AM 148 Lecture 6

Steven Reeves

University of California, Santa Cruz

sireeves@ucsc.edu

May 3, 2018

Overview

1. Histogram
   - Atomics
   - Generating Color Distributions from an image
2. Segmented Scan
3. Sort
4. Sparse Matrix Vector Product

What is a Histogram?

- Histogram gives a representation of the distribution of numerical data

What is a Histogram?

- Histogram gives a representation of the distribution of numerical data
- Its an estimate of the true PDF

What is a Histogram?

- Histogram gives a representation of the distribution of numerical data
- Its an estimate of the true PDF
- Based on *binning*
- Algorithm classifies data based on a bin and collects the binned data

How can we make one?

A serial implementation is straightforward

```
void histogram(unsigned int *histo, type *measurements, int
    bin_count, int array_length)
{
        for(int i = 0; i < bin_count;  i++)
                histo[i] = 0;
        for(int i = 0; i <array_length; i++)
                histo[computeBin(measurements[i])]++;
}
```

Here your data could be integers, floats, or strings e.g. "Green", "Blue", etc

First Try Parallel Histogram

```
__global__ void first_hist(unsigned int *histo, type *data,
    int n)
{
        int tid = threadIdx.x + blockDim.x*blockIdx.x;
        if(tid > n)
                return;

        histo[computeBin(data[tid])]++;
//Where computeBin is a __device__ function!
}
```

Why didn't this work?

- The source of our issues is with the `d_bins[myBin]++` line

Why didn't this work?

- The source of our issues is with the `d_bins[myBin]++` line
- What is going on with this line?
  1. Read bin Value from global memory to a register
  2. Increment Bin Value
  3. Write Bin Value from register to global memory

Why didn't this work?

- The source of our issues is with the `d_bins[myBin]++` line
- What is going on with this line?
  1. Read bin Value from global memory to a register
  2. Increment Bin Value
  3. Write Bin Value from register to global memory
- Race condition!

Fixing the issue

- Atomics are built in CUDA pragmas that serialize memory transactions/operations
- Locks a locale in memory so that no other thread can read/write to it until current thread is done
- We'll use `atomicAdd`

Issue with Atomics

- Serialized access to memory location

Issue with Atomics

- Serialized access to memory location
- Creates a performance bottleneck

Thought experiment

Lets suppose we have 1 million measurements which we wish to create a histogram for. Using the atomic method which will be fastest?

Thought experiment

Lets suppose we have 1 million measurements which we wish to create a histogram for. Using the atomic method which will be fastest?

1. A histogram with 10 bins

Thought experiment

Lets suppose we have 1 million measurements which we wish to create a histogram for. Using the atomic method which will be fastest?

1. A histogram with 10 bins
2. A histogram with 100 bins

Thought experiment

Lets suppose we have 1 million measurements which we wish to create a histogram for. Using the atomic method which will be fastest?

1. A histogram with 10 bins
2. A histogram with 100 bins
3. A histogram with 1000 bins

A Shared Memory Approach

- Create block local histogram
- Use atomics

A Shared Memory Approach

- Create block local histogram
- Use atomics
- Combines block local histograms into global histogram

Application of Histogram

- Generating Color distributions of an image

Application of Histogram

- Generating Color distributions of an image
- Generating frequency/probability distributions from raw data

Code

Segmented Scan

- Sometimes we don't wish to do a full scan on an array

Segmented Scan

- Sometimes we don't wish to do a full scan on an array
- Launching many separate scans is inefficient

Segmented Scan

- Sometimes we don't wish to do a full scan on an array
- Launching many separate scans is inefficient
- Combine Arrays as segments, use a flagging array to mark segments.

