GPU Computing with MATLAB

Andy Thé
Product Marketing Manager
Image Processing Applications
GPU Computing with MATLAB

- For MATLAB Programmers
  - Acceleration MATLAB Code with GPUs
  - Minimal code changes

- For CUDA Programmers
  - Create test harnesses for your kernels
  - Quickly explore algorithm parameters
  - Analyze and visualize kernel results
<table>
<thead>
<tr>
<th>What is MATLAB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo: Designing a Camera Pipeline in MATLAB</td>
</tr>
<tr>
<td>Demo: Brain Scan Demo – CPU vs GPU</td>
</tr>
<tr>
<td>Demo: White Balance Example using CUDA Code</td>
</tr>
<tr>
<td>Summary</td>
</tr>
</tbody>
</table>
What is MATLAB?

- High level language and development environment for:
  - Algorithm and application development
  - Data analysis
  - Mathematical modeling
  - Multicore and GPU computing*
- Extensive math, engineering, and plotting functionality
- Add-on products for image and video processing, communications, signal processing, financial modeling, and more
- Over 1.3 million COM & EDU users

* Requires Parallel Computing Toolbox
Algorithm Development Process

Requirements

Research & Design
- Explore and discover
- Gain insight into problem
- Evaluate options, trade-offs

Implementation
- Migrate design to production
- Optimize performance
- Deploy / Integrate / Test

Test & Verification

Languages:
- .NET
- .dll
- CUDA
- .C/C++
- Java
- HDL
Running MATLAB code on the GPU

- 200+ MATLAB functions that are GPU enabled
  - Random number generation
  - FFT
  - Matrix multiplications
  - Solvers
  - Convolutions
  - Min/max
  - SVD
  - Cholesky and LU factorization

- Additional support in toolboxes
  - **Image Processing**
    - Morphological filtering, 2-D filtering, …
  - **Communications**
    - Turbo, LDPC, and Viterbi decoders, …
  - **Signal Processing**
    - Cross correlation, FIR filtering, …
  - **Neural Networks**
    - Network training and simulation

- Ability to launch CUDA kernels

- Ability to deploy MATLAB GPU applications

BWMORPH  IMHIST
BWLOOKUP  IMNOISE
CORR2  IMOPEN
EDGE  IMSIZE
HISTEQ  IMROTATE
IMADJUST  IMSHOW
IMBOTHAT  IMTOPHAT
IMCLOSE  IMWARP
IMDILATE  MEAN2
IMERODE  MEDFILT2
IMFILTER  PADARRAY
IMGRADIENT  RGB2GRAY
## Agenda

<table>
<thead>
<tr>
<th>What is MATLAB?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Demo</strong>: Designing a Camera Pipeline in MATLAB</td>
</tr>
<tr>
<td>Demo: Brain Scan Demo – CPU vs GPU</td>
</tr>
<tr>
<td>Demo: White Balance Example using CUDA Code</td>
</tr>
<tr>
<td>Summary</td>
</tr>
</tbody>
</table>
Demo: Digital Camera Pipeline
From Sensor Data to Image
Stages of the Camera Pipeline

1. **Noise Reduction**: Reduce noise in the raw data

2. **Demosaic**: Interpolate the raw image into an RGB image

3. **Tone Mapping**: Convert sensor RGB values to RGB values

4. **White Balance**: Adjust color of image to compensate for different lighting conditions

5. **Gamma Correction**: Adjust color of image for display
Approaches to GPU Computing in MATLAB

- GPU enabled functions
- Simple programming constructs: `gpuArray, gather`
- Advanced programming constructs: `arrayfun`
- Interface for experts: `CUDAKernel, MEX support`

http://www.mathworks.com/help/distcomp/run-cuda-or-pxt-code-on-gpu.html
## Agenda

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is MATLAB?</td>
<td></td>
</tr>
<tr>
<td>Demo: Designing a Camera Pipeline in MATLAB</td>
<td></td>
</tr>
<tr>
<td><strong>Demo:</strong> Brain Scan Demo – CPU vs GPU</td>
<td></td>
</tr>
<tr>
<td>Demo: White Balance Example using CUDA Code</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>
Demo: Brain Scan
CPU vs GPU
## Agenda

