

# CS344 Introduction to Parallel Programming

## Lesson 7.1: Additional Parallel Computing Topics

### L7.1-7.2-Overview

Parallel optimization patterns

Libraries

Programming power tools

Platforms

- Languages

- Cross-platform

Dynamic Parallelism - Stephen Jones, NVIDIA

### L7.1-7.3-Parallel Optimization Patterns

Parallel Optimization Patterns

Stratton et al. In Par 2012  
Computer 2012

→ survey optimized GPU codes

observe patterns that emerge

classify into 7 basic techniques

### L7.1-7.4-Data Layout Transformation

1. Data layout transformation

Quiz: Global memory coalescing is important because:

- DRAM systems transfer large chunks of data per transaction
- DRAM systems transfer data in many small chunks, one per thread request
- GPUs have no on-chip caches to prefetch data

### L7.1-7.5-Additional Data Transformation Methods

#### 1. Data layout transformation

→ Reorganize data layout for better memory perf.

Burst utilization, e.g. AoS → SoA



"Partition camping"

Ninja Array of Structures of Tiled Arrays (ASTA)

### L7.1-7.6-Burst Utilization

#### 1. Data layout transformation

→ Reorganize data layout for better memory perf.

```
struct foo {
    float a; Array
    float b; of
    float c; Structures
    float d; 
} A[8];
```



```
struct foo {
    float a[8]; Structure
    float b[8]; of
    float c[8];
    float d[8]; Arrays
} A;
```



```
struct foo {
    float a; Array
    float b; of
    float c; Structures
    float d;
} A[8];
```

```
struct foo {
    float a[8]; Structure
    float b[8]; of
    float c[8];
    float d[8]; Arrays
} A;
```

Quiz: which layout will perform better on these codes?

int i = threadIdx.x;

A[i].a++;

A[i].b += A[i].c \* A[i].d;

O A-o-S

int i = threadIdx.x;

A.a[i]++;

A.b[i] += A.c[i] \* A.d[i];

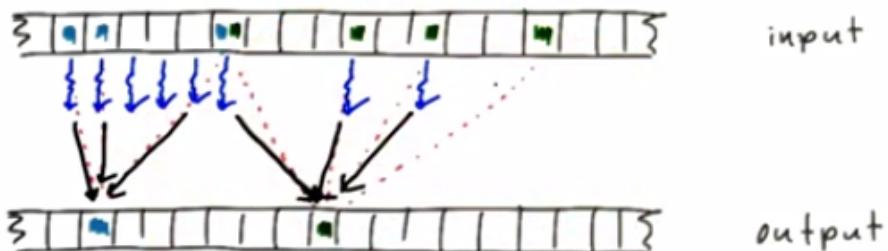
O S-o-A

### L7.1-7.7-Scatter To gather Transformation

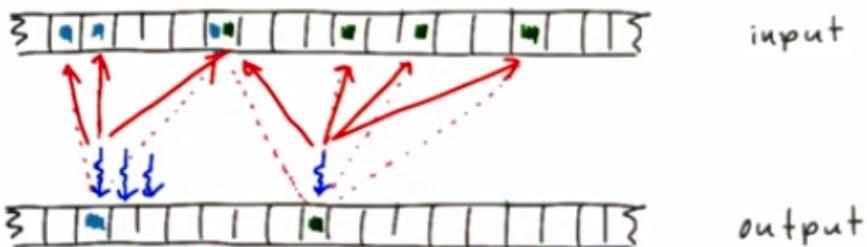
#### Parallel Optimization Patterns

1. Data layout transformation
2. Scatter - to - gather transformation

#### 2. Scatter - to - gather transformation



#### 2. Scatter - to - gather transformation



#### 2. Scatter - to - gather transformation

Quiz: which code snippet represents a gather? \_\_\_\_\_  
 which will run more efficiently? \_\_\_\_\_

①

```
float third = in[i] / 3.0f;
out[i-1] += third;
out[i] += third;
out[i+1] += third;
```

②

out[i] = (in[i-1] +  
 in[i] +  
 in[i+1])  
 / 3.0f;

