CS 179: GPU Computing

Lecture 3 / Homework 1

Recap

- Adding two arrays… a close look
  - Memory:
    - Separate memory space, cudaMalloc(), cudaMemcpy(), …
  - Processing:
    - Groups of threads (grid, blocks, warps)
    - Optimal parameter choice (#blocks, #threads/block)
  - Kernel practices:
    - Robust handling of workload (beyond 1 thread/index)

Parallelization

- What are parallelizable problems?

Parallelization

- What are parallelizable problems?

- e.g.
  - Simple shading:

```
for all pixels (i,j):
        replace previous color with new color
        according to rules
```

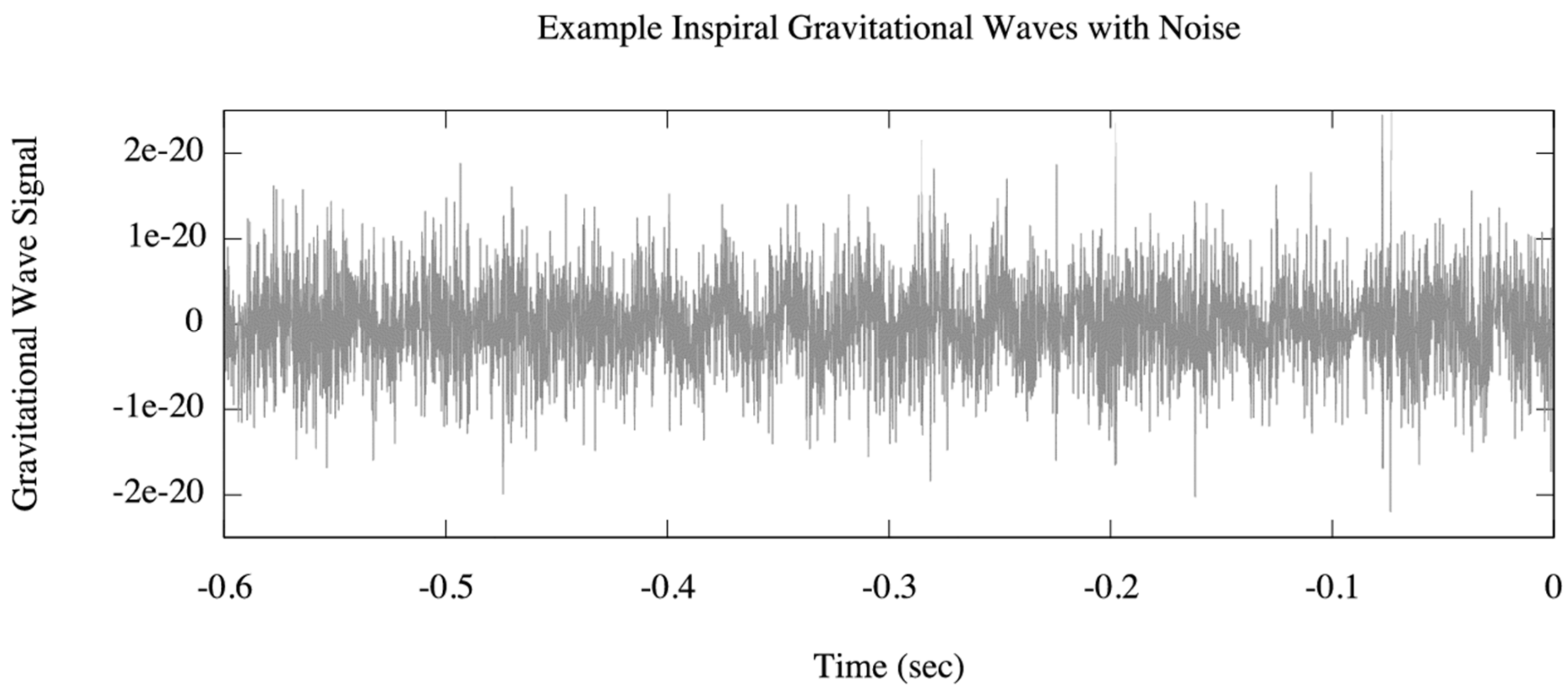
  - Adding two arrays:

```
for (int i = 0; i < N; i++)
        C[i] = A[i] + B[i];
```

Parallelization

- What *aren't* parallelizable problems?
  - Subtle differences!

