MathWorks

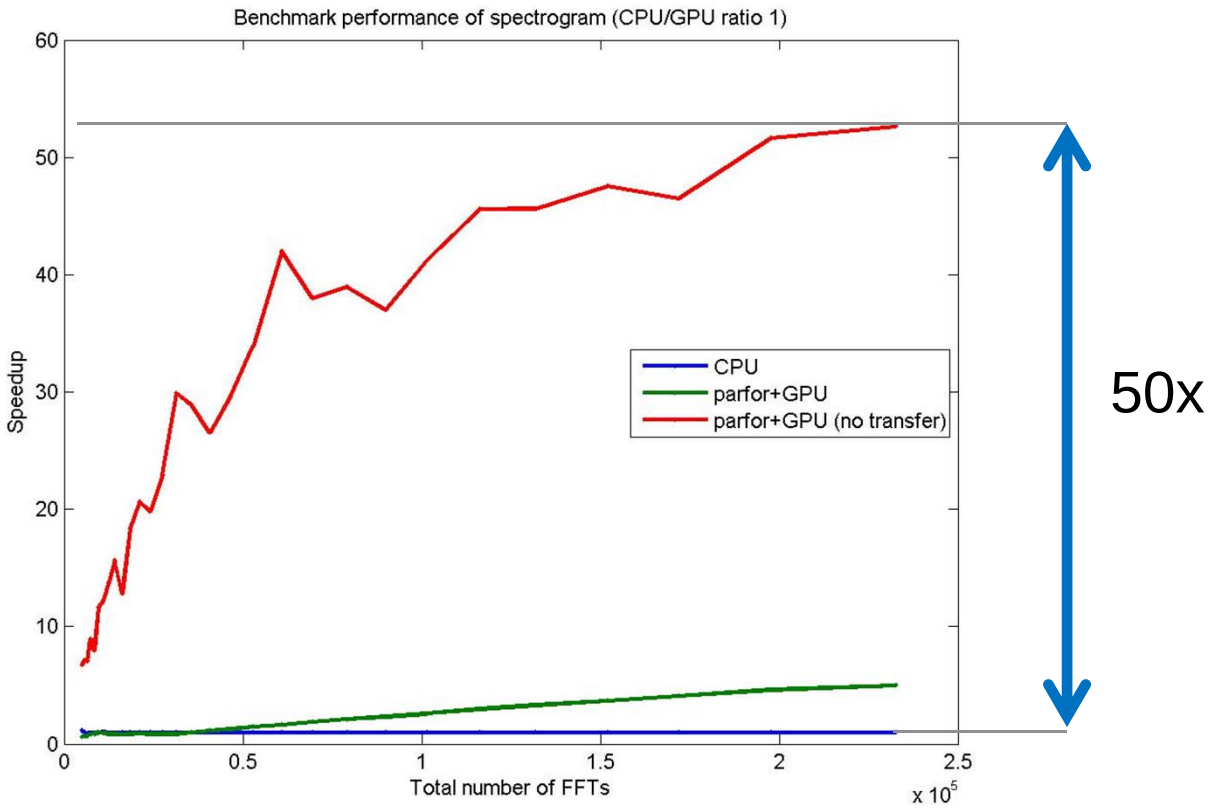
MATLAB&SIMULINK

GPU Computing with MATLAB

Loren Dean
Director of Engineering, MATLAB Products
MathWorks

© 2010 The MathWorks, Inc.

Spectrogram shows 50x speedup in a GPU cluster



Agenda

- Background
- Leveraging the desktop
  - Basic GPU capabilities
  - Multiple GPUs on a single machine
- Moving to the cluster
  - Multiple GPUs on multiple machines
- Q&A