Exclusive sum scan:

$$(1, 2, 3, 4, 5, 6, 7, 8,) \implies (0, 1, 3, 6, 10, 15, 21, 28)$$

Exclusive sum scan:

$$(1, 2, 3, 4, 5, 6, 7, 8,) \implies (0, 1, 3, 6, 10, 15, 21, 28)$$

$$(1, 2|3, 4, 5|6, 7, 8) \implies (0, 1|0, 3, 7|0, 6, 13)$$

using

$$(1, 0, 1, 0, 0, 1, 0, 0)$$

Compact

Before we begin a sort, let's talk about an algorithm called compact

Compact

Before we begin a sort, let's talk about an algorithm called compact

- Compact is an algorithm to partition data

Compact

Before we begin a sort, let's talk about an algorithm called compact

- Compact is an algorithm to partition data
- Input data $\rightarrow$ smaller partition of input data
- Compacting the larging input set into something smaller

Compact

Before we begin a sort, let's talk about an algorithm called compact

- Compact is an algorithm to partition data
- Input data $\rightarrow$ smaller partition of input data
- Compacting the larging input set into something smaller
- If we only want to do computation on a subset of data

Compact Continued

- Input

$$[S_0, S_1, S_2, S_3, S_4, \ldots]$$

Compact Continued

- Input

$$[S_0, S_1, S_2, S_3, S_4, \dots]$$

- Predicate (is my index even for example)

$$[T, F, T, F, T, \dots]$$

Compact Continued

- Input

$$[S_0, S_1, S_2, S_3, S_4, \dots]$$

- Predicate (is my index even for example)

$$[T, F, T, F, T, \dots]$$

- Output

$$S_0, S_2, S_4, \dots$$

Compact Continued

- Input

$$[S_0, S_1, S_2, S_3, S_4, \ldots]$$

- Predicate (is my index even for example)

$$[T, F, T, F, T, \ldots]$$

- Output

$$S_0, S_2, S_4, \ldots$$

- To generate the output we need to compute the *scatter address* of each output element

Compact In Parallel

- To Compact in parallel we need to compute scatter addresses

Compact In Parallel

- To Compact in parallel we need to compute scatter addresses
- Given this set of predicates

$$[T, F, F, T, T, F, T, F]$$

- we compute addresses

$$[0-, -, 1, 2, -, 3, -]$$

Compact In Parallel

- To Compact in parallel we need to compute scatter addresses
- Given this set of predicates

$$[T, F, F, T, T, F, T, F]$$

- we compute addresses

$$[0-, -, 1, 2, -, 3, -]$$

- Change predicates

$$[1, 0, 0, 1, 1, 0, 1, 0]$$

And generate

$$[0, 1, 1, 1, 2, 3, 3, 4]$$

Compact In Parallel

- To Compact in parallel we need to compute scatter addresses
- Given this set of predicates

$$[T, F, F, T, T, F, T, F]$$

- we compute addresses

$$[0-, -, 1, 2, -, 3, -]$$

- Change predicates

$$[1, 0, 0, 1, 1, 0, 1, 0]$$

And generate

$$[0, 1, 1, 1, 2, 3, 3, 4]$$

- This is a Scan operation!

Sorting an array

- Most sorts are serial algorithms!

Sorting an array

- Most sorts are serial algorithms!
- We need to find efficient Parallel Algoirthms!
  - Keep Hardware busy (lots of threads)

Sorting an array

- Most sorts are serial algorithms!
- We need to find efficient Parallel Algoirthms!
    - Keep Hardware busy (lots of threads)
    - Limit thread divergence

Sorting an array

- Most sorts are serial algorithms!
- We need to find efficient Parallel Algoirthms!
  - Keep Hardware busy (lots of threads)
  - Limit thread divergence
  - Prefer Coalesced Memory Access

Radix Sort

Radix sort relies on sorting using the binary notation of a number. Here are a number of steps for a basic Radix sort algorithm:

1. Start with least signicant bit

Radix Sort

Radix sort relies on sorting using the binary notation of a number. Here are a number of steps for a basic Radix sort algorithm:

1. Start with least signicant bit
2. Split Input into 2 sets based on bit, otherwise preserve order

Radix Sort

Radix sort relies on sorting using the binary notation of a number. Here are a number of steps for a basic Radix sort algorithm:

1. Start with least signicant bit
2. Split Input into 2 sets based on bit, otherwise preserve order
3. Move to next most significant bit, rinse and repeat.

