GPU Computing with MATLAB

Application Engineer
엄 준 상

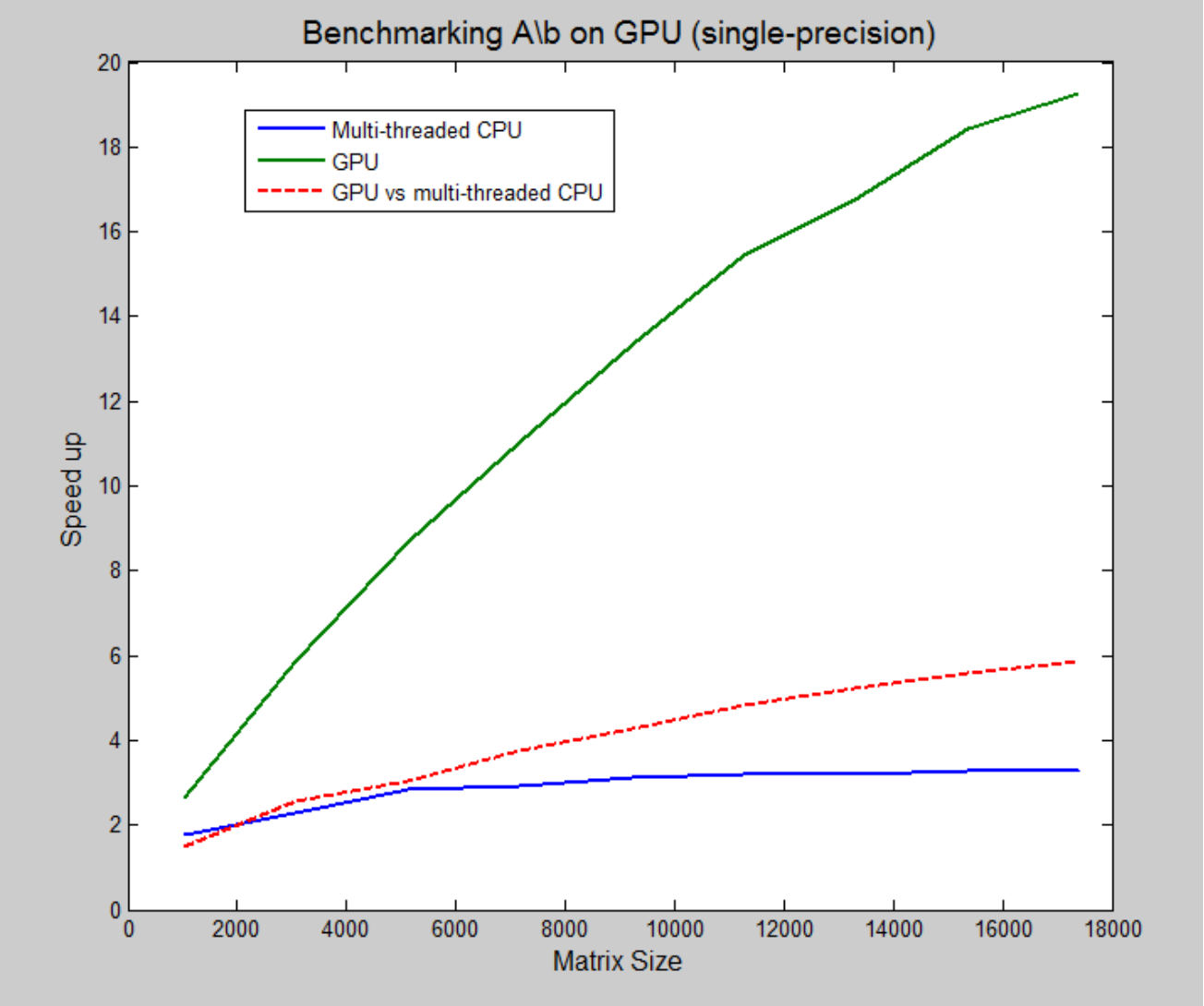
Agenda

- GPU background
- Introduction to parallel computing products
- GPU computing with MATLAB
- Sharing GPU applications

Graphics Processing Units (GPUs)

- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory

Benchmarking A\b on the GPU

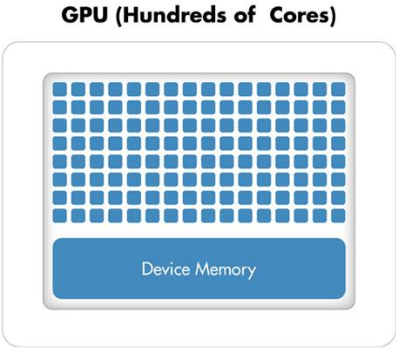


Common Terms Used in GPU Computing

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

Criteria for Good Problems to Run on a GPU

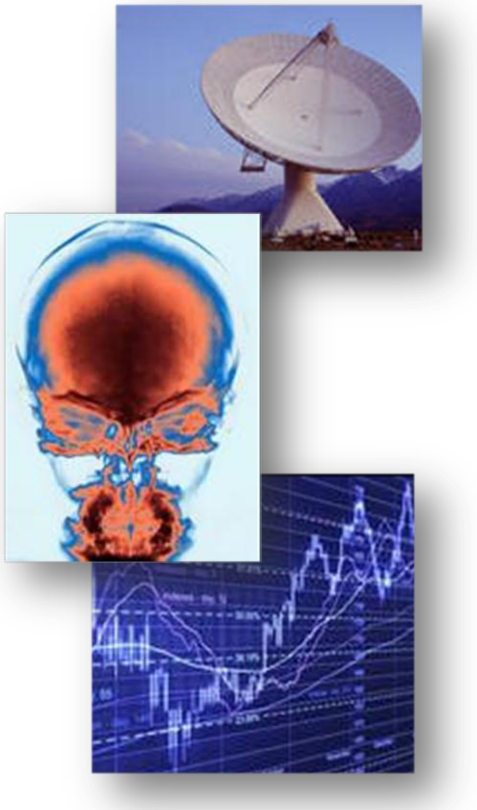
- **Massively parallel:**
  - Able to break down calculations into hundreds or thousands of independent units of work
  - Motivation: Best performance when hundreds of GPU cores are kept busy