## 2. Scatter - to - gather transformation

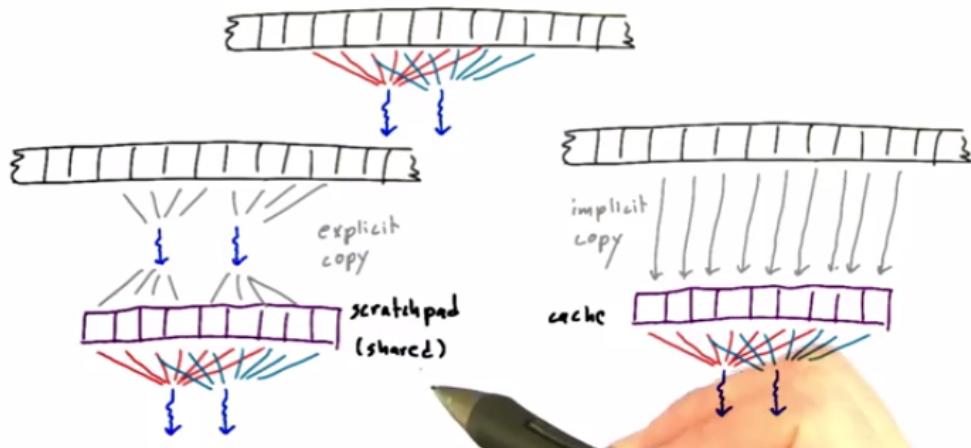
Gather: many overlapping reads

Scatter: many potentially conflicting writes

→ may need to combine with #5 "binning"

### L7.1-7.8-Tiling

3. Tiling: buffering data into fast on-chip storage for repeated access



3. Tiling: buffering data into fast on-chip storage for repeated access

Quiz: Which code snippets would benefit from tiling?

Averaging values from 5 arrays?

```
__global__ void foo(float out[], float A[], float B[], float C[], float D[], float E[])
{
    int i = threadIdx.x;
    out[i] = (A[i] + B[i] + C[i] + D[i] + E[i]) / 5.0f;
}
```

Averaging 5 values from 1 array?

```
__global__ void bar(float out[], float in[])
{
    int i = threadIdx.x;
    out[i] = (in[i-2] + in[i-1] + in[i] + in[i+1] + in[i+2]) / 5.0f;
}
```



### L7.1-7.9-Tiled vs Non-Tiled Code

code available:

<https://github.com/udacity/cs344/blob/master/Unit7%20Code%20Snippets/tiling/>

### L7.1-7.10-Tiling Program Quiz

3. Tiling: buffering data into fast on-chip storage for repeated access

Programming exercise: tile both foo() and bar()  
with 128-thread blocks and 128-element tiles