How many people are using…

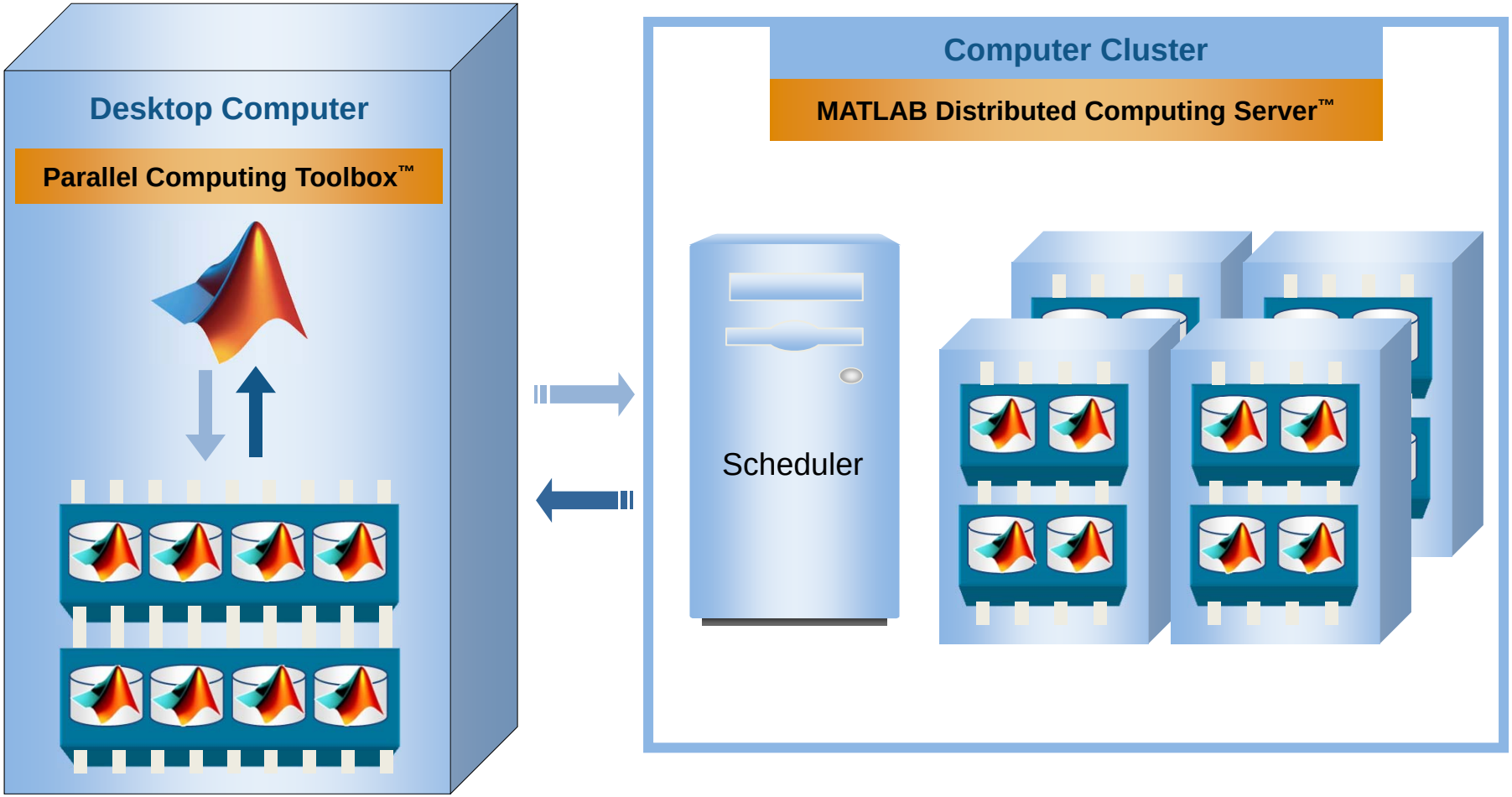
- MATLAB
- MATLAB with GPUs
- Parallel Computing Toolbox
  - R2010b prerelease
- MATLAB Distributed Computing Server

Why GPUs and why now?

- Operations are IEEE Compliant
- Cross-platform support now available
- Single/double performance inline with expectations

Parallel Computing with MATLAB

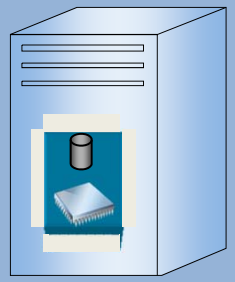
Tools and Terminology



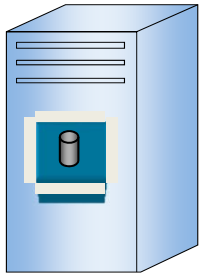
Parallel Capabilities

Task Parallel	Data Parallel	Environment
Built-in support with Simulink, toolboxes, and blocksets		`matlabpool` Local workers
`parfor`	distributed array >200 functions	Configurations `batch` MathWorks job manager
`job/task`	`spmd` co-distributed array MPI interface	third-party schedulers `job/task`

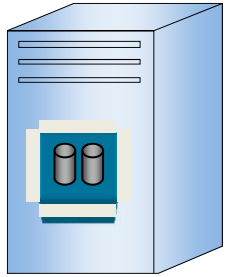
Evolving With Technology Changes



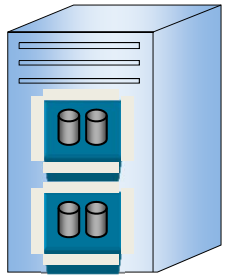
GPU



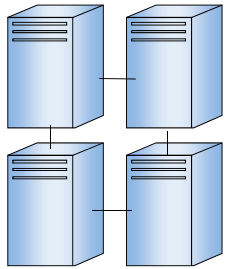
Single processor



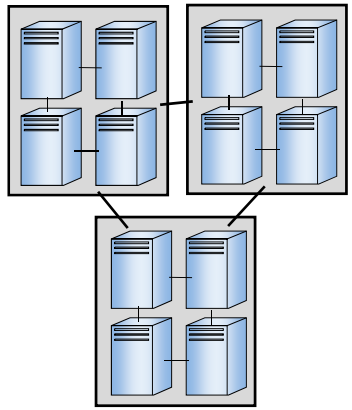
Multicore



Multiprocessor



Cluster



Grid,
Cloud

What's new in R2010b?