Radix Sort Example

Lets suppose we have the following array of unsigned integers.

$$[0, 5, 2, 7, 1, 3, 6, 4] \implies [000, 101, 010, 111, 001, 011, 110, 100]$$

Radix Sort Example

Lets suppose we have the following array of unsigned integers.

$$[0, 5, 2, 7, 1, 3, 6, 4] \implies [000, 101, 010, 111, 001, 011, 110, 100]$$

group least significant bit

$$[000, 010, 110, 100, 101, 111, 001, 011]$$

Radix Sort Example

Lets suppose we have the following array of unsigned integers.

$$[0, 5, 2, 7, 1, 3, 6, 4] \implies [000, 101, 010, 111, 001, 011, 110, 100]$$

group least significant bit

$$[000, 010, 110, 100, 101, 111, 001, 011]$$

move to next bit

$$[000, 100, 101, 001, 010, 110, 111, 011]$$

Radix Sort Example

Lets suppose we have the following array of unsigned integers.

$$[0, 5, 2, 7, 1, 3, 6, 4] \implies [000, 101, 010, 111, 001, 011, 110, 100]$$

group least significant bit

$$[000, 010, 110, 100, 101, 111, 001, 011]$$

move to next bit

$$[000, 100, 101, 001, 010, 110, 111, 011]$$

finally

$$[000, 001, 010, 011, 100, 101, 110, 111] \implies [0, 1, 2, 3, 4, 5, 6, 7]$$

Underlying Primitives

- Work Complexity of Radix Sort is $\mathcal{O}(kn)$ where k is number of bits

Underlying Primitives

- Work Complexity of Radix Sort is $\mathcal{O}(kn)$ where k is number of bits
- Grouping bits is a compact algorithm

Underlying Primitives

- Work Complexity of Radix Sort is $\mathcal{O}(kn)$ where k is number of bits
- Grouping bits is a compact algorithm
- Predicate `(i&1)==0` (or the opposite)

Underlying Primitives

- Work Complexity of Radix Sort is $\mathcal{O}(kn)$ where k is number of bits
- Grouping bits is a compact algorithm
- Predicate `(i&1)==0` (or the opposite)
- Exclusive scan over the predicate to give scatter addresses of zero bit

Underlying Primitives

- Work Complexity of Radix Sort is $\mathcal{O}(kn)$ where k is number of bits
- Grouping bits is a compact algorithm
- Predicate `(i&1)==0` (or the opposite)
- Exclusive scan over the predicate to give scatter addresses of zero bit
- Then (inclusive) scan over the one bit predicates added with the last address of zero bits

Underlying Primitives

- Work Complexity of Radix Sort is $\mathcal{O}(kn)$ where k is number of bits
- Grouping bits is a compact algorithm
- Predicate `(i&1)==0` (or the opposite)
- Exclusive scan over the predicate to give scatter addresses of zero bit
- Then (inclusive) scan over the one bit predicates added with the last address of zero bits
- This algorithm can be optimized by increasing the number of bits per compaction (more subsets)

CUDA Example

Sparse Matrices

- Sparse Matrices are Matrices with a majority of entries with value 0.
- Dense Matrices are Matrices with little to no 0 entries

Sparse Matrices

- Sparse Matrices are Matrices with a majority of entries with value 0.
- Dense Matrices are Matrices with little to no 0 entries
- We have done Dense Matrix-vector multiplication $\approx \mathcal{O}(N^2)$ operations

Sparse Matrices

- Sparse Matrices are Matrices with a majority of entries with value 0.
- Dense Matrices are Matrices with little to no 0 entries
- We have done Dense Matrix-vector multiplication $\approx \mathcal{O}(N^2)$ operations
- If we can leverage sparsity, we save computation.

