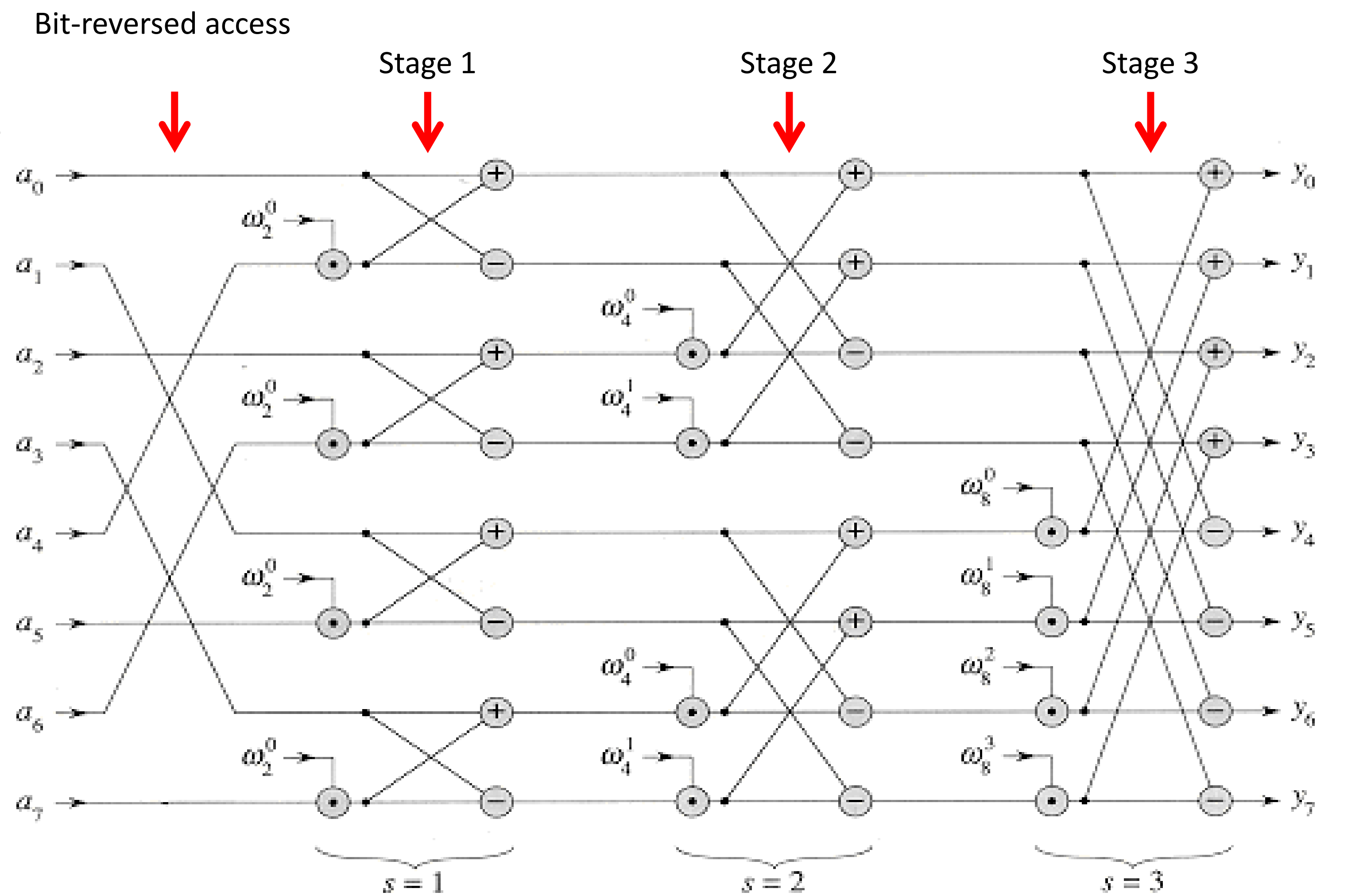
CS 179: GPU Programming

Lecture 9 / Homework 3

Recap

- Some algorithms are “less obviously parallelizable”:
  - Reduction
  - Sorts
  - FFT (and certain recursive algorithms)