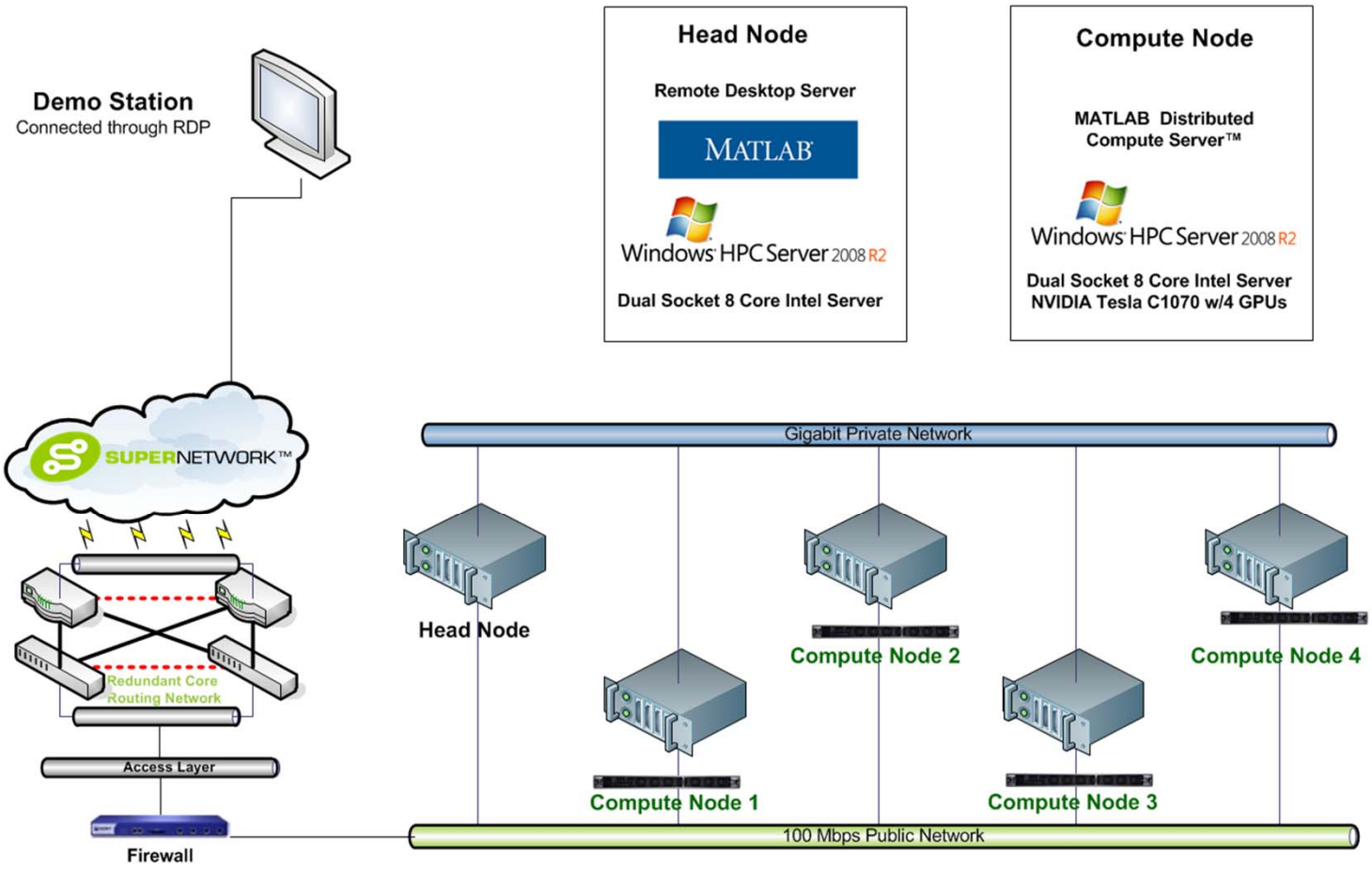
- Parallel Computing Toolbox
  - GPU support
  - Broader algorithm support (QR, rectangular \)
- MATLAB Distributed Computing Server
  - GPU support
  - Run as user with MathWorks job manager
  - Non-shared file system support
- Simulink®
  - Real-Time Workshop® support with PCT and MDCS

GPU Functionality

- Call GPU(s) from MATLAB or toolbox/server worker
- Support for CUDA 1.3 enabled devices
- 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

Demo hardware

MathWorks Microsoft peer1 hosting



Example:

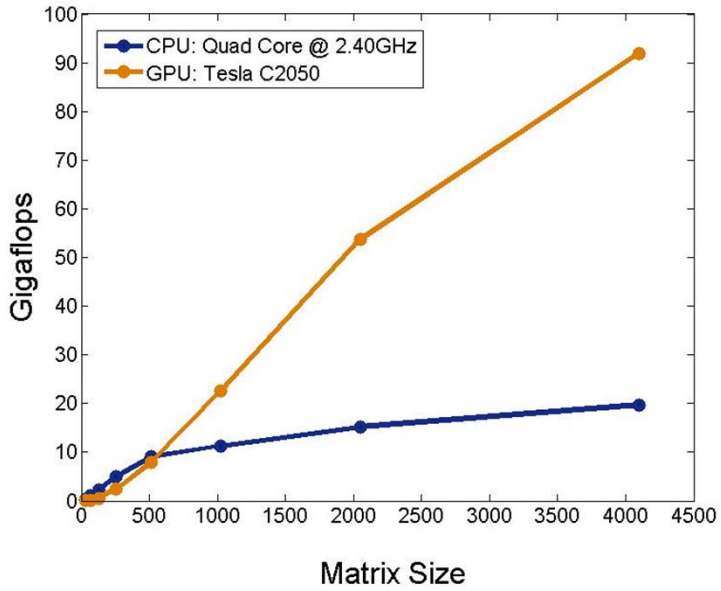
GPU Arrays

```
>> A = someArray(1000, 1000);
>> G = gpuArray(A); % Push to GPU memory
…
>> F = fft(G);
>> x = G\b;
…
>> z = gather(x); % Bring back into MATLAB
```

GPUArray Function Support

- >100 functions supported
  - `fft, fft2, ifft, ifft2`
  - Matrix multiplication (`A*B`)
  - Matrix left division (`A\b`)
  - LU factorization
  - `'  .'`
  - `abs, acos, …, minus, …, plus, …, sin, …`
- Key functions not supported
  - `conv, conv2, filter`
  - indexing

GPU Array benchmarks



A\b*	Tesla C1060	Tesla C2050 (Fermi)	Quad-core Intel CPU	Ratio (Fermi:CPU)
Single	191	250	48	5:1
Double	63.1	128	25	5:1
Ratio	3:1	2:1	2:1	

* Results in Gflops, matrix size 8192x8192. Limited by card memory.
Computational capabilities not saturated.

GPU Array benchmarks

MTIMES	Tesla C1060	Tesla C2050 (Fermi)	Quad-core Intel CPU	Ratio (Fermi:CPU)
Single	365	409	59	7:1
Double	75	175	29	6:1
Ratio	4.8:1	2.3:1	2:1	

FFT	Tesla C1060	Tesla C2050 (Fermi)	Quad-core Intel CPU	Ratio (Fermi:CPU)
Single	50	99	2.29	43:1
Double	22.5	44	1.47	30:1
Ratio	2.2:1	2.2:1	1.5:1	

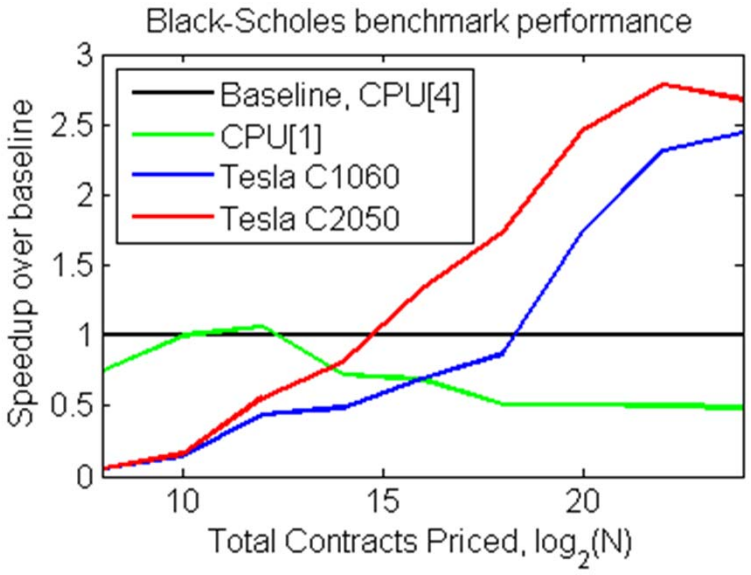
Example:

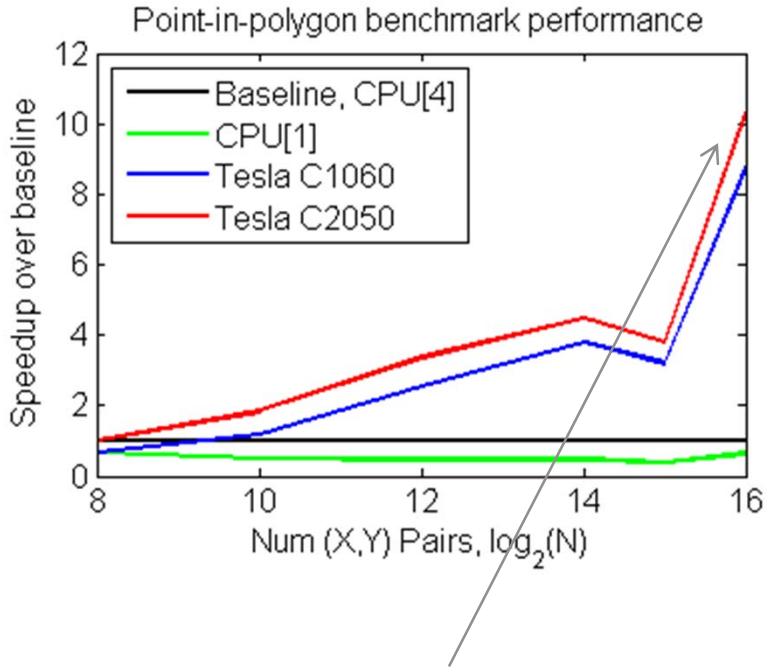
arrayfun: Element-Wise Operations

```
>> y = arrayfun(@foo, x); % Execute on GPU
```

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

Some `arrayfun` benchmarks





Note: Due to memory constraints, a different approach is used at N=16 and above.