<table>
<thead>
<tr>
<th>What is MATLAB?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo: Designing a Camera Pipeline in MATLAB</td>
<td></td>
</tr>
<tr>
<td>Demo: Brain Scan Demo – CPU vs GPU</td>
<td></td>
</tr>
<tr>
<td>Demo: White Balance Example using CUDA Code</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>
Stages of the Camera Pipeline

1. **Noise Reduction**: Reduce noise in the raw data

2. **Demosaic**: Interpolate the raw image into an RGB image

3. **Tone Mapping**: Convert sensor RGB values to RGB values

4. **White Balance**: Adjust color of image to compensate for different lighting conditions

5. **Gamma Correction**: Adjust color of image for display
Demonstration: White balance algorithm

- Goal: Deliver algorithm as CUDA C/C++

- How can MATLAB support the CUDA development process?
Programming Parallel Applications (GPU)

- GPU enabled functions
- Simple programming constructs: `gpuArray`, `gather`
- Advanced programming constructs: `arrayfun`
- Interface for experts: `CUDAKernel`, MEX support

http://www.mathworks.com/help/distcomp/run-cuda-or-ptx-code-on-gpu.html
Summary Part 1: Apply Scaling Factors on GPU

**Code run entirely on CPU**

```matlab
function imageData = whitebalance(imageData)
% WHITEBALANCE forces average image color to be gray.
% Find the average values for each channel.
avg_rgb = mean(mean(imageData));
% Find average gray value and compute scaling factor.
factors = max(mean(avg_rgb), 128)/avg_rgb;
% Adjust the image to the new gray value.
imageData(:,:,1) = uint8(imageData(:,:,1)*factors(1));
imageData(:,:,2) = uint8(imageData(:,:,2)*factors(2));
imageData(:,:,3) = uint8(imageData(:,:,3)*factors(3));
```

Execution time on CPU: **120 ms**

**Applying Scaling Factors on GPU**

```matlab
function adjustedImage = whitebalance_gpu(imageData)

% Load the Kernel
kernel = parallel.gpu.CUDAKernel('applyScaleFactors.ptxw64', 'applyScaleFactors.cu');
...
% Copy our Image to the GPU
imageDataGPU = gpuArray(imageData);
...
% Apply kernel to scale the color values
adjustedImageGPU = feval(kernel, adjustedImageGPU, imageDataGPU, factors, nRows, nCols);
```

Execution time on GPU: **2.2 ms**

50x Faster
Summary Part 2: Compute Scaling Factors on GPU

**Code run entirely on CPU**

```matlab
function imageData = whitebalance(imageData)
    % WHITEBALANCE forces average image color to be gray.
    % Find the average values for each channel.
    avg_rgb = mean(mean(imageData));
    % Find average gray value and compute scaling factors.
    factors = max(mean(avg_rgb), 128)./avg_rgb;
    % Adjust the image to the new gray value.
    imageData(:,:,1) = uint8(imageData(:,:,1)*factors(1));
    imageData(:,:,2) = uint8(imageData(:,:,2)*factors(2));
    imageData(:,:,3) = uint8(imageData(:,:,3)*factors(3));
end
```

**Computing Scaling Factors on GPU**

```matlab
function imageData = whitebalance(imageData)
    % WHITEBALANCE forces average image color to be gray.
    % Find the average values for each channel.
    avg_rgb = mean(mean(imageData));
    % Compute the factors from the mean.
    computeFactorsKernel = parallel.gpu.CUDAKernel(...
        'computeScaleFactors.ptxw64','computeScaleFactors.cu');
    factors = feval(computeFactorsKernel, avg_rgb);
    % 3 doubles of shared memory
    computeFactorsKernel.SharedMemorySize = 3*8;
    computeFactorsKernel.ThreadBlockSize = [3 1 1];
    factors = feval(computeFactorsKernel, avg_rgb);
    imageData(:,:,1) = uint8(imageData(:,:,1)*factors(1));
    imageData(:,:,2) = uint8(imageData(:,:,2)*factors(2));
    imageData(:,:,3) = uint8(imageData(:,:,3)*factors(3));
end
```
Summary Part 3: Compute Mean on GPU with NPP