Parallel FFT structure (radix-2)



http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap32.htm

cuFFT 1D example

```
#define NX 262144

cufftComplex *data_host
        = (cufftComplex*)malloc(sizeof(cufftComplex)*NX);
cufftComplex *data_back
        = (cufftComplex*)malloc(sizeof(cufftComplex)*NX);


// Get data...


cufftHandle plan;
cufftComplex *data1;
cudaMalloc((void**)&data1, sizeof(cufftComplex)*NX);
cudaMemcpy(data1, data_host, NX*sizeof(cufftComplex), cudaMemcpyHostToDevice);

/* Create a 1D FFT plan. */
int batch = 1;  // Number of transforms to run
cufftPlan1d(&plan, NX, CUFFT_C2C, batch);

/* Transform the first signal in place. */
cufftExecC2C(plan, data1, data1, CUFFT_FORWARD);


/* Inverse transform in place. */
cufftExecC2C(plan, data1, data1, CUFFT_INVERSE);

cudaMemcpy(data_back, data1, NX*sizeof(cufftComplex), cudaMemcpyDeviceToHost);
```

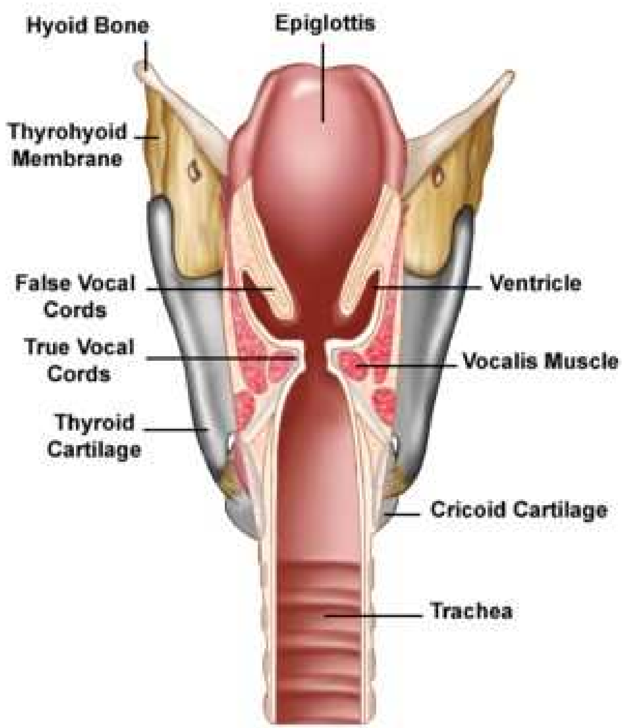
Correction:
Remember to use
cufftDestroy(plan)
when finished with
transforms

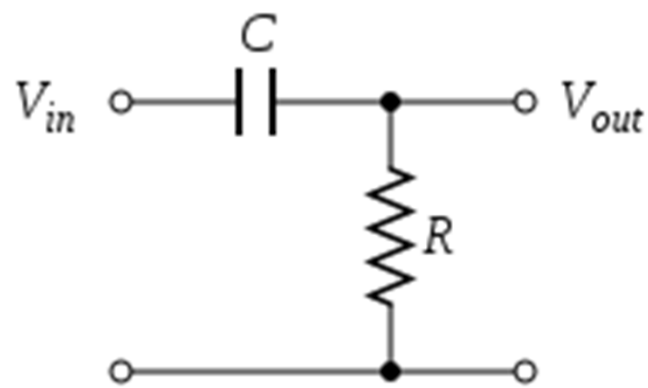
Today

- Homework 3
  - Large-kernel convolution
- Project Introductions

Systems

- Given input signal(s), produce output signal(s)





LTI system review (Week 1)

- “Linear time-invariant” (LTI) systems
  - Lots of them!

- Can be characterized entirely by “impulse response” $h[n]$

- Output given from input by *convolution*:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Parallelization

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

- Convolution is parallelizable!
  - Sequential pseudocode (ignoring boundary conditions):

```
(set all y[i] to 0)
For (i from 0 through x.length - 1)
        for (j from 0 through h.length - 1)
                y[i] += (appropriate terms from x and h)
```

A problem…

- This worked for *small* impulse responses
  - E.g. $h[n]$, $0 \le n \le 20$ in HW 1

- Homework 1 was “small-kernel convolution”:
  - (Vocab alert: Impulse responses are often called “kernels”!)