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

General Purpose GPU Areas

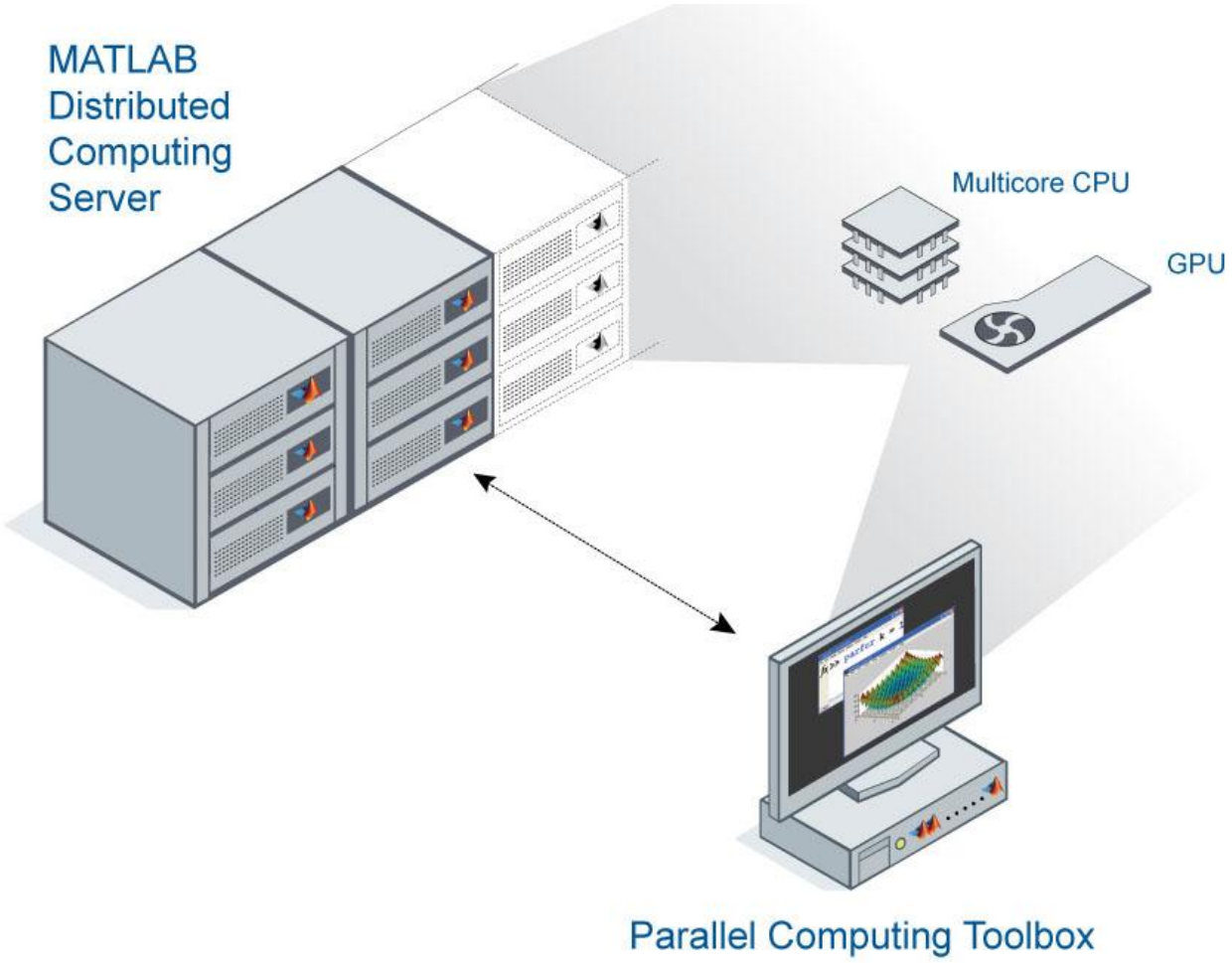
- Signal processing
- Image and video processing
- Computational chemistry
- Bioinformatics and medical imaging
- Computational finance
- Energy production