CPU[4] = multhithreading enabled
CPU[1] = multhithreading disabled

Example:

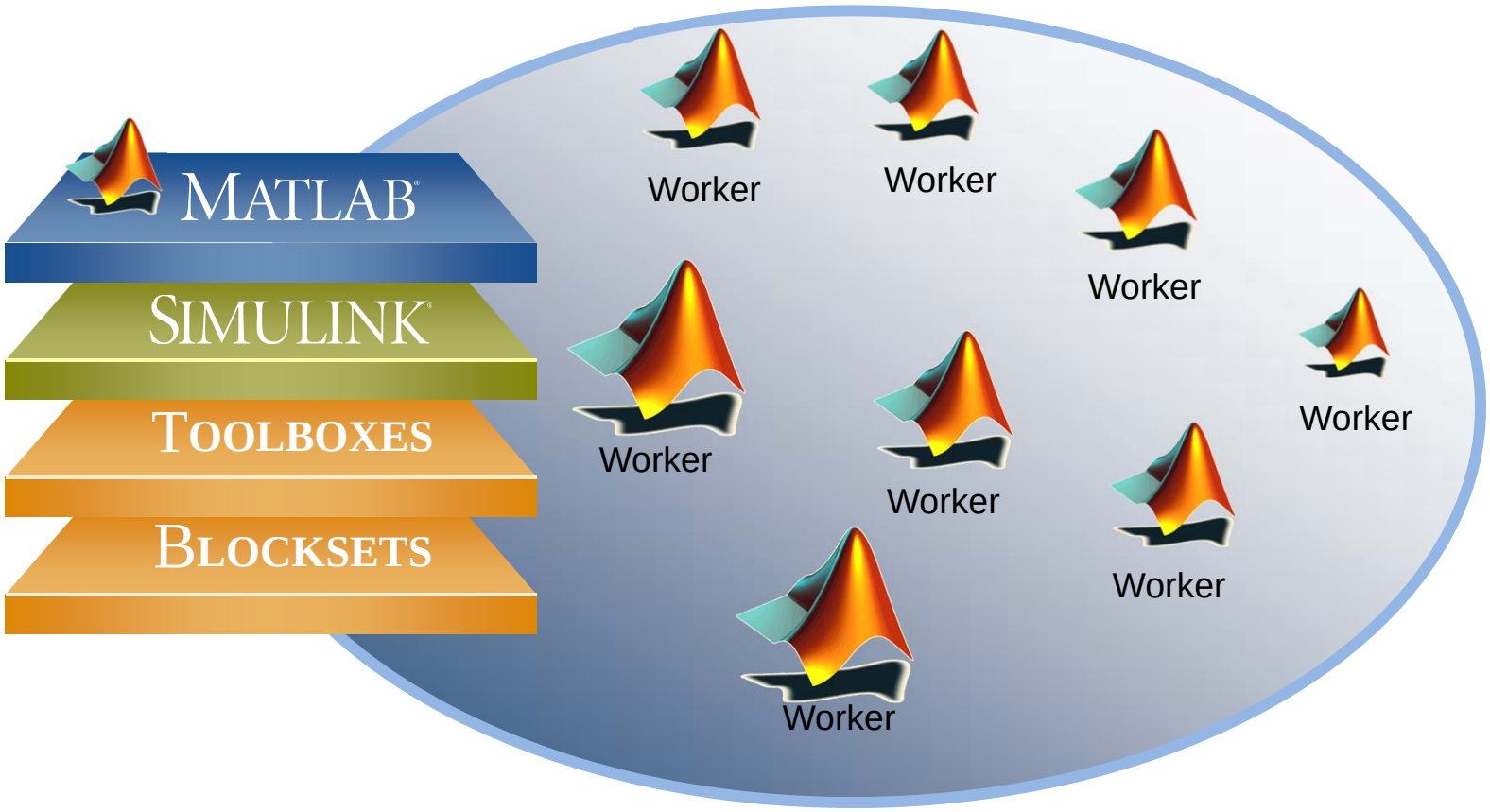
Invoking CUDA Kernels

```
% Setup
kern = parallel.gpu.CUDAKernel('myKern.ptx', cFcnSig)

% Configure
kern.ThreadBlockSize=[512 1];
kern.GridSize=[1024 1024];

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

Options for scaling up

- Leverage matlabpool
  - Enables desktop or cluster cleanly
  - Can be done either interactively or in batch
- Decide how to manage your data
  - Task parallel
    - Use parfor
    - Same operation with different inputs
    - No interdependencies between operations
  - Data parallel
    - Use spmd
    - Allows for interdependency between operations at the CPU level

MATLAB Pool Extends Desktop MATLAB



Example:

Spectrogram on the desktop
(CPU only)

