	hypot	log2	qr	times
assert	cumprod	idivide	logical	rad2deg	toeplitz
atan	cumsum	ifft	lsqr	rand	trace
atan2	deg2rad	ifft2	lt	randi	transpose
atan2d	del2	ifftn	lu	randn	trapz
atand	det	ifftshift	mat2str	randperm	tril
atanh	detrend	imag	max	rank	triu
bandwidth	diag	ind2sub	median	rdivide	true
besselj	diff	Inf	mean	real	typecast
bessely	discretize	inpolygon	meshgrid	reallog	uint16
beta	disp	int16	min	realpow	uint32
betainc	display	int2str	minus	realsqrt	uint64
betaincinv	dot	int32	mldivide	rectint	uint8
betaln	double	int64	mod	rem	uminus
bicg	eig	int8	mode	repelem	union
bicgstab	eps	interp1	mpower	repmat	unique
bitand	eq	interp2	mrdivide	reshape	uniquetol
bitcmp	erf	interp3	mtimes	rgb2hsv	unwrap
bitget	erfc	interpn	NaN	roots	uplus
bitor	erfcinv	intersect	ndgrid	rot90	vander
bitset	erfcx	inv	ndims	round	var
bitshift	erfinv	ipermute	ne	sec	vertcat
bitxor	exp	isaUnderlying	nextpow2	secd	xor
blkdiag	expint	isbanded	nnz	sech	zeros
bsxfun	expm	iscolumn	nonzeros	setdiff	
cart2pol	expm1	isdiag	norm	setxor	
cart2sph	eye	isempty	normest	shiftdim	
cast	factorial	isequal	not	sign	
cat	false	isequaln	nthroot	sin	
cdf2rdf	fft	isfinite	null	sind	
ceil	fft2	isfloat	num2str	single	
chol	fftn	ishermitian	numel	sinh	
circshift	fftshift	isinf	ones	size	
classUnderlying	filter	isinteger	or	sort	
colon	filter2	islogical	orth	sortrows	

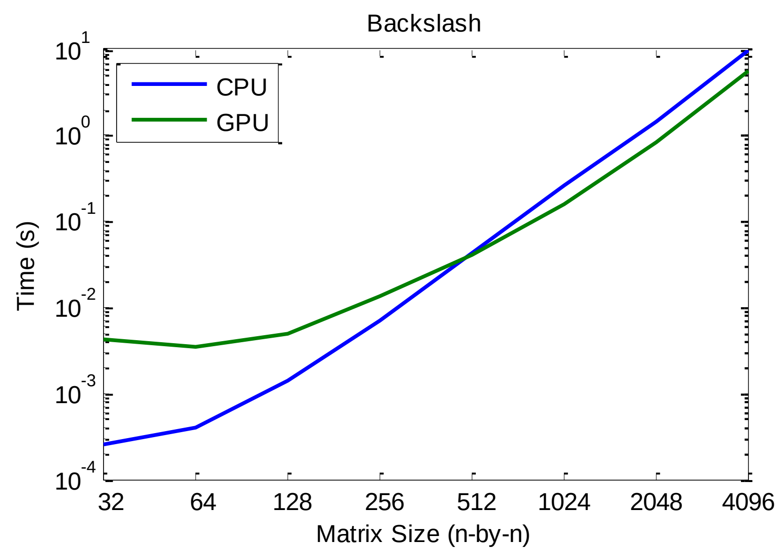
https://www.mathworks.com/help/distcomp/run-built-in-functions-on-a-gpu.html