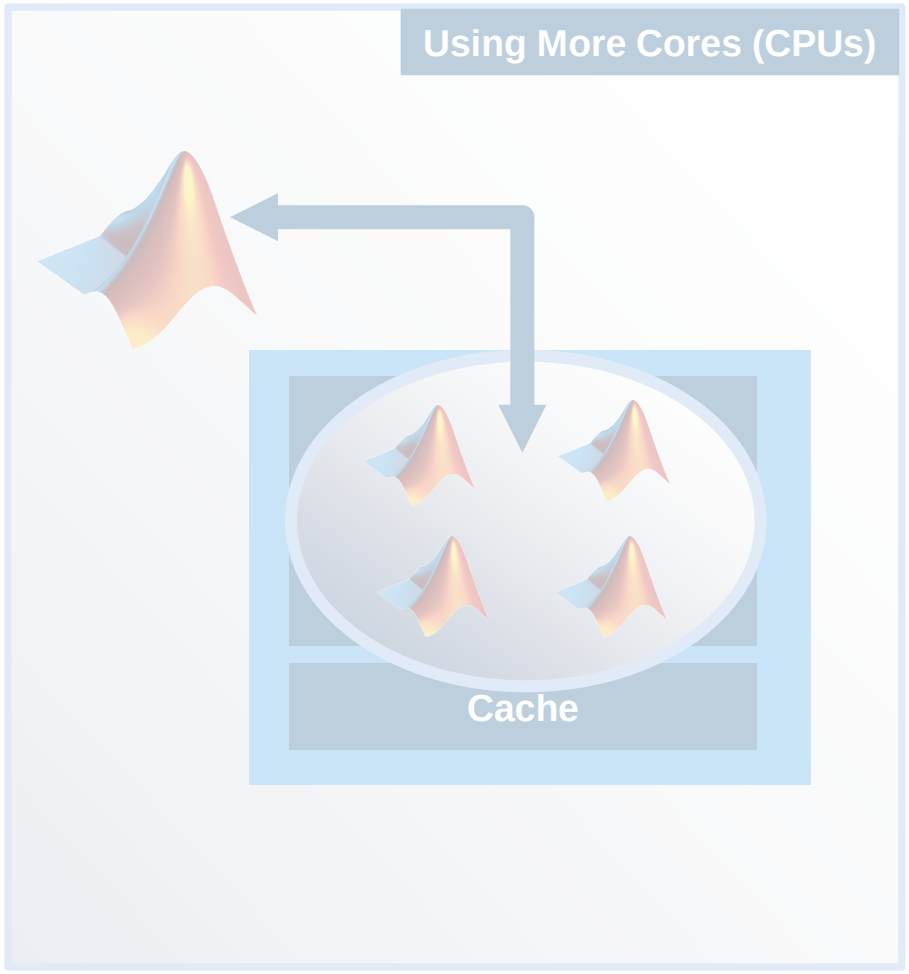
Agenda

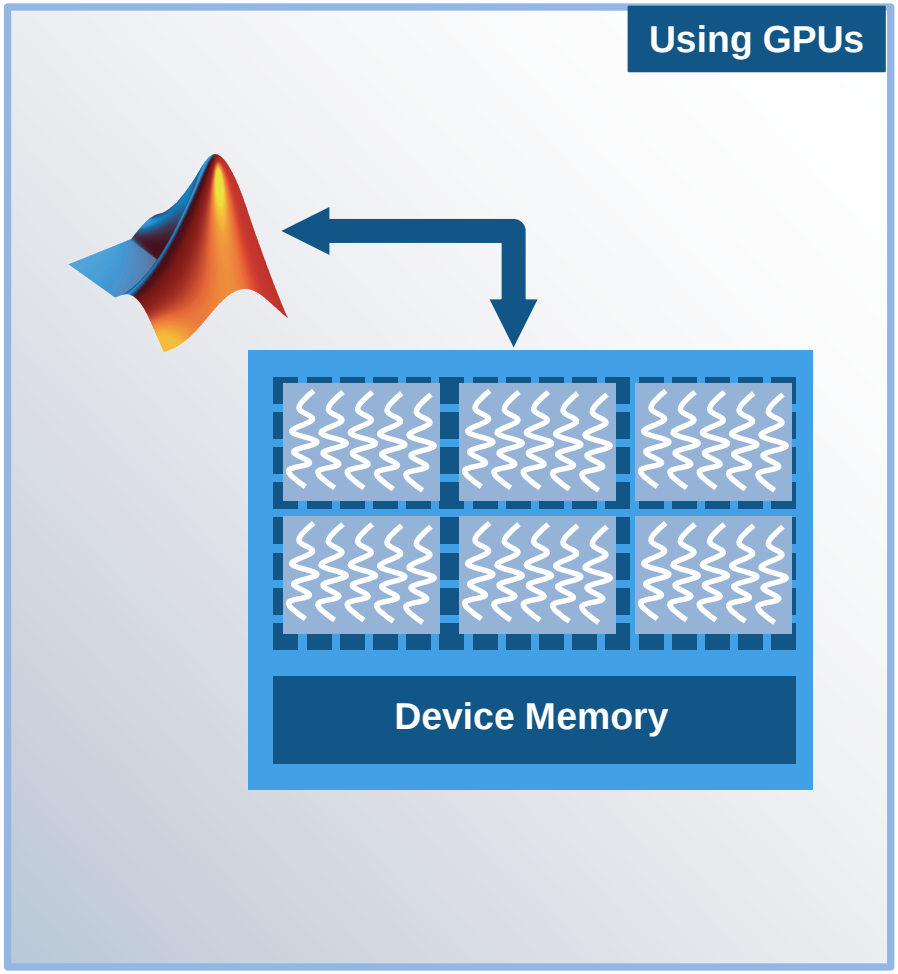
- GPU background
- Introduction to parallel computing products
- GPU computing with MATLAB
- Sharing GPU applications