OK to ignore boundary effects at beginning and end of in[] and out[] arrays

% time of non-tiled version for foo() \_\_\_\_\_  
bar() \_\_\_\_\_



### L7.1-7.11-Privatization

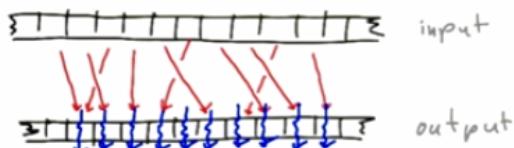
#### 4. Privatization

threads sharing input: Good output: Bad



### L7.1-7.12-Binning

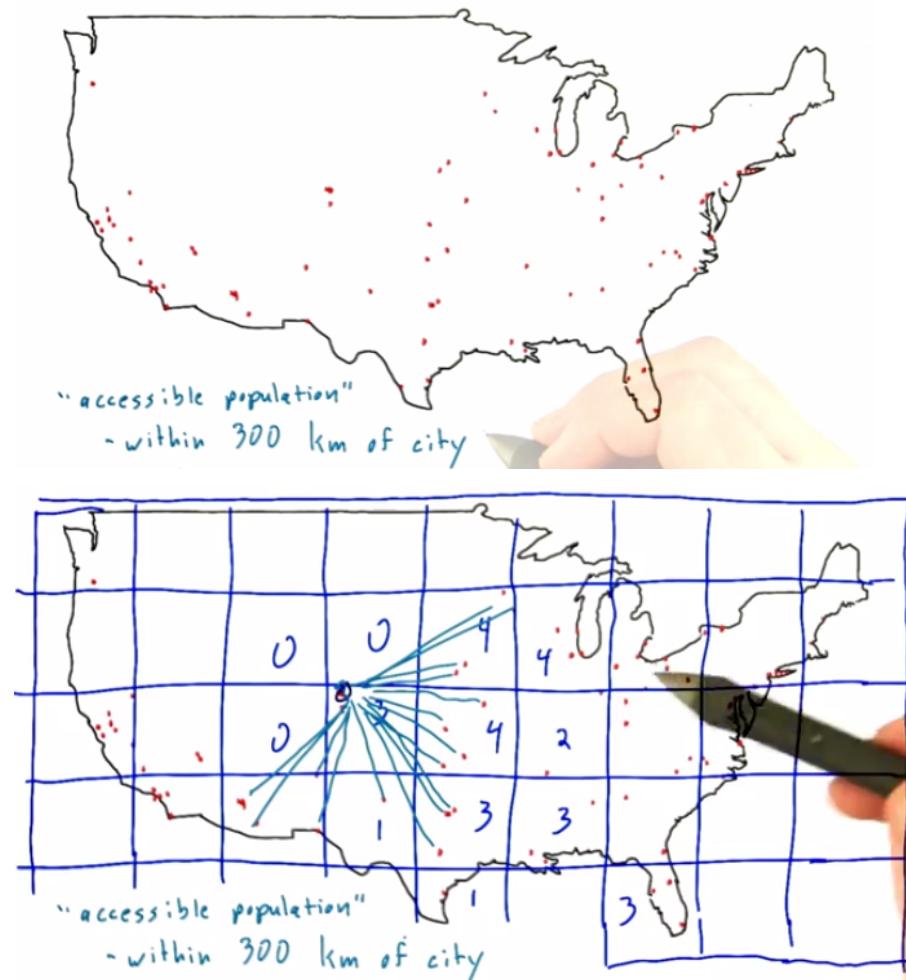
#### 5. Binning / spatial data structures



binning: build data structure that maps output locations to the relevant input data

sort input into bins containing input elements

### L7.1-7.13-Accessible Population

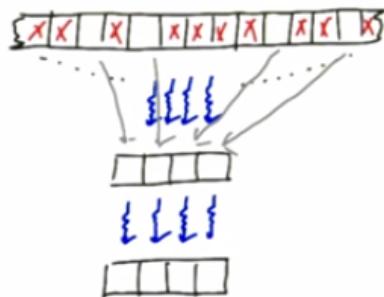
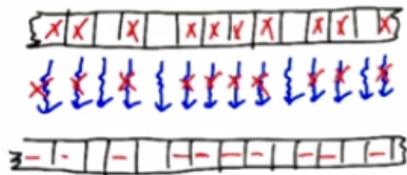


**Quiz:** which operations would benefit from this geographic binning?

- Compute a histogram of city distances from Denver?
- Given a list of roads (lists of line segments), compute population of cities within 20 km?
- Given a list of counties, and a list of cities each with an index to the containing county, compute the city-dwelling population of all counties

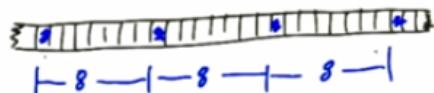
## L7.1-7.14-Compaction

### 6. Compaction



Quiz: Suppose only every 8<sup>th</sup> element will be processed.

Assume computation-limited



About how much less time will computation take on the compacted array than the original input?

— every 8<sup>th</sup> element?

— every 32<sup>nd</sup> element?

— every 128<sup>th</sup> element?