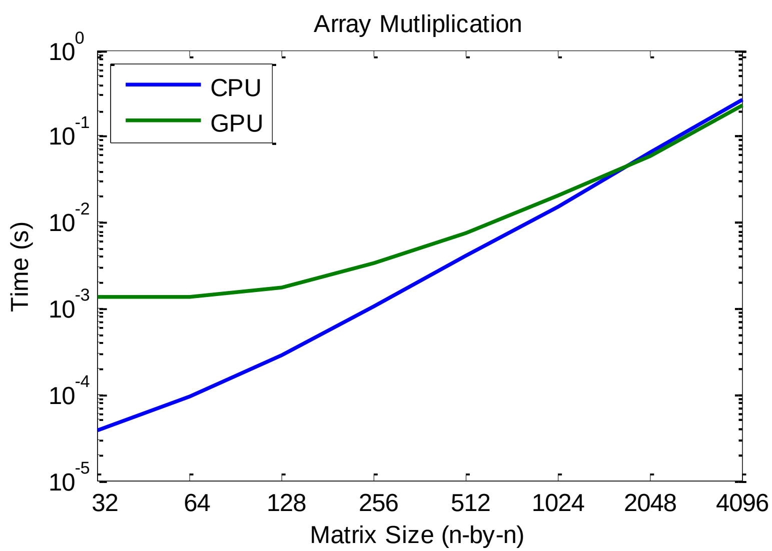
Retrieving Data to the CPU

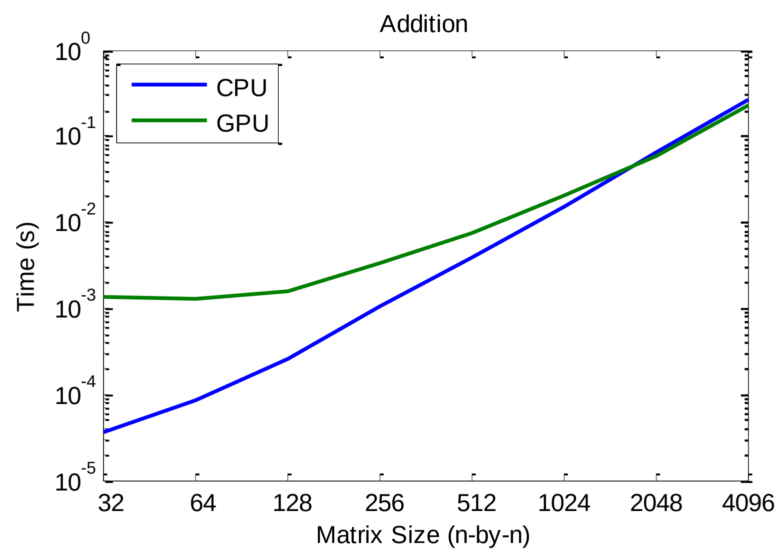


Summary: Invoking Built-In Functions









FFT on GPU

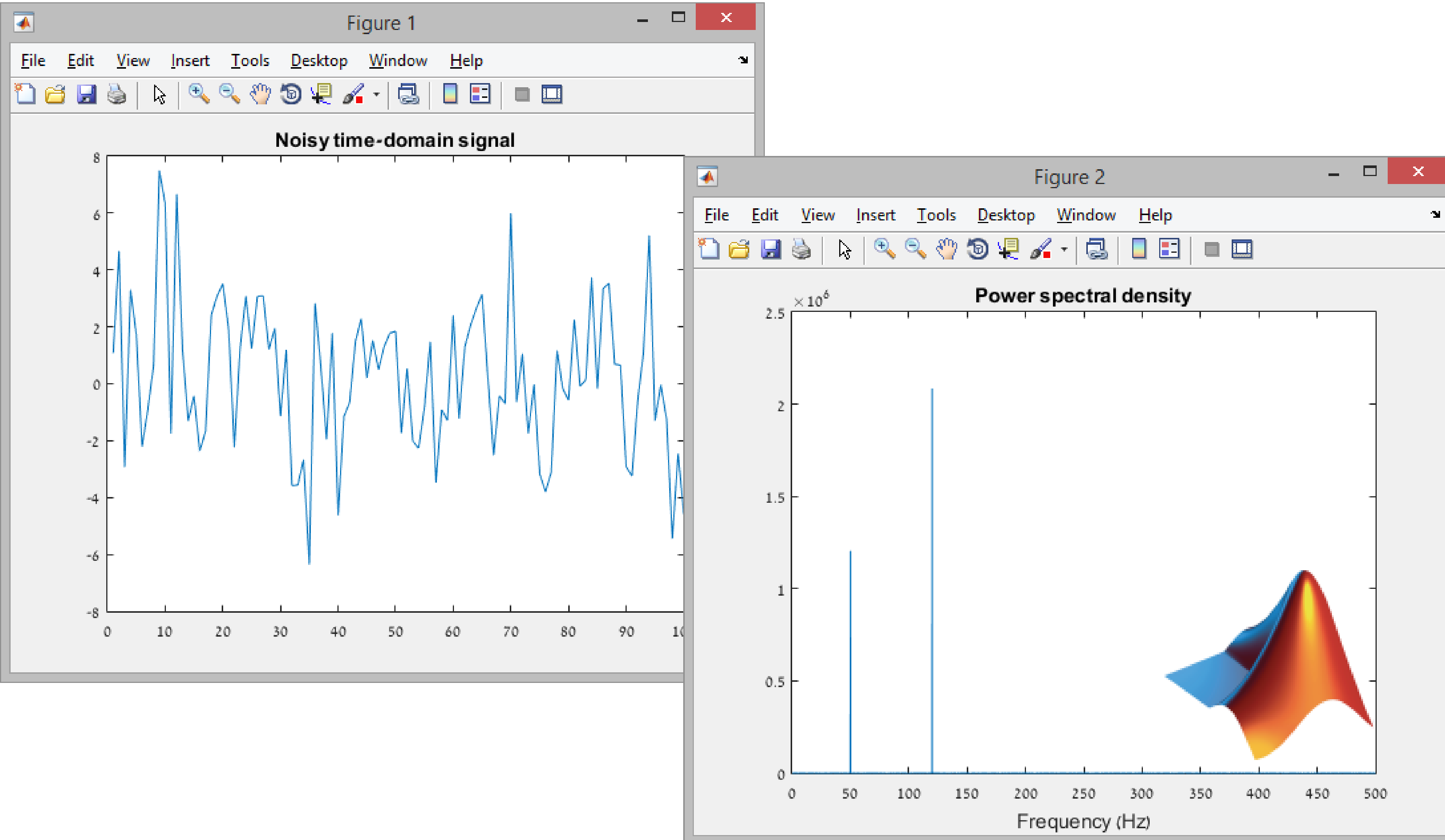


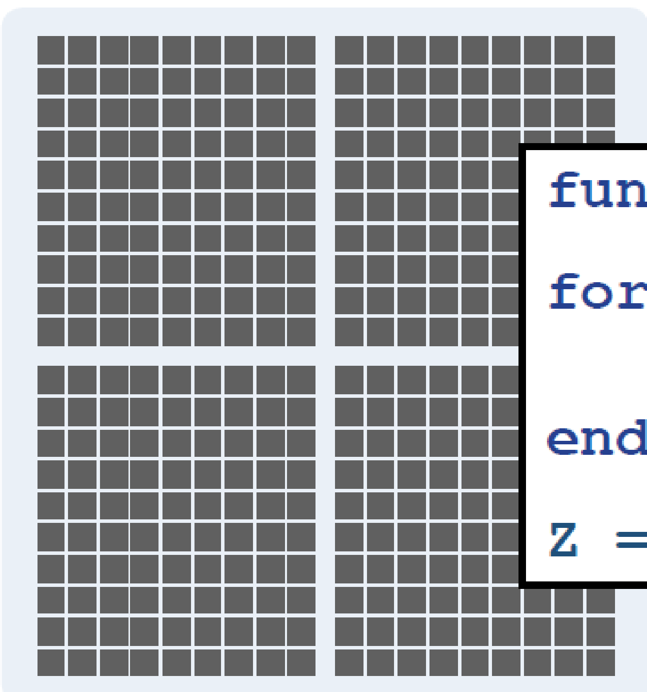
Figure 1
File Edit View Insert Tools Desktop Window Help
Noisy time-domain signal
8
6
4
2
0
-2
-4
-6
-8
0 10 20 30 40 50 60 70 80 90
Figure 2
File Edit View Insert Tools Desktop Window Help
Power spectral density
× 10^6
2.5
2
1.5
1
0.5
0
0 50 100 150 200 250 300 350 400 450 500
Frequency (Hz)