Parallel Computing Products



Performance Gain with More Hardware





GPU Support with Parallel Computing Toolbox

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

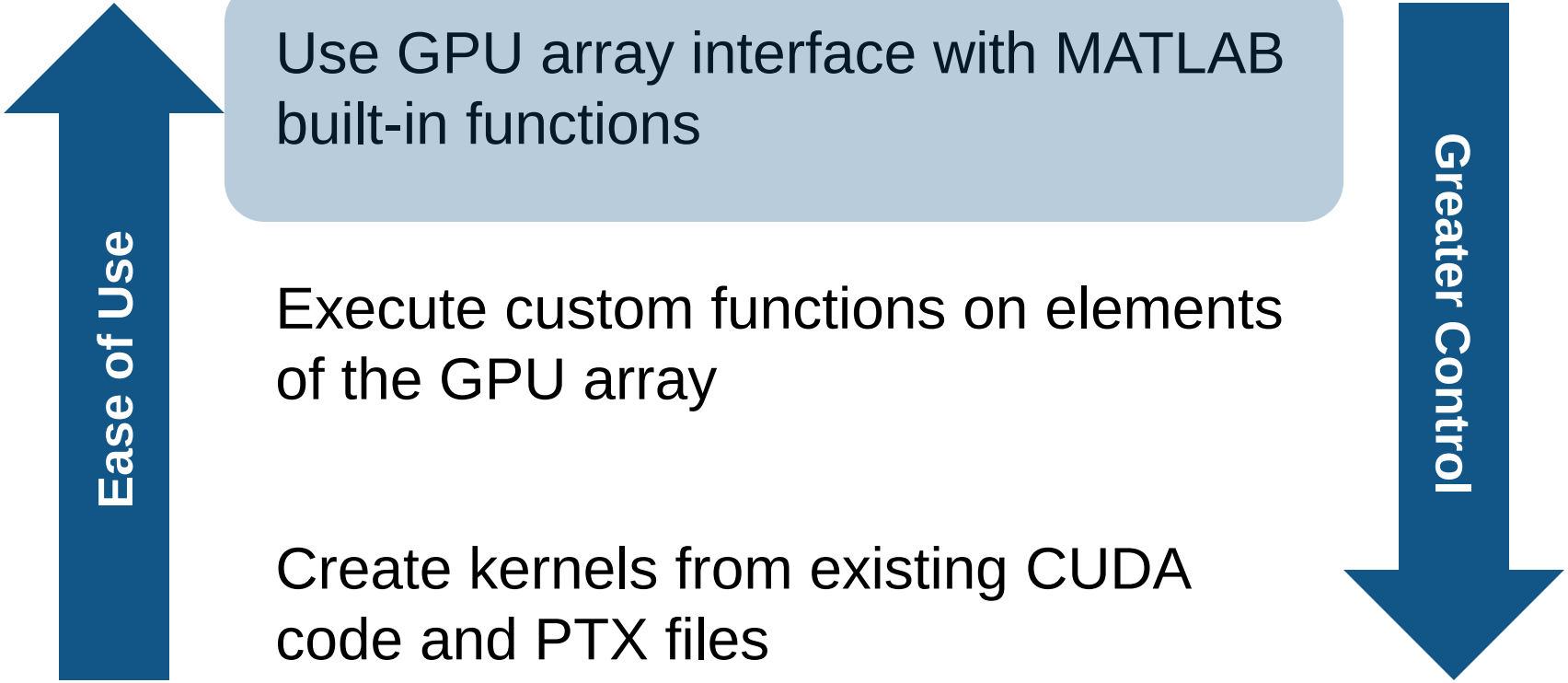
- Why we require compute capability 1.3
  - Support doubles (base data type in MATLAB)
  - Guarantee IEEE compliance
  - Provide cross-platform support

Evolving rapidly - Use latest release

Agenda

- GPU background
- Introduction to parallel computing products
- GPU computing with MATLAB
- Sharing GPU applications

Options for Targeting GPUs



Overloaded MATLAB Functions

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

Full list of built-in functions that support GPUArray

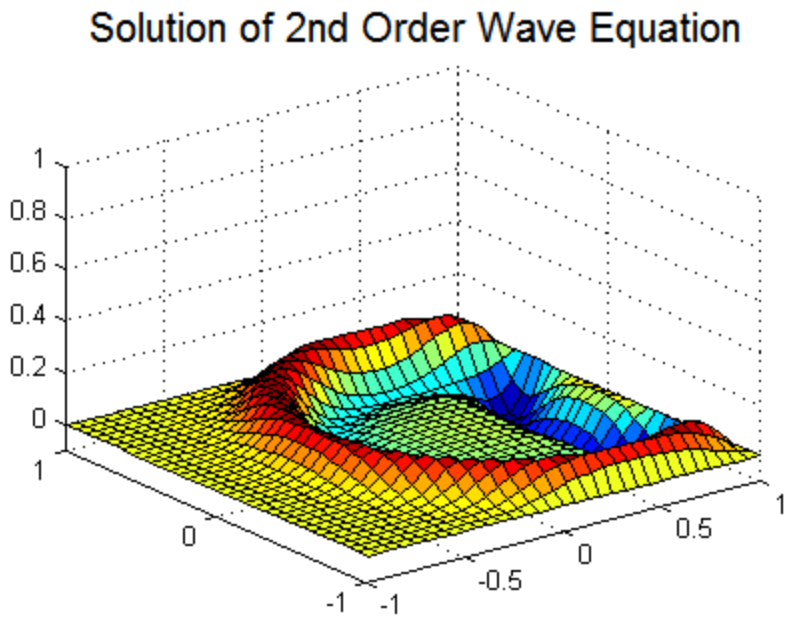
User's Guide → GPU Computing → Using GPUArray

Example: Solving 2D Wave Equation

- Solve 2nd order wave equation using spectral methods:

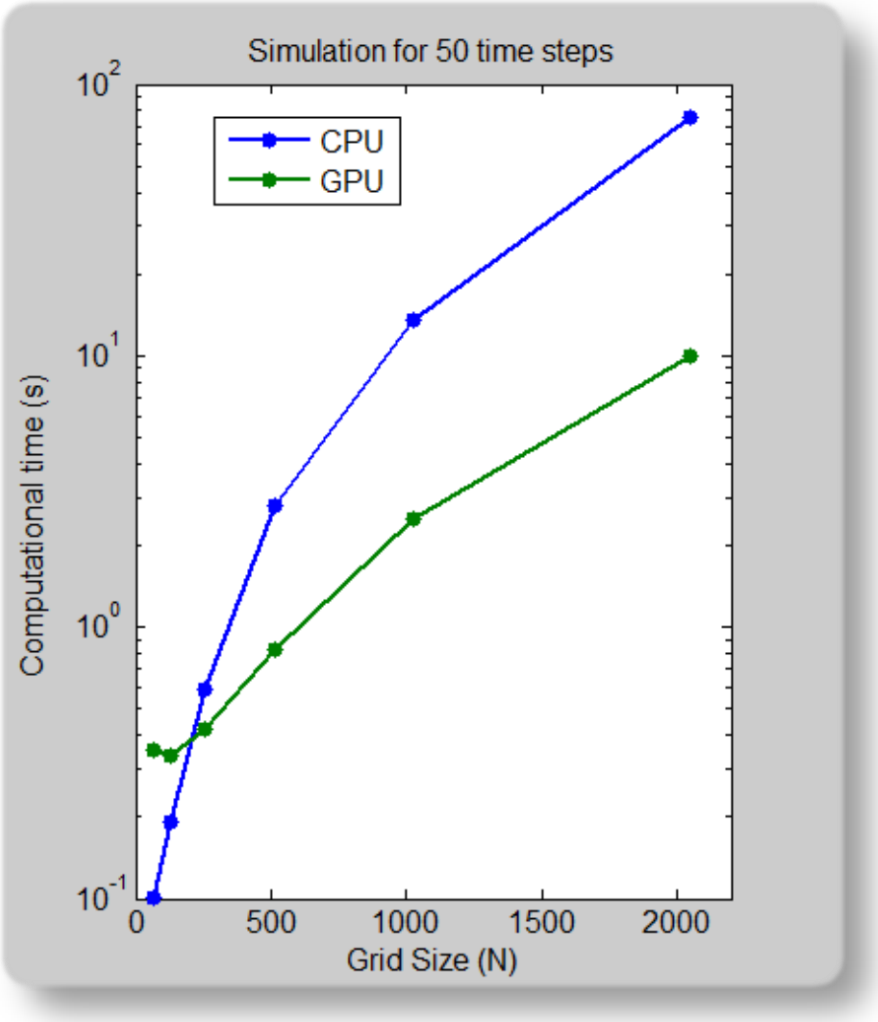
$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

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



Benchmark: Solving 2D Wave Equation

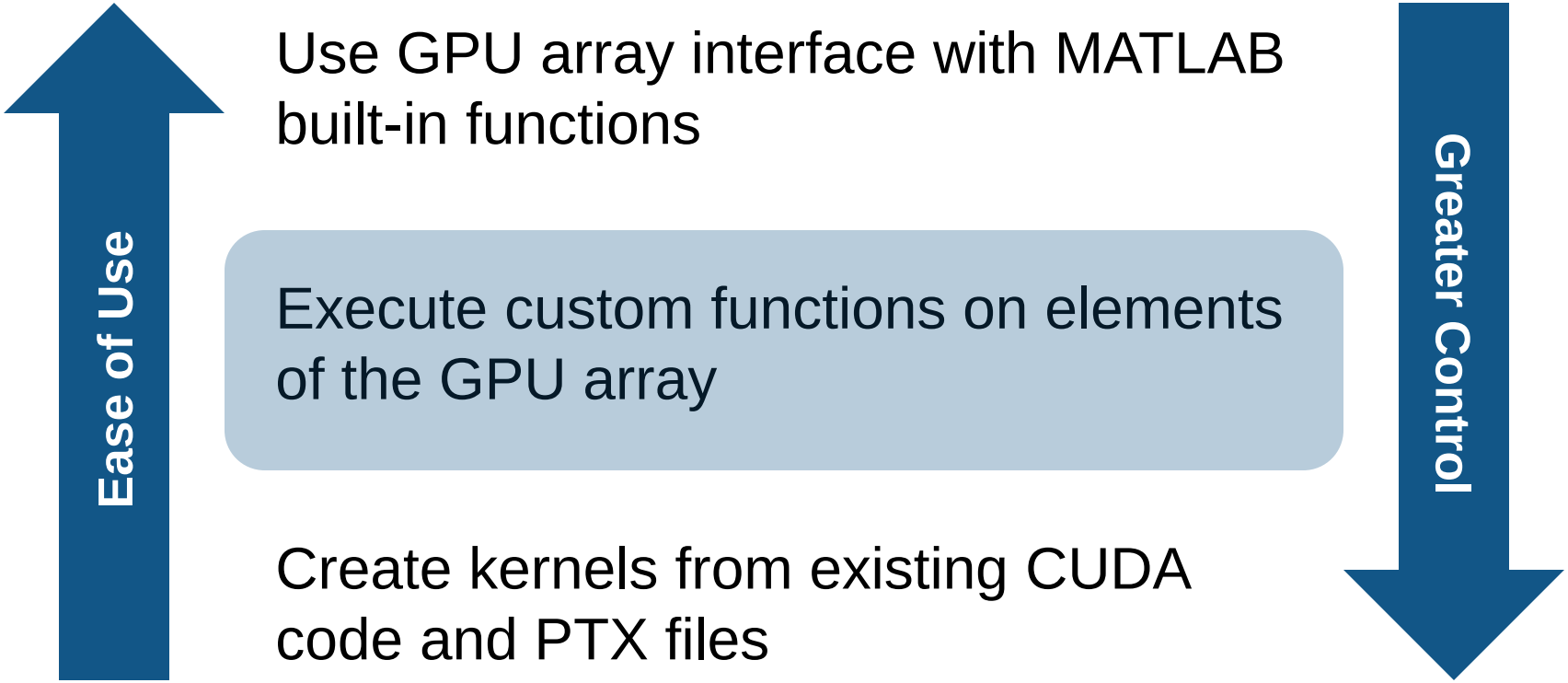
CPU vs GPU



Grid Size	CPU (s)	GPU (s)	Speedup
64 x 64	0.1004	0.3553	0.28
128 x 128	0.1931	0.3368	0.57
256 x 256	0.5888	0.4217	1.4
512 x 512	2.8163	0.8243	3.4
1024 x 1024	13.4797	2.4979	5.4
2048 x 2048	74.9904	9.9567	7.5