Types of Sparse Matrix Representations

- Dictionary of Keys (DOK)
  - Dictionary that maps row, column pairs to the value of the entry

Types of Sparse Matrix Representations

- Dictionary of Keys (DOK)
    - Dictionary that maps row, column pairs to the value of the entry
- List of Lists (LIL)
    - LIL Stores one list per row, containing a column index and matrix entry

Types of Sparse Matrix Representations

- Dictionary of Keys (DOK)
    - Dictionary that maps row, column pairs to the value of the entry
- List of Lists (LIL)
    - LIL Stores one list per row, containing a column index and matrix entry
- Coordinate List (COO)
    - Stores a list (row,column, value) tuples of nonzero entries

Types of Sparse Matrix Representations

- Dictionary of Keys (DOK)
  - Dictionary that maps row, column pairs to the value of the entry
- List of Lists (LIL)
  - LIL Stores one list per row, containing a column index and matrix entry
- Coordinate List (COO)
  - Stores a list (row,column, value) tuples of nonzero entries
- Compressed Sparse Row (CSR, "Yale Format)
  - Represents the matrix by 3 one dimensional arrays
  - Value array
  - Column index
  - Row pointer

(CSR Format)

- Value Array contains non-zero values

(CSR Format)

- Value Array contains non-zero values
- Column index contains the column index of the non-zero values

(CSR Format)

- Value Array contains non-zero values
- Column index contains the column index of the non-zero values
- The row pointer array contains the compressed index of the value that starts a new row
- Row Pointer contains $M+1$ entries, defined as $R[0]=0$, $R[i]=R[I-1]+$ $\#$ of nonzero elements in row $i-1$

Example

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

Example

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

Then the CSR format is

$$\mathbf{v} = [5, 8, 3, 6]$$

$$\mathbf{c} = [0, 1, 2, 1]$$

$$\mathbf{r} = [0, 0, 2, 3, 4]$$

SpMV

- Best Use of CSR format is Sparse Matrix Dense Vector multiplication (SpMV)

SpMV

- Best Use of CSR format is Sparse Matrix Dense Vector multiplication (SpMV)

1. Create a segmented representation of matrix from value and row pointer vectors
2. Gather vector values using column indices
3. Pairwise multiply 1 and 2
4. Inclusive segmented sum scan on 3

SpMV Example

Suppose we wish to do

$$\begin{pmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

SpMV Example

Suppose we wish to do

$$\begin{pmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Value vector

$$\mathbf{v} = [a, b, c, d, e, f]$$

SpMV Example

Suppose we wish to do

$$\begin{pmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Value vector

$$\mathbf{v} = [a, b, c, d, e, f]$$

- Column

$$\mathbf{c} = [0, 2, 0, 1, 2, 2]$$

SpMV Example

Suppose we wish to do

$$\begin{pmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Value vector

$$\mathbf{v} = [a, b, c, d, e, f]$$

- Column

$$\mathbf{c} = [0, 2, 0, 1, 2, 2]$$

- Rowptr

$$\mathbf{rp} = [0, 0, 2, 5]$$

SpMV Example Continued

1. Segmented representation

$$[a, b | c, d, e | f]$$

SpMV Example Continued

1. Segmented representation

$$[a, b|c, d, e|f]$$

2. vector values using column

$$[x, z, x, y, z, z]$$

SpMV Example Continued

1. Segmented representation

$$[a, b|c, d, e|f]$$

2. vector values using column

$$[x, z, x, y, z, z]$$

3. Pairwise multiplication

$$[ax, bz|cx, dy, ez|fz]$$

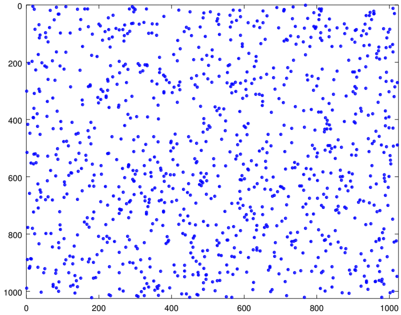
4. Segmented scan

$$[ax + bz|cx + dy + ez|fz]$$

$$\begin{pmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + \cancel{0y} + bz \\ cx + dy + ez \\ \cancel{0x + 0y} + fz \end{pmatrix}$$

In this simple case we save 3 multiplications and 3 adds. On large scale matrices the savings will be more substantial.

CUDA Example



Multiplied by a vector of 1s of size 1024

Kernel

Timing vs Dense MatVec