```
D = data;
iterations = 2000; % # of parallel iterations
stride = iterations*step; %stride of outer loop

M = ceil((numel(x)-W)/stride);%iterations needed
o = cell(M, 1); % preallocate output

for i = 1:M
    % What are the start points
    thisSP = (i-1)*stride:step: …
        (min(numel(x)-W, i*stride)-1);

    % Move the data efficiently into a matrix
    X = copyAndWindowInput(D, window, thisSP);

    % Take lots of fft's down the colmuns
    X = abs(fft(X));

    % Return only the first part to MATLAB
    o{i} = X(1:E, 1:ratio:end);

end
```

Example:

Spectrogram on the desktop
(CPU to GPU)

```
D = data;
iterations = 2000; % # of parallel iterations
stride = iterations*step; %stride of outer loop

M = ceil((numel(x)-W)/stride);%iterations needed
o = cell(M, 1); % preallocate output

for i = 1:M
    % What are the start points
    thisSP = (i-1)*stride:step: …
        (min(numel(x)-W, i*stride)-1);

    % Move the data efficiently into a matrix
    X = copyAndWindowInput(D, window, thisSP);

    % Take lots of fft's down the colmuns
    X = abs(fft(X));

    % Return only the first part to MATLAB
    o{i} = X(1:E, 1:ratio:end);

end
```

```
D = gpuArray(data);
iterations = 2000; % # of parallel iterations
stride = iterations*step; %stride of outer loop

M = ceil((numel(x)-W)/stride);%iterations needed
o = cell(M, 1); % preallocate output

for i = 1:M
    % What are the start points
    thisSP = (i-1)*stride:step: ...
        (min(numel(D)-W, i*stride)-1);

    % Move the data efficiently into a matrix
    X = copyAndWindowInput(D, window, thisSP);

    % Take lots of fft's down the colmuns
    X = gather(abs(fft(X)));

    % Return only the first part to MATLAB
    o{i} = X(1:E, 1:ratio:end);

end
```

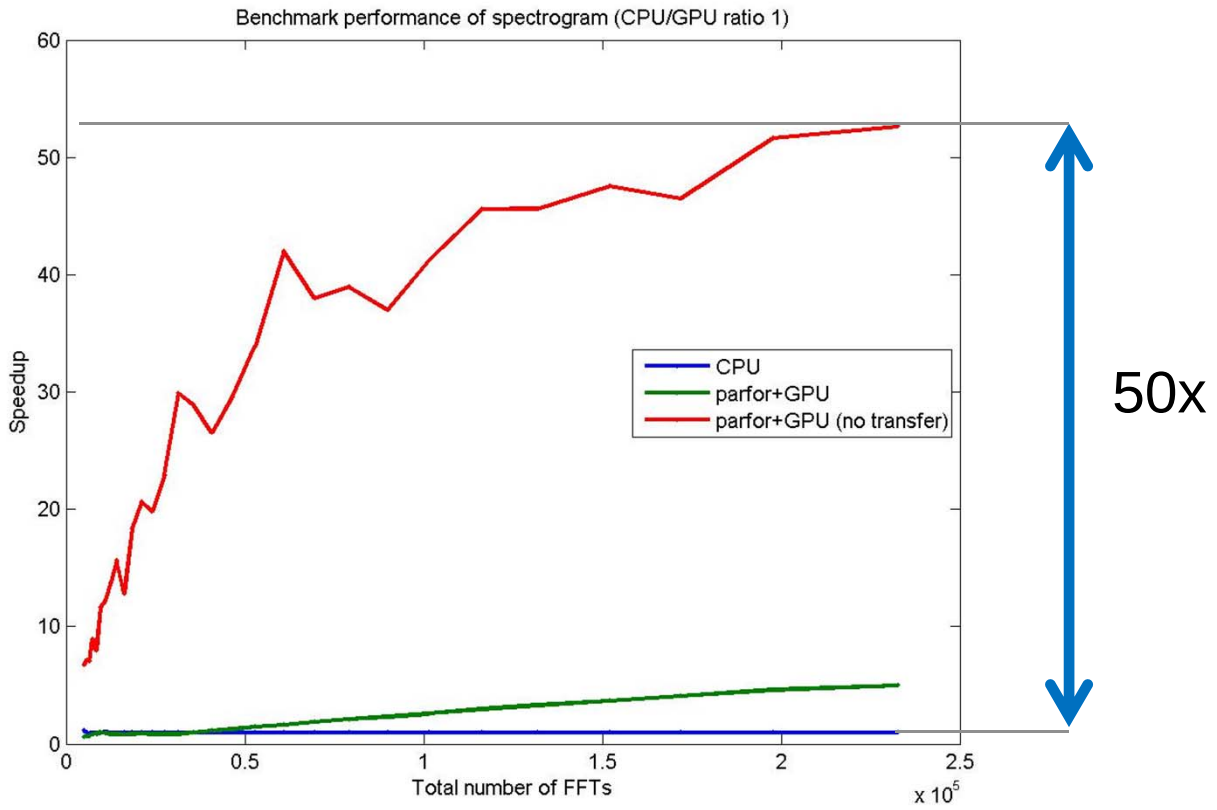
Example:

Spectrogram on the desktop
(GPU to parallel GPU)