## L7.1-7.15-Regularization

### 7. Regularization — load balancing

→ reorganizing input data to reduce load imbalance

irregular parallelism → regular parallelism

ex: provision 5 cities/bin, handle overflow specially,

- eg on CPU

- different kernel, different algorithm

### L7.1-7.16-Regularization has the most impact when

Quiz: check the true statements:

Regularization will likely have most impact when:

- load is mostly evenly distributed, with outliers  
ex: most US counties  $\leq 1$  city ; a few have  $> 10$
- load varies wildly, no real "average case"  
ex: power law
- application can predict load imbalance at run-time

### L7.1-7.17-Parallel Optimization Patterns Recap

#### Parallel Optimization Patterns

1. Data layout transformation
2. Scatter - to - gather transformation
3. Tiling
4. Privatization
5. Binning / spatial data structures
6. Compaction
7. Regularization

### L7.1-7.18-Libraries

#### Libraries

Huge ecosystem

NVIDIA, 3rd-party, commercial, open-source, etc.

Highlight libraries that are:

Mature

Designed for performance by experts

Retuned for new GPU architectures

Use them!

## Libraries

1. cuBLAS - BLAS (Basic Linear Algebra Subroutines)
2. cuFFT - 1D, 2D, 3D FFT routines
3. cuSPARSE - BLAS-like routines for sparse matrix formats
4. CU RAND - Pseudo- and quasi-random generation routines
5. NPP - Low-level image processing primitives
6. Magma - GPU + multicore CPU LAPACK routines
7. CULA - Eigensolvers, matrix factorizations & solvers
8. ArrayFire - Framework for data-parallel array manipulation

### L7.1-7.19-How to Use Libraries

#### Libraries

1. Substitute library calls

saxpy (...) → cublasSaxpy (...)

2. Manage data locality

cudaMalloc(), cudaMemcpy(), etc

cublasAlloc(), cublasSetVector(), etc

3. Rebuild and link nvcc myobj.o -l cublas

### L7.1-7.20-cuBLAS

```
1 // Accelerated using CUBLAS
2 int N = 1 << 20;
3 cublasInit();
4 cublasAlloc(N, sizeof(float), (void**)&d_x);
5 cublasAlloc(N, sizeof(float), (void*)&d_y);
6
7 cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
8 cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);
9
10 // Perform SAXPY on 1M elements: y[] = a*x[] + y[]
11 cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
12
13 cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);
14
15 cublasFree(d_x);
16 cublasFree(d_y);
17 cublasShutdown();
18
```

### L7.1-7.21-Timing Sorting in Thrust

code available:

<https://github.com/udacity/cs344/tree/master/Unit7%20Code%20Snippets/thrust>

## L7.1-7.22-Thrust

### 1. Thrust

Analogous to C++ Standard Template Library (STL)  
containers      iterators      ("Boost")

Host code using `thrust::device_vector`  
- Sort, scan, reduce, reduce-by-key

```
thrust::device_vector<float> X(3);  
...  
float result = thrust::reduce(X.begin(), X.end());
```

### 1. Thrust

Analogous to C++ Standard Template Library (STL)  
containers      iterators      ("Boost")

Host code using `thrust::device_vector`  
- Sort, scan, reduce, reduce-by-key

```
thrust::device_vector<float> X(3);  
float init = 0.0f;  
float result = thrust::reduce(X.begin(), X.end(),  
    init, thrust::maximum<float>());
```

### 1. Thrust

Analogous to C++ Standard Template Library (STL)  
containers      iterators      ("Boost")

Host code using `thrust::device_vector`  
- Sort, scan, reduce, reduce-by-key  
- Transform input vector(s)  
  - E.g. vector addition `thrust::plus`  
  - Apply user-defined transformation-functor

## 1. Thrust

Analogous to C++ Standard Template Library (STL)  
containers      iterators      ("Boost")

Host code using `thrust::device_vector`

- Sort, scan, reduce, reduce-by-key
- Transform input vector(s)
- Interoperate with CUDA code

`thrust::device_vector`  $\longleftrightarrow$  CUDA



## 1. Thrust

Analogous to C++ Standard Template Library (STL)  
containers      iterators      ("Boost")