Image Processing & GPU

- GPU-enabled Image Processing Toolbox functions in R2013a: *imrotate, imfilter, imdilate, imerode, imopen, imclose, imtophat, imbothat, imshow, padarray and bwlookup*
- 22 more functions in R2013b, including: *edge, imresize, bwmorph and medfilt2*
- 9 more functions in R2014a, including: *bwdist, imreconstruct, iradon, radon and imfill*
- *bwlabel & imregdemons* in R2014b
- 5 more in R2015a, including *regionprops*
- 20 in R2015b, including *bwareaopen, hough, imrotate, imresize and imcrop* (69 total)

2. Performing Calculations

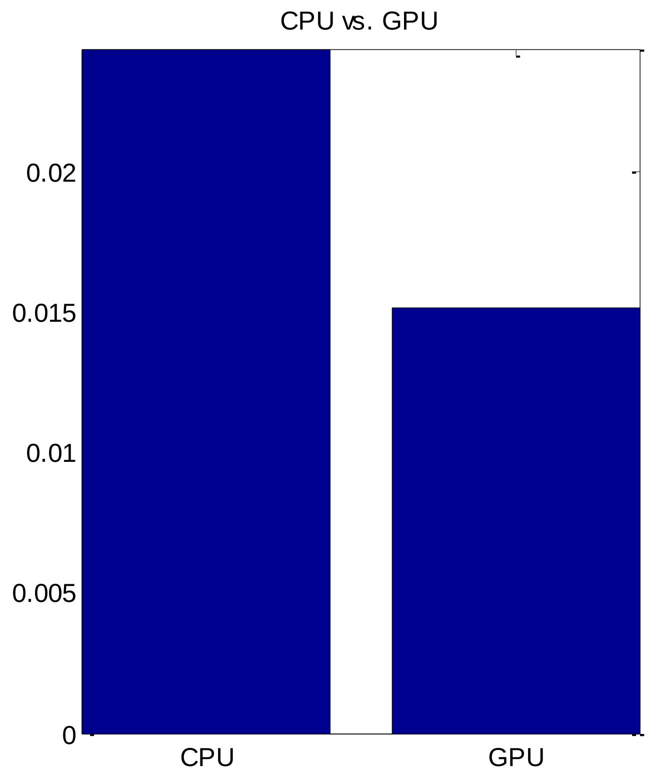
```
>> Z = arrayfun(@juliaCalc,Z,-0.8+0.156i)
```



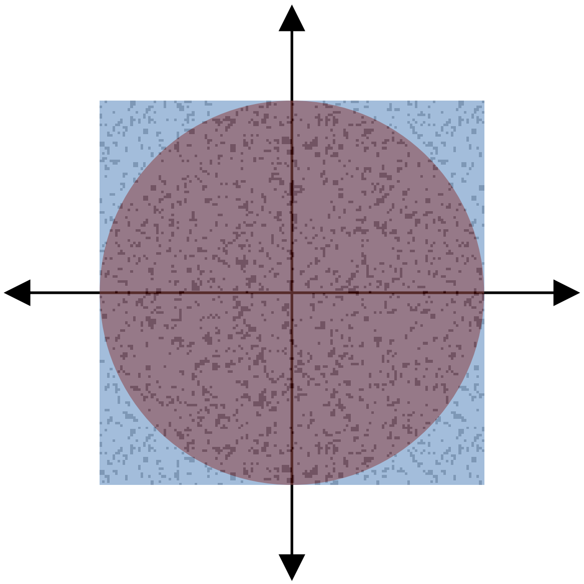
```
function Z = juliaCalc(Z,c)
for k = 1:20
    Z = Z.^2 + c;
end
Z = exp(-abs(Z));
```

Translating MATLAB Functions

- Accelerate scalar operations on large arrays
- Combine computations into a single kernel