Intel Xeon Processor X5650, NVIDIA Tesla C2050 GPU

Options for Targeting GPUs



Use GPU array interface with MATLAB built-in functions

Execute custom functions on elements of the GPU array

Create kernels from existing CUDA code and PTX files

Using `arrayfun` on GPU

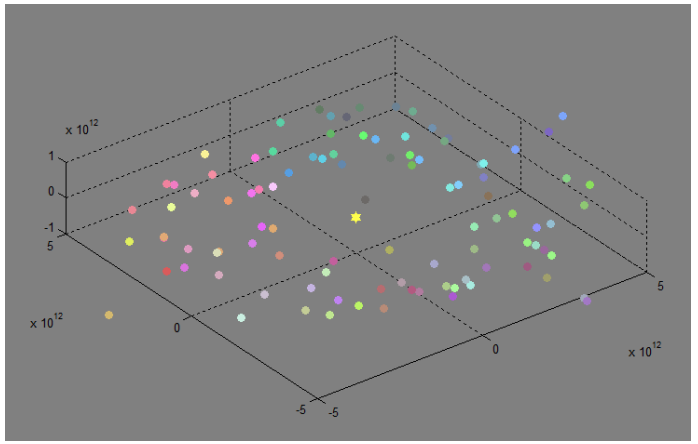
```
gain = 1.5;
offset = -0.1;
x = parallel.gpu.GPUArray.rand(1000,1); %Create on GPU
fh = @(x) myGPUfun(x, gain, offset);
x = arrayfun(fh, x)%Execute on GPU

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

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

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

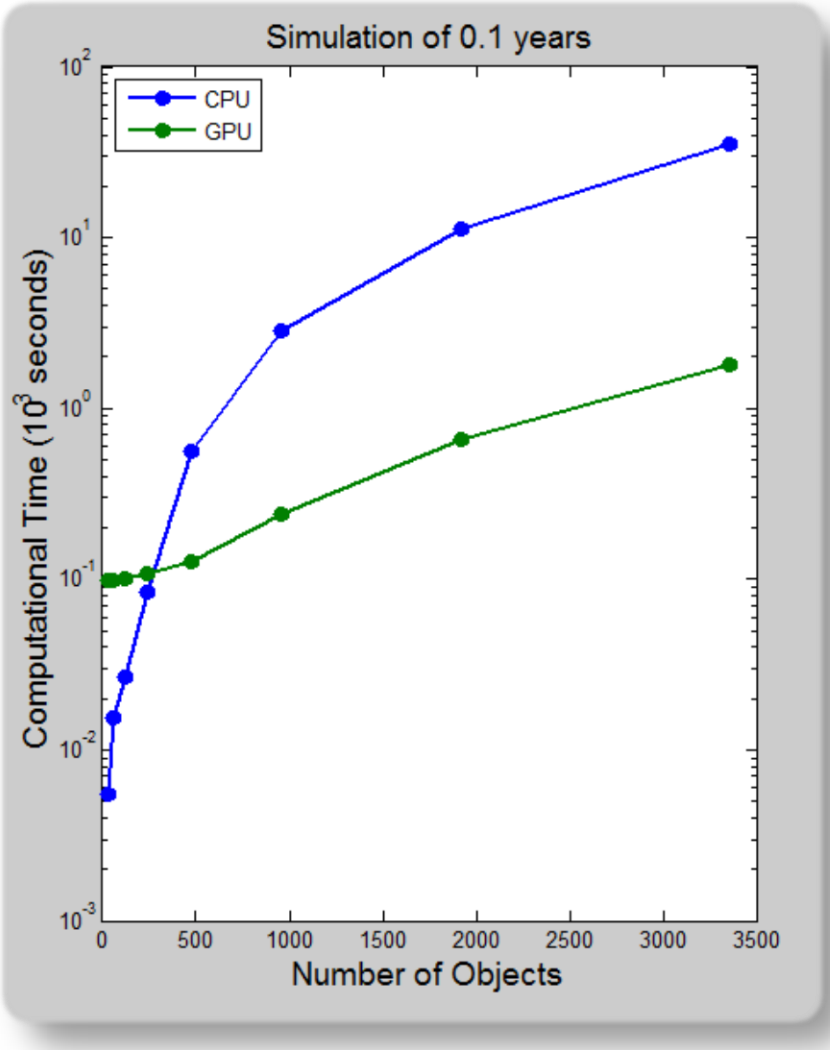
Example: N-Body Simulation

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



Benchmark: N-Body Simulation

CPU vs GPU



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

Intel Xeon Processor W3550, NVIDIA Tesla C2050 GPU

Options for Targeting GPUs

Use GPU array interface with MATLAB built-in functions

Execute custom functions on elements of the GPU array

Create kernels from existing CUDA code and PTX files

Invoking CUDA Kernels

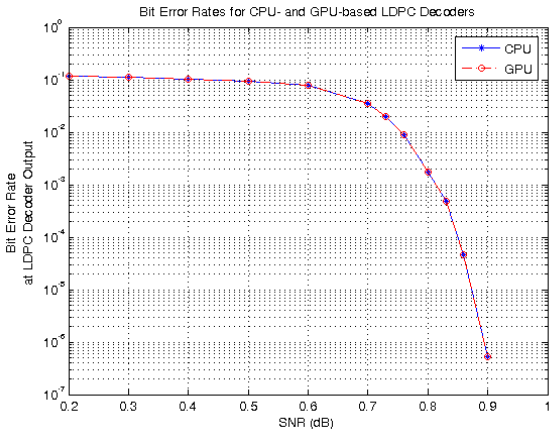
```
% Setup
kernel = parallel.gpu.CUDAKernel('myKern.ptx','myKern.cu');