Moving Averages



http://www.ligo.org

Moving Averages

Simple Moving Average

- $x[n]$: input (the raw signal)
- $y[n]$: simple moving average of $x[n]$

- Each point in $y[n]$ is the average of the last K points!

Simple Moving Average

- x[n]: input (the raw signal)
- y[n]: simple moving average of x[n]

- Each point in y[n] is the average of the last K points!
  - For all n ≥ K:

$$y[n] = x[n] + x[n-1] + \cdots + x[n-(K-1)]$$

Exponential Moving Average

- Each point in y[n] follows the relation:

$$y[0] = x[0]$$
$$y[n] = c \cdot x[n] + (1 - c) \cdot y[n-1], 0 \leq c \leq 1$$

- "Exponential" – can expand recurrence relation:

$$y[n] = c \cdot (\, x[n] + (1-c) \cdot x[n-1] + (1-c)^2 x[n-2] + \cdots + (1-c)^{n-1} x[1]\,)$$
$$+ (1-c)^n x[0]$$

- Each point in x[n] has an (exponentially) decaying influence!


GBPCAD, h1 Spotware cTrader 12:05:49 (UTC+0) on Jul 15th, 2013
Simple Moving Average (14)
Exponential Moving Average (14)
100 pips
1.61250
1.61000
1.60750
1.60500
1.60250
1.60000
1.59750
1.59500
1.59250
1.59000
1.58750
1.58500
1.58250
1.58000
1.57750
1.57500
1.57250
1.57000
03 Jul 2013, UTC+3
16:00
04 Jul
08:00
16:00
05 Jul
08:00
16:00
08 Jul 01:00
54:12

Comparison

- Simple moving average:

$$y[n] = x[n] + x[n-1] + \cdots + x[n-(K-1)]$$

  – Easily parallelizable?

- Exponential moving average:

$$y[n] = c \cdot x[n] + (1-c) \cdot y[n-1]$$

  – Easily parallelizable?

Comparison

- Simple moving average:

$$y[n] = x[n] + x[n-1] + \cdots + x[n-(K-1)]$$

  – Easily parallelizable? Yes

- Exponential moving average:

$$y[n] = c \cdot x[n] + (1-c) \cdot y[n-1]$$

  – Easily parallelizable? Not so much

Comparison

- Simple moving average:

$$y[n] = x[n] + x[n-1] + \cdots + x[n-(K-1)]$$

  – Easily parallelizable? Yes

Calculation for y[n] depends on calculation for y[n-1] !

- Exponential moving average:

$$y[n] = c \cdot x[n] + (1-c) \cdot y[n-1]$$

  – Easily parallelizable? Not so much

Comparison

- SMA pseudocode:

```
for i = 0 through N-1
        y[n] <- x[n] + ... + x[n-(K-1)]
```

- EMA pseudocode:

```
for i = 0 through N-1
        y[n] <- c*x[n] + (1-c)*y[n-1]
```

  - Loop iteration i depends on iteration i-1 !
  - Far less parallelizable!