3. Invoking a CUDA Kernel using feval



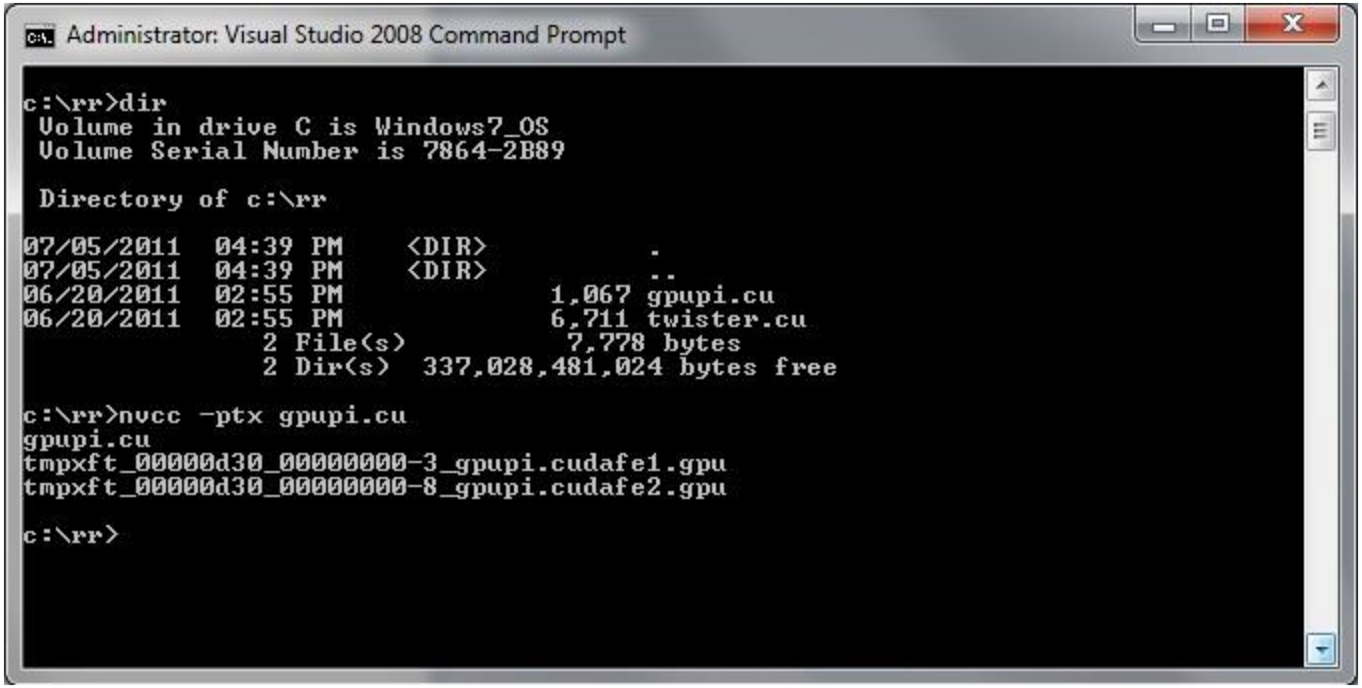
$$A_{circle} = \pi r^2$$

$$A_{square} = (2r)^2 = 4r^2$$

$$\frac{A_{circle}}{A_{square}} = \frac{\pi}{4}$$

$$\mathrm{P}(x^2 + y^2 < r^2) = \frac{\pi}{4}$$

Generating Parallel Thread Execution Files



Compile CUDA code to ptx using NVidia compiler

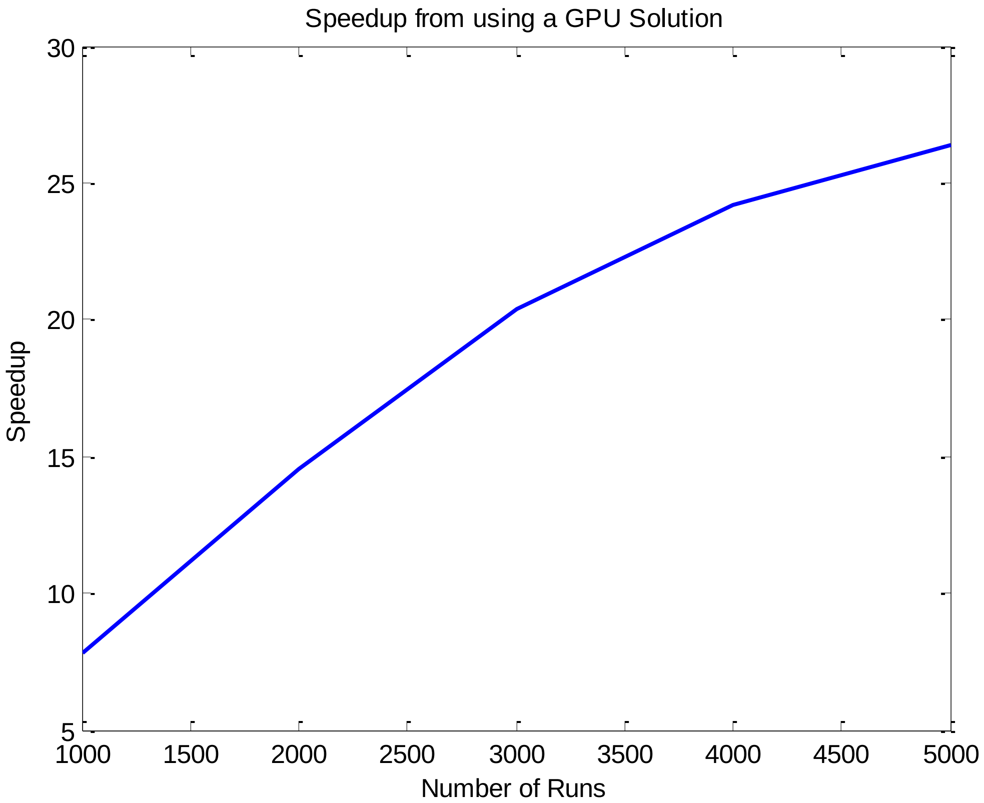
Summary: Invoking a CUDA Kernel

Pros

- Call existing CUDA code from MATLAB
- Highest level of functionality

Cons

- Requires knowledge of CUDA
- Requires knowledge of Parallel Programming



Summary of Features

Ease of Use

1) Invoking built-in functions on arrays existing on the GPU

2) Translating simple MATLAB functions to run on the GPU

3) Invoking CUDA code from MATLAB

Greater Functionality

2011-2016 Upgrades

- More **GPU-enabled** MATLAB functions & features, including capabilities from:

  - Statistics and Machine Learning Toolbox (>90)
  - Image Processing Toolbox (69)
  - Communications System Toolbox (11)
  - Signal Processing Toolbox (5)
  - Neural Network Toolbox
  - Phased Array System Toolbox

- MATLAB **Compiler** generated standalone executables and components now support applications that use the GPU

User Stories

NASA Langley Research Center Accelerates Acoustic Data Analysis with GPU Computing

The Challenge

Accelerate the analysis of sound recordings from wind tunnel tests of aircraft components

The Solution

Use MATLAB and Parallel Computing Toolbox to re-implement a legacy program for processing acoustic data, and cut processing time by running computationally intensive operations on a GPU

The Results

- Computations completed 40 times faster
- Algorithm GPU-enabled in 30 minutes
- Processing of test data accelerated

Wind tunnel test setup featuring the Hybrid Wing Body model (inverted), with 97-microphone phased array (top) and microphone tower (left).

More information

- Web:
  www.mathworks.com/discovery/matlab-gpu.html
- MATLAB CENTRAL (~900 results):
  http://www.mathworks.com/matlabcentral/
- MATLAB with Fun Blog:
  http://matlabisrael.blogspot.com/
- Linked in Group:
  MATLAB & Simulink users in Israel
- Seminars:
  http://www.systematics.co.il/products/mathworks/events/
- Courses:
  http://www.systematics.co.il/courses/mathworks/
- Webinars:
  http://www.mathworks.com/company/events/webinars/
- Support & Sales: 03-7660111

Questions?