A problem…

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

- Sequential runtime: O(n*m)
  - (n: size of x)
  - (m: size of h)
  - Troublesome for large m! (i.e. large impulse responses)

```
(set all y[i] to 0)
For (i from 0 through x.length - 1)
        for (j from 0 through h.length - 1)
                y[i] += (appropriate terms from x and h)
```

DFT/FFT

- Same problem with Discrete Fourier Transform!

$$X_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n/N}, \quad k \in \mathbb{Z}$$

- Successfully optimized *and* GPU-accelerated!
  - $O(n^2)$ to $O(n \log n)$
  - Can we do the same here?

“Circular” convolution

“Circular” convolution

- Linear convolution:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k]$$

- Circular convolution:

$$y[n] = \sum_{k=0}^{N-1} x[k]\, h[(n-k)\, mod\, N]$$

Example:

- x[0..3], h[0..1]

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

- Linear convolution:

  y[0] = x[0]h[0]
  y[1] = x[0]h[1] + x[1]h[0]
  y[2] = x[1]h[1] + x[2]h[0]
  y[3] = x[2]h[1] + x[3]h[0]
  y[4] = x[3]h[1] + x[4]h[0]

$$y[n] = \sum_{k=0}^{N-1} x[k]\, h[(n-k)\, mod\, N]$$

- Circular convolution:

  y[0] = x[0]h[0] + x[3]h[1] + x[2]h[2] + x[3]h[1]
  y[1] = x[0]h[1] + x[1]h[0] + x[2]h[3] + x[3]h[2]
  y[2] = x[1]h[1] + x[2]h[0] + x[3]h[3] + x[0]h[2]
  y[3] = x[2]h[1] + x[3]h[0] + x[0]h[3] + x[1]h[2]

  = 0

Circular Convolution Theorem*

$$y[n] = \sum_{k=0}^{N-1} x[k]\, h[(n-k)\ mod\ N]$$

- Can be calculated by: IFFT(FFT(x) .* FFT(h))
- i.e.

$$\vec{X} = FFT(\vec{x})$$
$$\vec{H} = FFT(\vec{h})$$

  – For all i:

$$Y_i = X_i H_i$$

  – Then:

$$\vec{y} = IFFT(\vec{Y})$$

* DFT case

Circular Convolution Theorem*

$$y[n] = \sum_{k=0}^{N-1} x[k]\, h[(n-k)\ mod\ N]$$

- Can be calculated by: IFFT(FFT(x) .* FFT(h))
- i.e.

$$\vec{X} = FFT(\vec{x})$$ O(n log n) Assume n > m

$$\vec{H} = FFT(\vec{h})$$ O(m log m)

  - For all i:

$$Y_i = X_i H_i$$ O(n)

Total: O(n log n)

  - Then:

$$\vec{y} = IFFT(\vec{Y})$$ O(n log n)

* DFT case

- $x[n]$ and $h[n]$ are different lengths?

- How to linearly convolve using circular convolution?

Padding

- x[n] and h[n] – presumed zero where not defined
  - Computationally: Store x[n] and h[n] as larger arrays
  - Pad both to at least x.length + h.length - 1

Example: (Padding)

- x[0..3], h[0..1]

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

- Linear convolution:

  y[0] = x[0]h[0]
  y[1] = x[0]h[1] + x[1]h[0]
  y[2] = x[1]h[1] + x[2]h[0]
  y[3] = x[2]h[1] + x[3]h[0]
  y[4] = x[3]h[1] + x[4]h[0]

N is now (4 + 2 – 1) = 5

- Circular convolution:

$$y[n] = \sum_{k=0}^{N-1} x[k]\, h[(n-k) \bmod N]$$

  y[0] = x[0]h[0] + x[1]h[4] + x[2]h[3] + x[3]h[2] + x[4]h[1]
  y[1] = x[0]h[1] + x[1]h[0] + x[2]h[4] + x[3]h[3] + x[4]h[2]
  y[2] = x[1]h[1] + x[2]h[0] + x[3]h[4] + x[4]h[3] + x[0]h[2]
  y[3] = x[2]h[1] + x[3]h[0] + x[4]h[4] + x[0]h[3] + x[1]h[2]
  y[4] = x[3]h[1] + x[4]h[0] + x[0]h[4] + x[1]h[3] + x[2]h[2]