```
D = gpuArray(data);
iterations = 2000; % # of parallel iterations
stride = iterations*step; %stride of outer loop

M = ceil((numel(x)-W)/stride);%iterations needed
o = cell(M, 1); % preallocate output

for i = 1:M
    % What are the start points
    thisSP = (i-1)*stride:step: ...
        (min(numel(D)-W, i*stride)-1);

    % Move the data efficiently into a matrix
    X = copyAndWindowInput(D, window, thisSP);

    % Take lots of fft's down the colmuns
    X = gather(abs(fft(X)));

    % Return only the first part to MATLAB
    o{i} = X(1:E, 1:ratio:end);

end
```

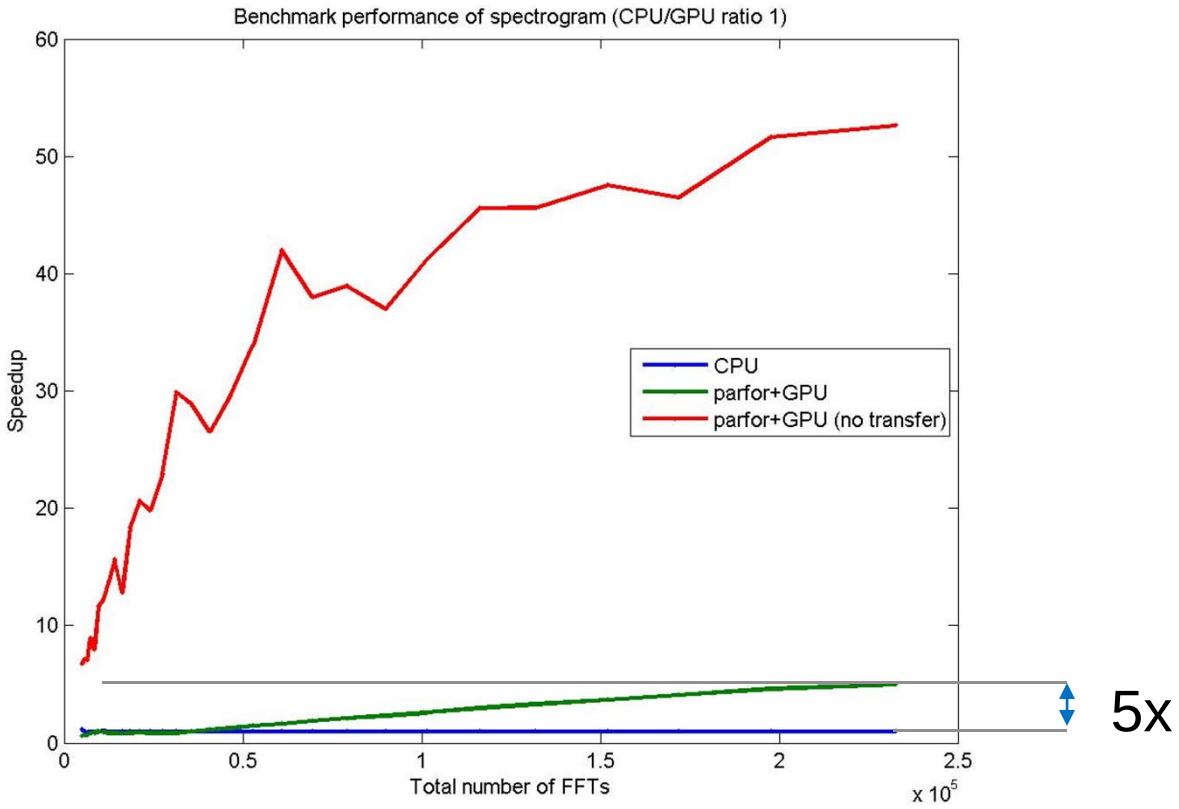
```
D = gpuArray(data);
iterations = 2000; % # of parallel iterations
stride = iterations*step; %stride of outer loop

M = ceil((numel(x)-W)/stride);%iterations needed
o = cell(M, 1); % preallocate output

parfor i = 1:M
    % What are the start points
    thisSP = (i-1)*stride:step: ...
        (min(numel(D)-W, i*stride)-1);

    % Move the data efficiently into a matrix
    X = copyAndWindowInput(D, window, thisSP);

    % Take lots of fft's down the colmuns
    X = gather(abs(fft(X)));