Host code using `thrust::device_vector`

- Sort, scan, reduce, reduce-by-key
- Transform input vector(s)
- Interoperate with CUDA code



Avoid boilerplate, exploit high performance code

## 1. Thrust

Programming exercise:

sort  $10^6$  floats using Thrust

time sorting  $10^5$  floats \_\_\_\_\_ bytes \_\_\_\_\_

(in ms)             $10^6$  floats \_\_\_\_\_ bytes \_\_\_\_\_

\_\_\_\_\_  $10^7$  floats \_\_\_\_\_ bytes \_\_\_\_\_

## L7.1-7.23-CUB

### CUDA C++ Programming Power Tools

1. Thrust - host-side interface, no kernels
2. CUB

Problem: software reuse in CUDA kernels

radix sort library

- how many threads?
- how much shared memory?
- which algorithm?

### CUDA C++ Programming Power Tools

1. Thrust - host-side interface, no kernels

2. CUB - CUDA Unbound

software reuse in CUDA kernels

compile-time binding with templates, typing  
high performance

enables optimization, auto-tuning

## L7.1-7.24-Why Use CUB



Staged data (optional)

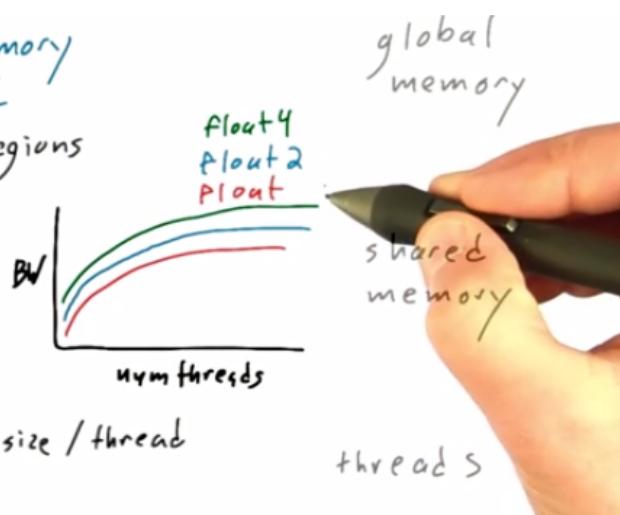
shared  
memory

Algorithms

threads

Accessing global memory can be complicated

- loading halo regions
- coalescing
- Little's Law
  - latency
  - bandwidth
  - occupancy
  - transaction size / thread

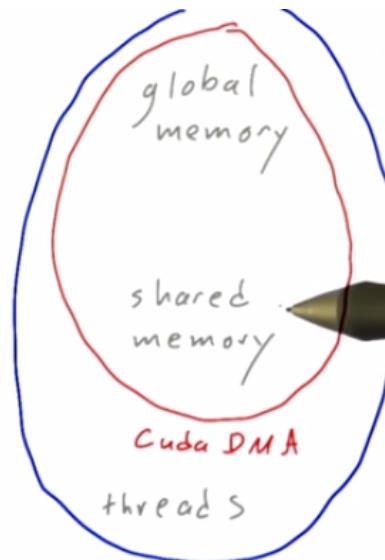


Accessing global memory can be complicated

- loading halo regions
- coalescing
- Little's Law
- Kepler LDG intrinsic

Summary: explicit shared memory

- predictable high performance
- burden on programmer



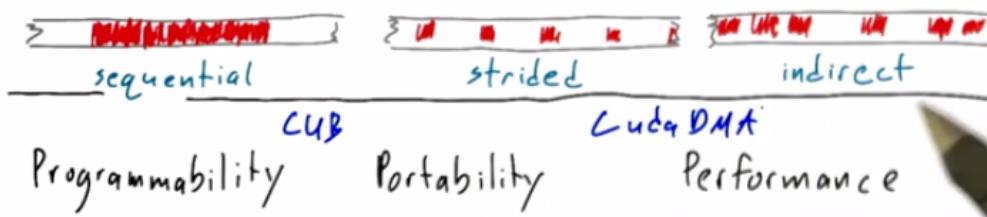
#### L7.1-7.25-CUDA DMA

Cuda DMA - template library designed to:

- make it easier to use shared memory
- at high performance

Cuda DMA objects → each shared-memory buffer

- explicit transfer pattern



## CUDA C++ Programming Power Tools

1. Thrust - host-side interface, no kernels
2. CUB - CUDA Unbound
3. Cuda DMA - global ↔ shared memory

### L7.1-7.26-CUB Exercise

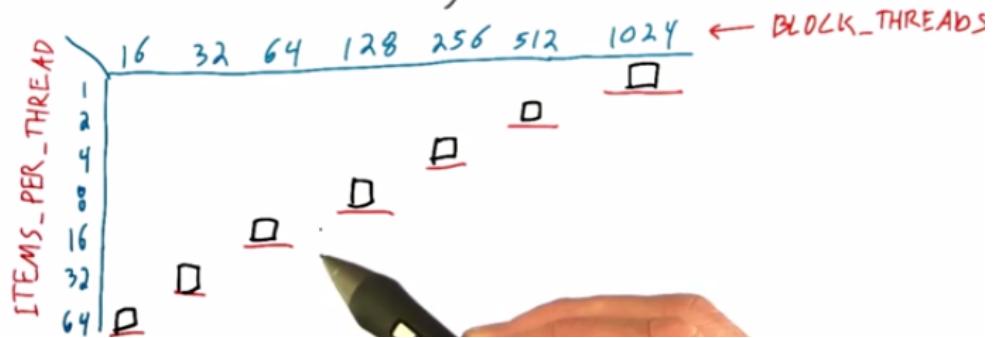
code available:

<https://github.com/udacity/cs344/blob/master/Unit7%20Code%20Snippets/cub/>

### L7.1-7.27-CUB Exercise Continued

#### Programming exercise - CUB

Simple example of `cub::blockscan` - 1 block scan  
measure "scan throughput" in clocks / element scanned.



### L7.1-7.28-Other Parallel Computing Platforms

#### Other Platforms

"CUDA" ≠ CUDA C/C++

Other languages targeting CUDA

Cross-platform solutions

## Other Languages

Wrap CUDA C++ → PyCUDA

Target CUDA directly

Copperhead — data-parallel subset of Python

- map, reduce, filter, scan, sort

- generates Thrust code

- interoperate with e.g. numpy, matplotlib

## Other Languages

Wrap CUDA C++ → PyCUDA

Target CUDA directly

Copperhead — data-parallel subset of Python

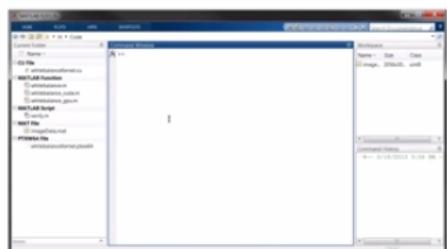
CUDA Fortran — Fortran with CUDA constructs

Halide — image processing DSL

Matlab

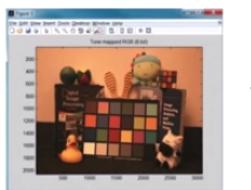
### L7.1-7.29-Matlab

## MATLAB

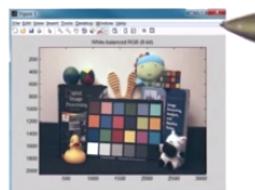


- Algorithm development
- Data analysis & viz
- Math modeling
- GPU computing

white balance



before



after

## L7.1-7.30-Matlab Demo

The image shows three MATLAB code editors side-by-side, each containing a version of the `whitebalance` function.