Summary

- Alternate algorithm for large impulse response convolution!
  - Serial: O(n log n) vs. O(mn)
    - Small vs. large m determines algorithm choice
    - Runtime does “carry over” to parallel situations (to some extent)

Homework 3, Part 1

- Implement FFT ("large-kernel") convolution
  - Use cuFFT for FFT/IFFT (if brave, try your own)
    - ~~Use "batch" variable to save FFT calculations~~

      Correction: Good practice in general, but results in poor performance on Homework 3
  - Complex multiplication kernel: Week 1-style
  - (HW1 difference: Consider right-hand boundary region)

Complex numbers

- cufftComplex: cuFFT complex number type
  - Example usage:

```
cufftComplex a;
a.x = 3;        // Real part
a.y = 4;        // Imaginary part
```

- Element-wise multiplying:

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

Homework 3, Part 2

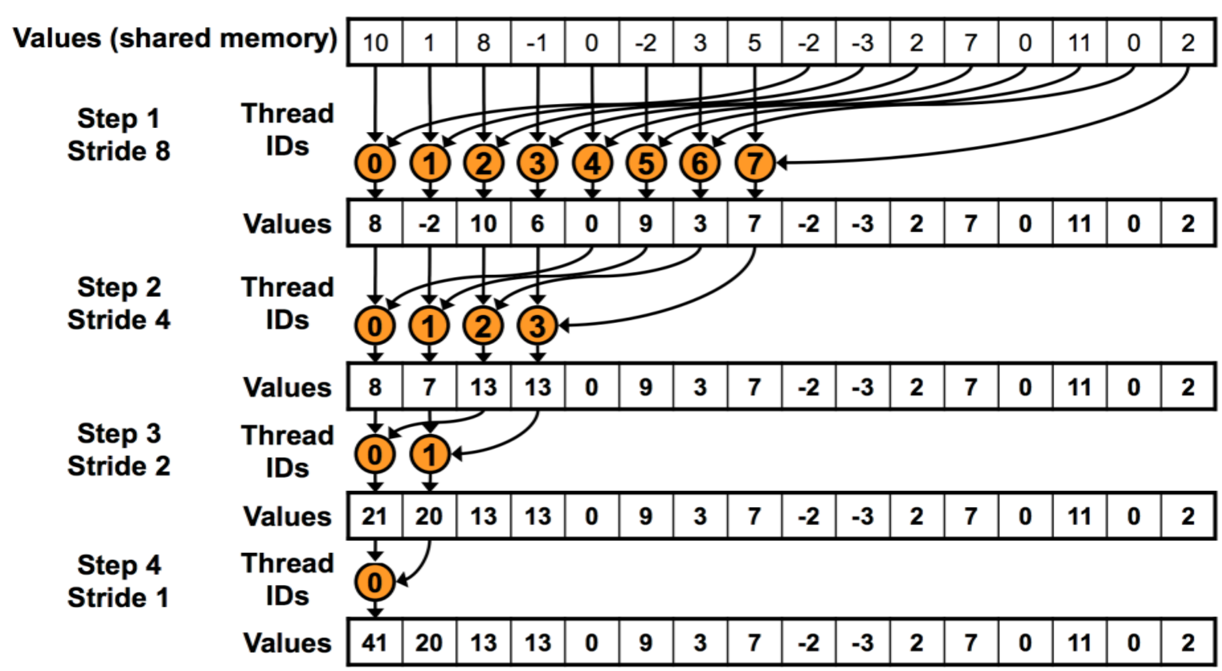
Normalization

- Amplitudes must lie in range [-1, 1]
  - Normalize s.t. maximum magnitude is 1 (or 1 - ε)

- How to find maximum amplitude?

Reduction

- This time, maximum (instead of sum)
  - Lecture 7 strategies
  - “Optimizing Parallel Reduction in CUDA” (Harris)



Homework 3, Part 2

- Implement GPU-accelerated normalization
  - Find maximum (reduction)
  - Divide by maximum to normalize

(Demonstration)

- Rooms can be modeled as LTI systems!

Other notes

- Machines:
  - Normal mode: haru, mx, minuteman
  - Audio mode: haru
- Due date: Friday (4/24), ~~3 PM~~

  Correction: 11:59 PM
  - Extra office hours: Thursday (4/23), 8-10 PM

Projects