Comparison

- SMA pseudocode:

```
for i = 0 through N-1
        y[n] <- x[n] + ... + x[n-(K-1)]
```

  - Better GPU-acceleration

- EMA pseudocode:

```
for i = 0 through N-1
        y[n] <- c*x[n] + (1-c)*y[n-1]
```

  - Loop iteration i depends on iteration i-1 !
  - Far less parallelizable!
  - Worse GPU-acceleration

Morals

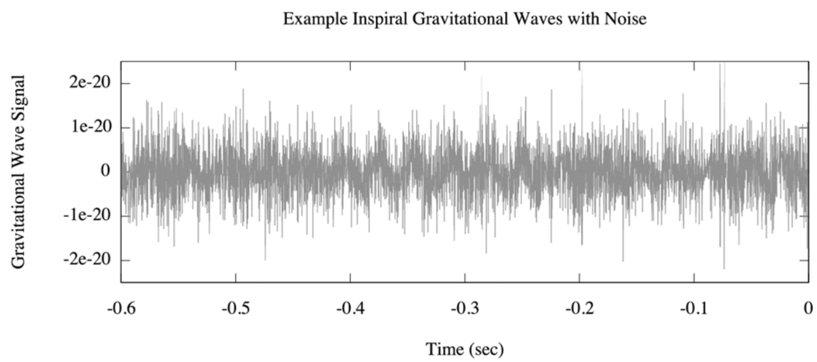
- Not all problems are parallelizable!
  - Even similar-looking problems

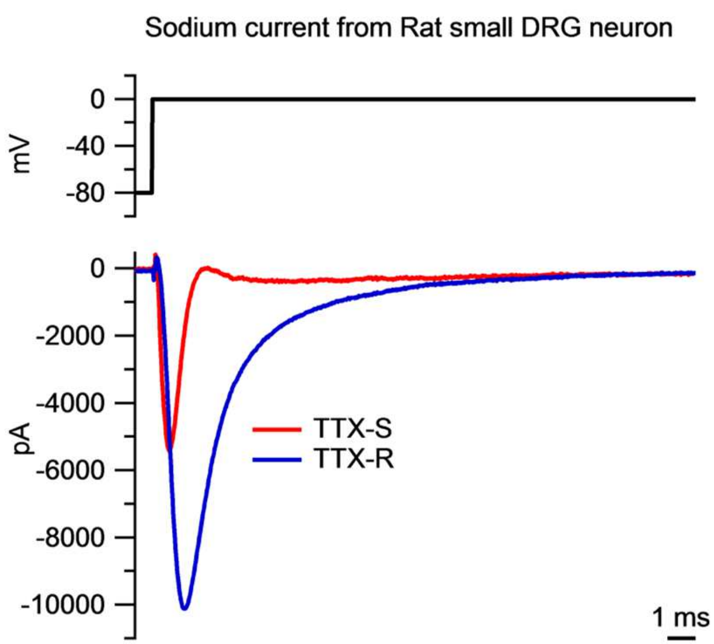
- Recall: *Parallel* algorithms have potential in GPU computing