    % Return only the first part to MATLAB
    o{i} = X(1:E, 1:ratio:end);

end
```

Spectrogram shows 50x speedup in a GPU cluster

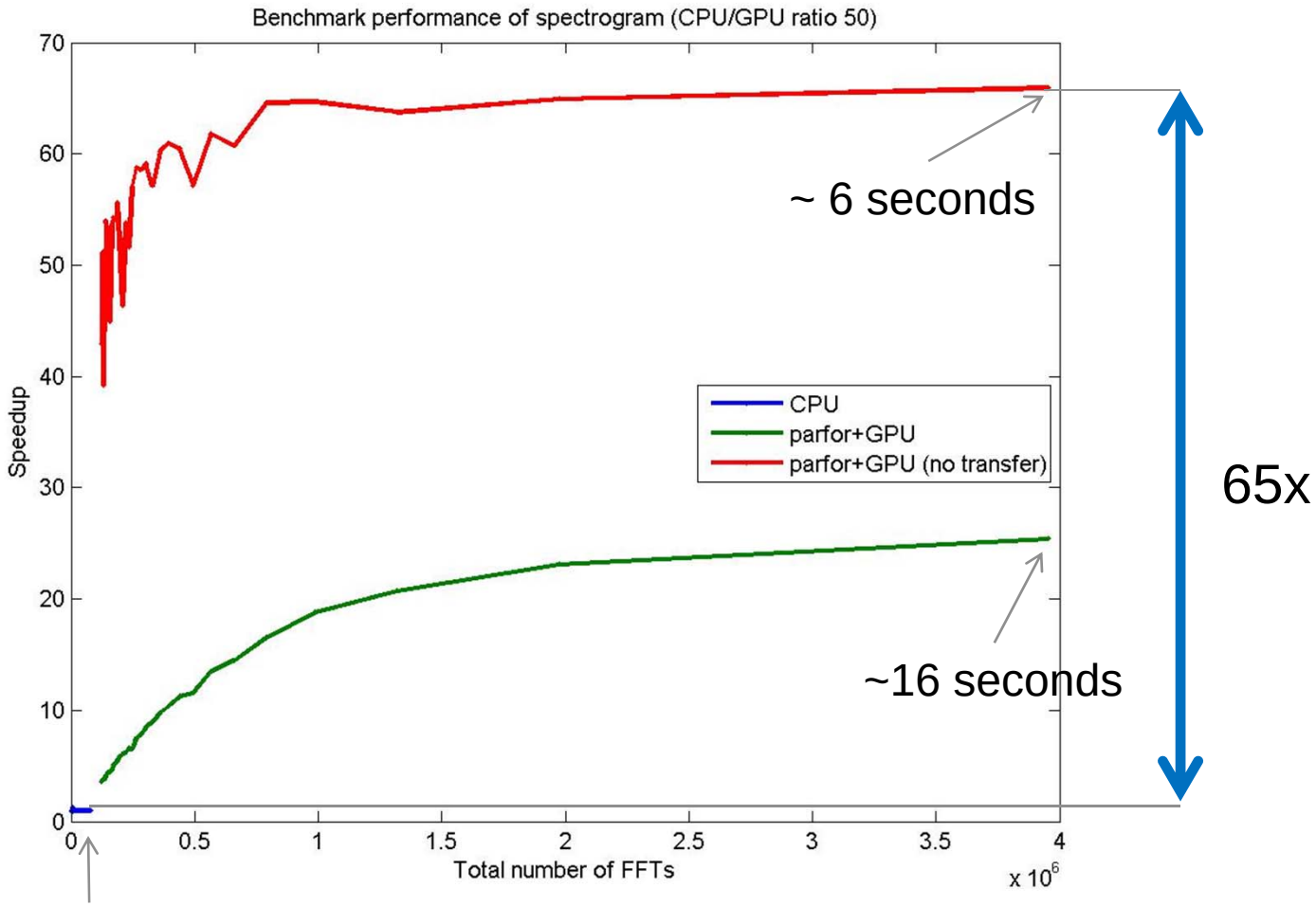


But…Speedup is 5x with data transfer

(Remember we have to transfer data of GigE and then the PCIe bus)



GPU can go well beyond the CPU



CPU ~ 10-15 seconds

Summary of Options for Targeting GPUs

Across one or more GPUs
on one or more machines:

1) Use GPU array interface with MATLAB built-in functions

2) Execute custom functions on elements of the GPU array

3) Create kernels from existing CUDA code and PTX files

What hardware is supported?

- NVIDIA hardware meeting the CUDA 1.3 hardware spec.
- A listing can be found at: http://www.nvidia.com/object/cuda_gpus.html

How come *`function_xyz`* is not GPU-accelerated?

- The accelerated functions available in this first release were gated by available resources.
- We will add capabilities with coming releases based on requirements and feedback.

Why did we adopt CUDA and not OpenCL?

- CUDA has the only ecosystem with all of the libraries necessary for technical computing

Why are CUDA 1.1 and CUDA 1.2 not supported?

As mentioned earlier, CUDA 1.3 offers the following capabilities that earlier releases of CUDA do not

- Support for doubles. The base data type in MATLAB is double.
- IEEE compliance. We want to insure we get the correct answer.
- Cross-platform support.

What benchmarks are available?

- Some benchmarks are available in the product and at www.mathworks.com/products/parallel-computing/
- More will be added over time