Code run entirely on CPU

```matlab
function imageData = whitebalance(imageData)
    % WHITEBALANCE forces average image color to be gray.
    avg_rgb = mean(mean(imageData));
    avg_rgb = max(mean(avg_rgb), 128)./avg_rgb;
    imageData(:,:,1) = uint8(imageData(:,:,1)*factors(1));
    imageData(:,:,2) = uint8(imageData(:,:,2)*factors(2));
    imageData(:,:,3) = uint8(imageData(:,:,3)*factors(3));
end
```

Implementing NPP through MEX

```matlab
function adjustedImage = whitebalance_gpu(imageData)
    % Find the average values for each channel.
    avg_rgb = computeMeanMEX(imageDataGPU);
    % Copy our Image to the GPU
    imageDataGPU = gpuArray(imageData);
    % Apply kernel to scale the color values
    adjustedImageGPU = feval(kernel, adjustedImageGPU, imageDataGPU, factors, nRows, nCols);
end
```
Summary Part 4: Integrate C with CUDA

Code run entirely on CPU

```c
function imageData = whitebalance(imageData)
% WHITEBALANCE forces average image color to be gray.

% Find the average values for each channel.
avg_rgb = mean(mean(imageData));

% Find average gray value and compute scaling factor.
factors = max(mean(avg_rgb), 128)./avg_rgb;

% Adjust the image to the new gray value.
imageData(:,:,1) = uint8(imageData(:,:,1)*factors(1));
imageData(:,:,2) = uint8(imageData(:,:,2)*factors(2));
imageData(:,:,3) = uint8(imageData(:,:,3)*factors(3));
```

Implementing C++ with CUDA Kernels

```c
// dllmain.cpp : Defines the entry point for the DLL application.
#include <mex.h>
#include <gpu\mxGPUArray.h>
#include "whitebalance.h"

// A function to compute the mean of all the elements of a uint8 matrix on the GPU.
// Returns a single scalar value on the GPU.
void mexFunction ( const int nlhs, mxArray *plhs[],
                   const int nrhs,
                   const mxArray * const prhs[] )
{
    //Ensure the GPU system is initialized.
    int mwGpuStat = 0;
    mwGpuStat = mxInitGPU();
    if ( mwGpuStat != 0 )
        mexErrMsgTxt( "Error initializing MW GPU system" );
    ... 
```
## Agenda

<table>
<thead>
<tr>
<th>What is MATLAB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo: Designing a Camera Pipeline in MATLAB</td>
</tr>
<tr>
<td>Demo: Brain Scan Demo – CPU vs GPU</td>
</tr>
<tr>
<td>Demo: White Balance Example using CUDA Code</td>
</tr>
</tbody>
</table>

## Summary
Programming Parallel Applications (GPU)

- GPU enabled functions
- Simple programming constructs: `gpuArray`, `gather`
- Advanced programming constructs: `arrayfun`
- Interface for experts: `CUDAKernel`, MEX support

http://www.mathworks.com/help/distcomp/run-cuda-or-ptx-code-on-gpu.html
MATLAB Value-add to CUDA Programmers

- Develop prototype code to explore algorithms
- Manage GPU data and launch kernels using a simple interface
- Incrementally develop and test kernels
- Analyze and visualize kernel results
Scaling to Run 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

A = gpuArray(A); % transfer data

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

- 200+ MATLAB functions that are GPU enabled
  - Random number generation
  - FFT
  - Matrix multiplications
  - Solvers
  - Convolutions
  - Min/max
  - SVD
  - Cholesky and LU factorization

- Additional support in toolboxes
  - Image Processing: Morphological filtering, 2-D filtering, ...
  - Communications: Turbo, LDPC, and Viterbi decoders, ...
  - Signal Processing: Cross correlation, FIR filtering, ...
  - Neural Networks: Network training and simulation

- Ability to launch CUDA kernels

- Ability to deploy MATLAB GPU applications
GPU Computing with MATLAB

- For MATLAB Programmers
  - Acceleration MATLAB Code with GPUs
  - Minimal code changes

- For CUDA Programmers
  - Create test harnesses for your kernels
  - Quickly explore algorithm parameters
  - Analyze and visualize kernel results
Questions & Answers