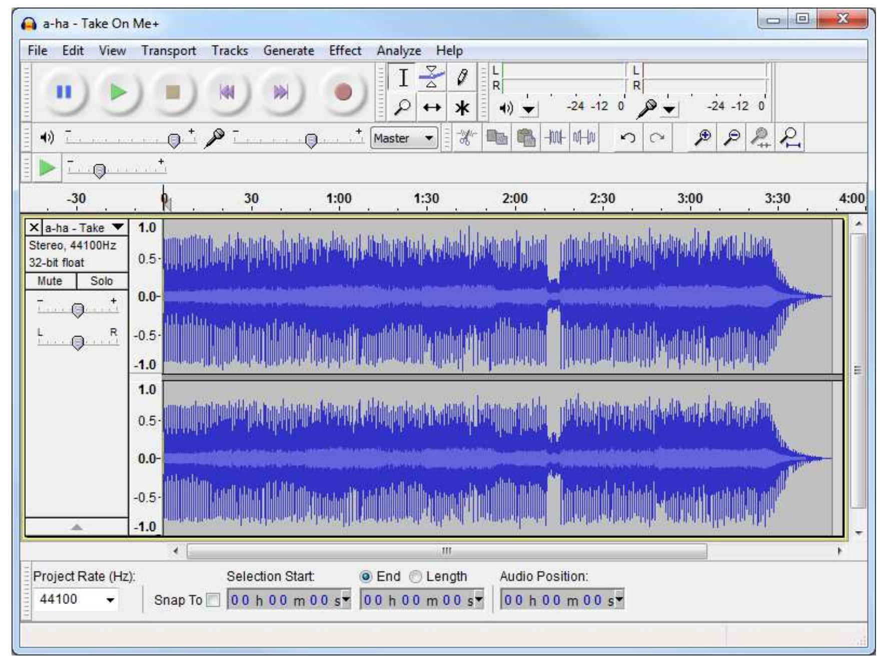
Small-kernel convolution

Homework 1 (coding portion)

Signals

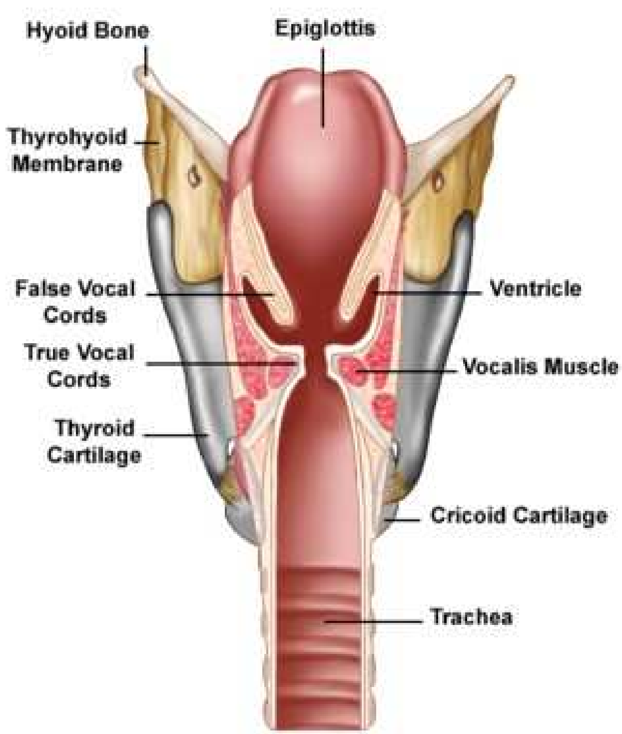

Example Inspiral Gravitational Waves with Noise
Gravitational Wave Signal
2e-20
1e-20
0
-1e-20
-2e-20
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
Time (sec)

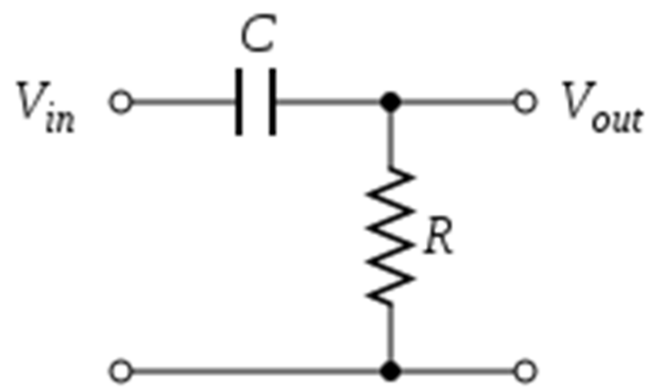

Sodium current from Rat small DRG neuron
mV
0
-40
-80
pA
0
-2000
-4000
-6000
-8000
-10000
TTX-S
TTX-R
1 ms


a-ha - Take On Me+
File
Edit
View
Transport
Tracks
Generate
Effect
Analyze
Help
Master
-30
0
30
1:00
1:30
2:00
2:30
3:00
3:30
4:00
a-ha - Take
Stereo, 44100Hz
32-bit float
Mute
Solo
1.0
0.5
0.0
-0.5
-1.0
Project Rate (Hz):
44100
Snap To
Selection Start:
End
Length
Audio Position:

Systems

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





Discretization

- Discrete samplings
- of continuous signals
  – Continuous audio signal -> WAV file
  – Voltage -> Voltage every T milliseconds

- (Will focus on discrete-time signals here)

Linear systems

- If system has:

$$x_1[n] \rightarrow y_1[n]$$
$$x_2[n] \rightarrow y_2[n]$$

- Then (for constants *a*, *b*):

$$ax_1[n] + bx_2[n] \rightarrow ay_1[n] + by_2[n]$$

Linear systems

- Consider a *tiny piece* of the signal

Linear systems

- Consider a *tiny piece* of the signal…

- Delta function:

$$\delta[n-k] = \begin{cases} 1, n = k \\ 0, n \neq k \end{cases}$$

- "Signal at a point" *k*:

$$x[k]\delta[n-k]$$

Linear systems

- If we know that:

$$\delta[n-k] \rightarrow h_k[n]$$

- Then, by linearity:

$$x[k]\delta[n-k] \rightarrow x[k]h_k[n]$$

- Response at time *k* defined by response to delta function!

Time-invariance

- If:

$$x[n] \rightarrow y[n]$$

- Then (for integer m):

$$x[n+m] \rightarrow y[n+m]$$

Time-invariance

- If system has:

$$\delta[n-k] \rightarrow h_k[n]$$
$$\delta[n-l] \rightarrow h_l[n]$$

- Then $h_k[n]$ and $h_l[n]$ are time-shifted versions of each other!

Time-invariance and linearity

- Define $h[n]$ as the *impulse response* to delta function:

$$\delta[n] \to h[n]$$

- Then:

$$\delta[n-k] \to h[n-k]$$

- And by linearity:

$$x[k]\delta[n-k] \to x[k]h[n-k]$$

Time-invariance and linearity

- Can write our original signal as:

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

- Then, since (*last slide*):

$$x[k]\delta[n-k] \rightarrow x[k]h[n-k]$$

- By linearity:

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

Morals

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

- Can be characterized entirely by $h[n]$

- Output given from input by:

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

Convolution example

- Suppose we have input $x[0..99]$, system given by $h[0..3]$

- Example output value:

$$y[50] = x[47]h[3] + x[48]h[2] + x[49]h[1] + x[50]h[0]$$

Computability

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

- For *finite-duration* $h[n]$, sum is computable with this formula
  - Computed for finite $x[n]$, e.g. audio file
- Sum 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)
```

This assignment

- Accelerate this computation!
  - Fill in TODOs on assignment 1
    - Kernel implementation
    - Memory operations
  - We give the skeleton:
    - CPU implementation (a good reference!)
    - Output error checks
    - h[n] (default is Gaussian impulse response)
    - …

The code

- Framework code has two modes:
  - Normal mode (AUDIO_ON zero)
    - Generates random x[n]
    - Can run performance measurements on different sizes of x[n]
    - Can run multiple repeated trials (adjust channels parmeter)
  - Audio mode (AUDIO_ON nonzero)
    - Reads input WAV file as x[n]
    - Outputs y[n] to WAV
    - Gaussian is an imperfect *low-pass* filter – high frequencies attenuated!

Demonstration

Debugging tips

- Printf
  - Beware – you have many threads!
  - Set small number of threads to print
- Store intermediate results in global memory
  - Can copy back to host for inspection
- Check error returns!
  - gpuErrchk macro included – wrap around function calls

Debugging tips

- Use small convolution test case
  - E.g. 5-element x[n], 3-element h[n]

Compatibility

- Our machines:
  - haru.caltech.edu
  - (We'll try to get more up as the assignment progresses)
- CMS machines:
  - Only normal mode works
    - (Fine for this assignment)
- Your own system:
  - Dependencies: libsndfile (audio mode)

Administrivia

- Due date:
  - Wednesday, 3 PM (correction)
- Office hours (ANB 104):
  - Kevin/Andrew: Monday, 9-11 PM
  - Eric: Tuesday, 7-9 PM