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

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

Best Practices for using GPU with MATLAB

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

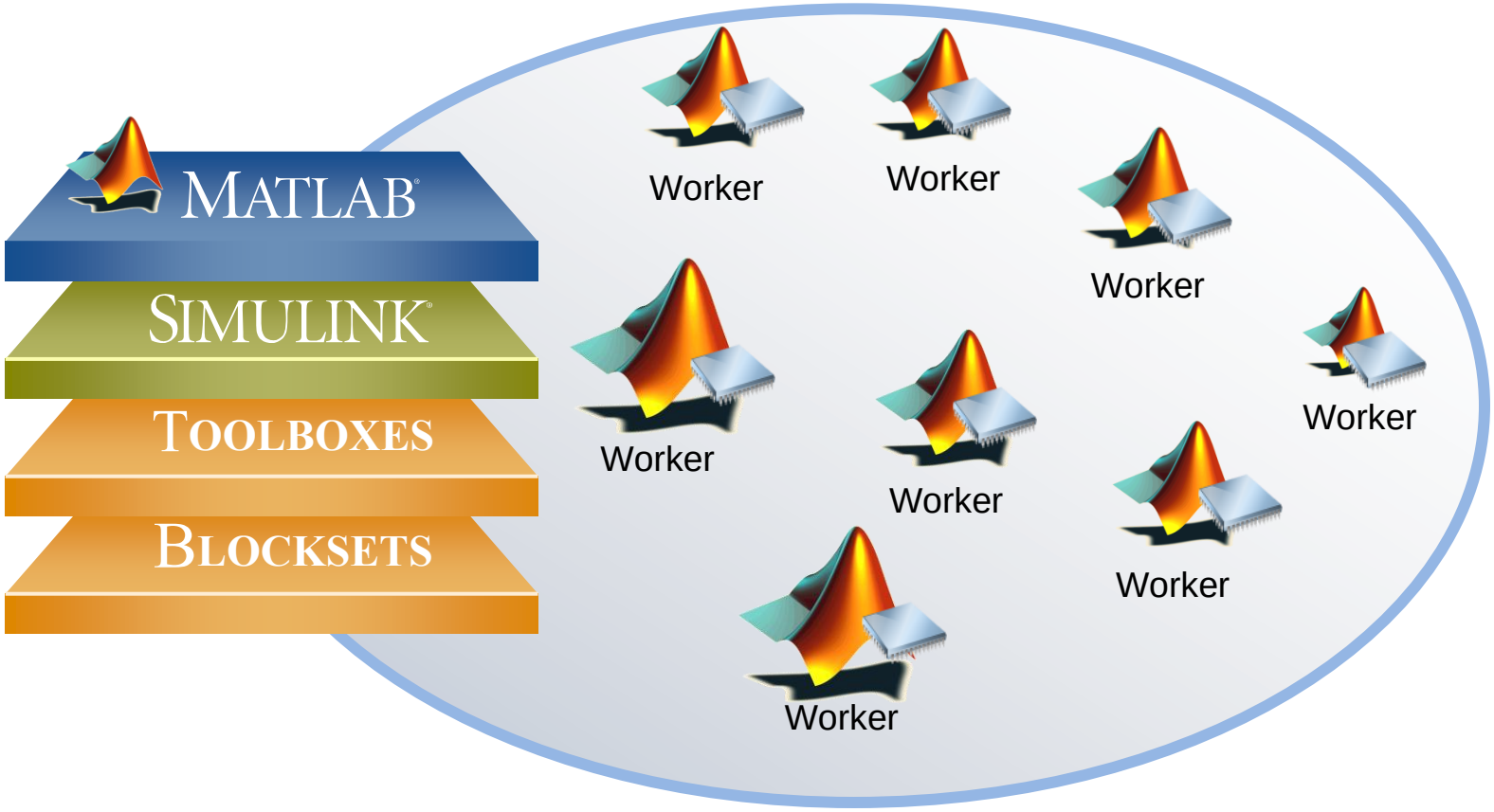
Support for Communications System Toolbox

- GPU implementations of LDPC Decoder, Viterbi Decoder, AWGN Channel, PSK Modulator, Block Interleaver, Block Deinterleaver
- DVB-S System Simulation Demo
  http://www.mathworks.com/products/communications/demos.html




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

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

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

Scaling Up to Run on Multiple GPUs



Running on Multiple GPUs

Single GPU

```
N = 1000;  % Number of iterations

A = gpuArray(A); % transfer data to GPU

for ix = 1:M
    % Do the GPU-based calculation
    X = myGPUFunction(ix,A);
    % Gather data
    Xtotal(ix,:)= gather(X);
end
```