- Editor - H:\Code\whitebalance.m:**

```
1 function adjustedImage = whitebalance(imageData)
2 % WHITEBALANCE forces the average image color to be gray
3
4 % Find the average values for each channel
5 pageSize = size(imageData,1) * size(imageData,2);
6 avg_rgb = mean( reshape(imageData, [pageSize,3]) );
7
8 % Find the average gray value and compute the scaling array
9 avg_all = mean(avg_rgb);
10 scaleArray = max(avg_all, 128)./avg_rgb;
11 scaleArray = reshape(scaleArray,1,1,3);
12
13 % Adjust the image to the new gray value
14 adjustedImage = uint8(bsxfun(@times,double(imageData),scaleArray));
15 end
```
- Editor - H:\Code\whitebalance\_gpu.m:**

```
1 function adjustedImage = whitebalance_gpu(imageData)
2 % WHITEBALANCE forces the average image color to be gray.
3
4 % Find the average values for each channel
5 pageSize = size(imageData,1) * size(imageData,2);
6 avg_rgb = mean( reshape(imageData, [pageSize,3]) );
7
8 % Find the average gray value and compute the scaling factor
9 avg_all = mean(avg_rgb);
10 factor = max(avg_all, 128)./avg_rgb;
11 factor = reshape(factor,1,1,3);
12
13 % Adjust the image to the new gray value
14 imageData = gpuArray(imageData);
15 adjustedImage = uint8(bsxfun(@times,double(imageData),factor));
16 end
```
- Editor - H:\Code\whitebalance\_cuda.m:**

```
1 function adjustedImage = whitebalance_cuda(imageData)
2 % WHITEBALANCE forces the average image color to be gray
3
4 % Find the average values for each channel
5 pageSize = size(imageData,1) * size(imageData,2);
6 avg_rgb = mean( reshape(imageData, [pageSize,3]) );
7
8 % Find the average gray value and compute the scaling factor
9 avg_all = mean(avg_rgb);
10 factor = max(avg_all, 128)./avg_rgb;
11 factor = reshape(factor,1,1,3);
12
13 % Load the kernel and set up threads
14 kernel = parallel.gpu.CUDAKernel('whitebalanceKernel.ptxw64',...
15                                'whitebalanceKernel.cu' );
16 [nRows, nCols,~] = size(imageData);
17 blockSize = 256;
18 kernel.ThreadBlockSize = [blockSize, 1, 3];
19 kernel.GridSize = [ceil(nRows/blockSize), nCols];
20
```

```


6 - avg_rgb = mean( reshape(imageData, [pageSize,3]) );
7 -
8 % Find the average gray value and compute the scaling factor
9 - avg_all = mean(avg_rgb);
10 - factor = max(avg_all, 128)./avg_rgb;
11 - factor = reshape(factor,1,1,3);
12 -
13 % Load the kernel and set up threads
14 - kernel = parallel.gpu.CUDAKernel('whitebalanceKernel.ptxw64',...
15 %'whitebalanceKernel.cu' );
16 - [nRows, nCols,~] = size(imageData);
17 - blockSize = 256;
18 - kernel.ThreadBlockSize = [blockSize, 1, 3];
19 - kernel.GridSize = [ceil(nRows/blockSize), nCols];
20 -
21 % Copy image data to the GPU and allocate and initialize return data
22 - imageDataGPU = gpuArray(imageData);
23 - adjustedImage = gpuArray.zeros( size(imageData), 'uint8' );
24 -
25 % Adjust the image to the new gray value
26 - adjustedImage = feval( kernel, adjustedImage, imageDataGPU, ...
27 % factor, nRows, nCols );
28 - end


```

Course Codes



- High-level programming language
- More than 1,000,000 people use MATLAB
- Easily design/test algorithms for GPUs
  - `gpuArray` command
  - Interface to CUDA kernels

#### L7.1-7.31-Cross Platform Solutions

Cross - Platform solutions

GPGUs, multicore CPUs  
NVIDIA, AMD, Intel, Arm ...

OpenCL ————— Similar to CUDA

Open GL Compute - Integrated w/ OpenGL graphics

Open ACC — Directives
 

- Great for legacy code
- evolution of OpenMP
- C, C++, Fortran

## L7.1-7.32-Summary

### Summary

- Parallel optimization patterns
- Libraries
- Programming power tools
- Other platforms
  - languages
  - cross-platform solutions
- Dynamic Parallelism      Dr. Stephen Jones, NVIDIA