Multiple GPUs

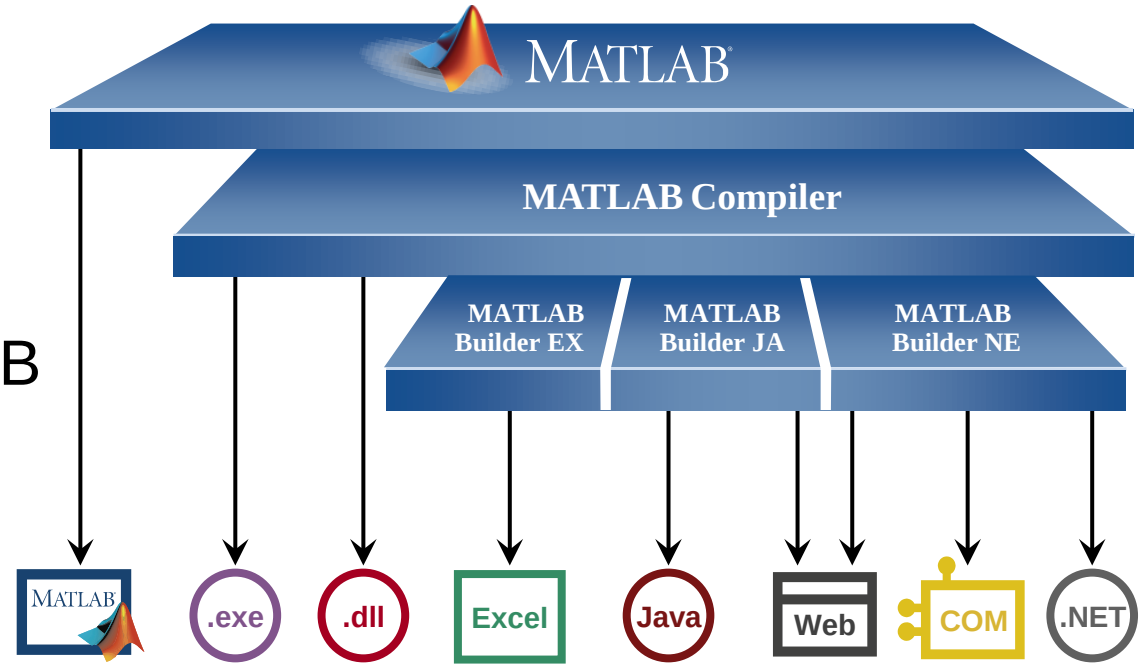
```
N = 1000;  % Number of iterations

spmd
  % Assign each worker a different GPU
  gpuDevice(labindex);
  A = gpuArray(A); % transfer data
end

parfor ix = 1:M
    % Do the GPU-based calculation
    X = myGPUFunction(ix,A);
    % Gather data
    Xtotal(ix,:)= gather(X);
end
```

Agenda

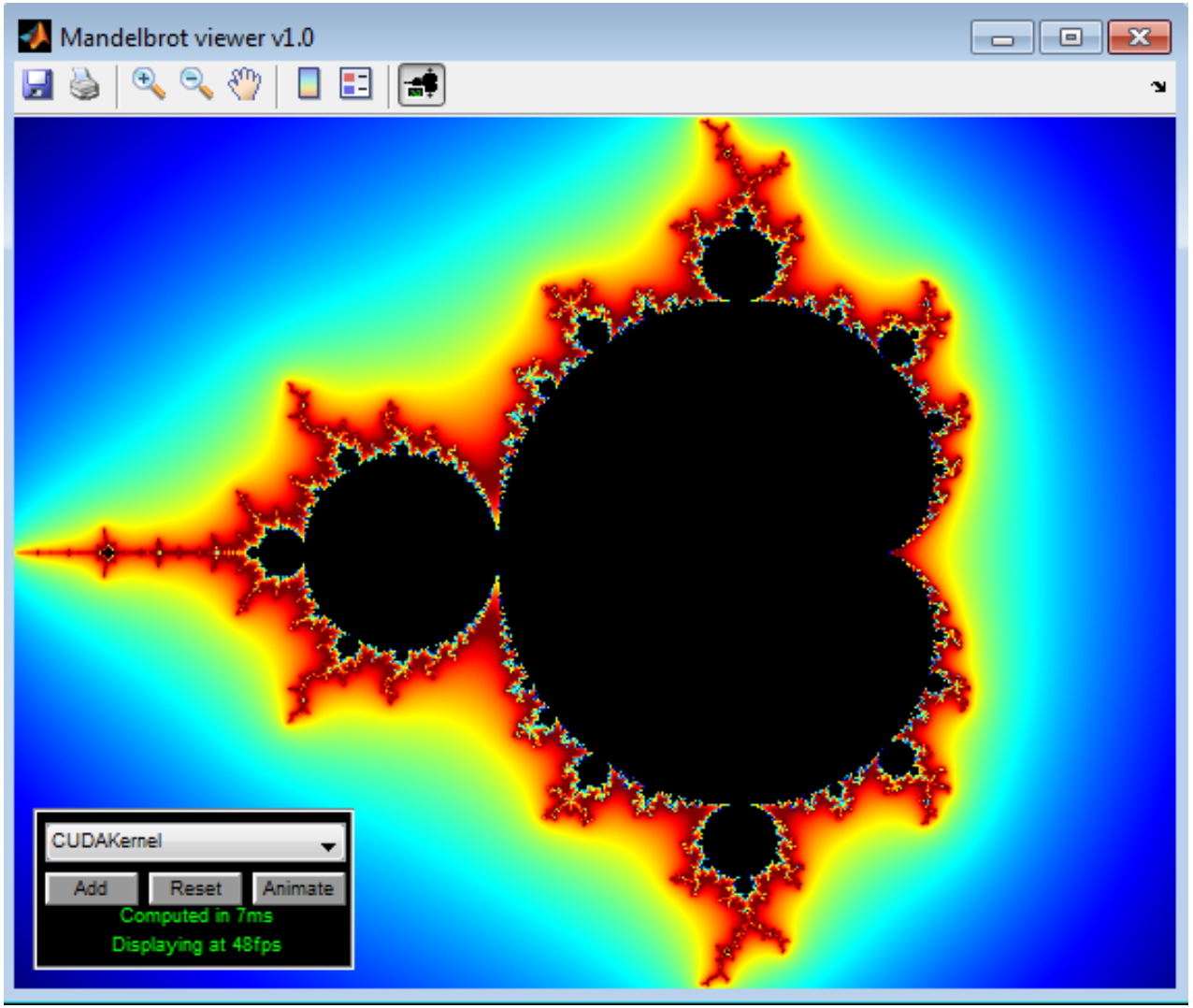
- GPU background
- Introduction to parallel computing products
- GPU computing with MATLAB
- Sharing GPU applications

Deploying Applications with MATLAB

- Give MATLAB code to other users
- Share applications with end users who do not need MATLAB
  - Stand-alone executables
  - Shared libraries
  - Software components



Example: Deploying Mandelbrot Set Viewer



Additional Resources

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