AMS 148 Lecture 5

Steven Reeves

University of California, Santa Cruz

sireeves@ucsc.edu

April 23, 2020

Overview

1. Reduce
   - Parallel Add Reduce
   - Brent's Theorem
   - CUDA Reduce
   - Finite Integrals using Reduce
2. Scan
   - Inclusive Scan
   - Exclusive Scan
   - Application to CDF calculation

How do we add one billion floats $\in [1, 2)$

How do we add one billion floats $\in [1, 2)$

```
summ = 0.0f;
for(int i = 0; i < n; i++)
        summ += array[i];
```

Problems with this implementation

- Slow

Problems with this implementation

- Slow
- Precision issues

$$10,000,000 + 1.234789 = 10,000,001.0$$

Problems with this implementation

- Slow
- Precision issues

$$10,000,000 + 1.234789 = 10,000,001.0$$

- How can we solve these issues?

Reduce

Let's first consider the underlying operation:

- Reduce

Reduce

Let's first consider the underlying operation:

- Reduce
  - Reduces an array to one data point

Reduce

Let's first consider the underlying operation:

- Reduce
  - Reduces an array to one data point
  - Requires a binary associative operator.

Reduce as a mathematical function

Let $\mathbf{x}$ be an array containing some data type, and let $\oplus$ be a binary associative operator.

Reduce as a mathematical function

Let $\mathbf{x}$ be an array containing some data type, and let $\oplus$ be a binary associative operator.

- Binary: If a, b are of the same type then $a \oplus b$ is of that type.

Reduce as a mathematical function

Let $\mathbf{x}$ be an array containing some data type, and let $\oplus$ be a binary associative operator.

- Binary: If a, b are of the same type then $a \oplus b$ is of that type.
- Associative: Let a, b, c be the same type, then

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

Reduce as a mathematical function

Let $\mathbf{x}$ be an array containing some data type, and let $\oplus$ be a binary associative operator.

- Binary: If a, b are of the same type then $a \oplus b$ is of that type.
- Associative: Let a, b, c be the same type, then

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

- The reduce is cast as

$$\mathcal{R}(\mathbf{x}, \oplus) = x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1}$$

Reduction Example

Suppose we want to add 8 floats.

- Float is data type

Reduction Example

Suppose we want to add 8 floats.

- Float is data type
- Binary operator is addition
- We know addition is associative

Reduction Example

Suppose we want to add 8 floats.

- Float is data type
- Binary operator is addition
- We know addition is associative
- The result is a summation.

```
for(int i = 0; i < 8; i++)
        summ += array[i];
```

Reduction Example

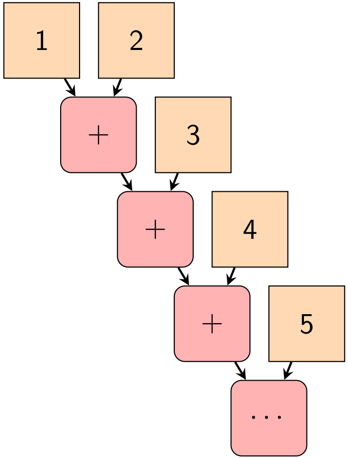
Suppose we want to add 8 floats.

- Float is data type
- Binary operator is addition
- We know addition is associative
- The result is a summation.

```
for(int i = 0; i < 8; i++)
        summ += array[i];
```

How can we use the above assumptions to make this summation parallel?

Computational Tree



Parallel Sum Computational Tree

On the Board

Complexity of Parallel Reduce

N	Steps
2	1
4	2
8	3

Table: Step complexity for parallel reduce

Complexity of Parallel Reduce

N	Steps
2	1
4	2
8	3

Table: Step complexity for parallel reduce

What type of pattern do we see?

Complexity of Parallel Reduce

N	Steps
2	1
4	2
8	3

Table: Step complexity for parallel reduce

What type of pattern do we see?

$$\text{Steps} = \mathcal{O}(\log_2(N))$$

True Scaling?

Note it only scales like $\log_2(N)$ if we have N processors. Suppose we have only $p < N$ processors?

True Scaling?

Note it only scales like $\log_2(N)$ if we have N processors. Suppose we have only $p < N$ processors? Then we use Brent's Theorem, to find the true scaling.

Representing Algorithms as Graphs

- We can represent algorithms as computational trees

Representing Algorithms as Graphs

- We can represent algorithms as computational trees
- These trees are often *directed acyclic graphs*(DAGs)

Representing Algorithms as Graphs

- We can represent algorithms as computational trees
- These trees are often *directed acyclic graphs*(DAGs)
- DAGs are useful to illustrate an algorithms flow and dependencies

Example DAGs

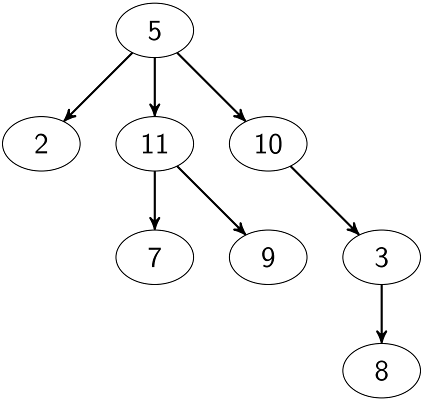


Figure: Example directed acyclic graph

Brent's Theorem

- T_1 = serial execution time (number of nodes at row 1 in this case)

Brent's Theorem

- T_1 = serial execution time (number of nodes at row 1 in this case)
- T_∞ = depth of the DAG

Brent's Theorem

- T_1 = serial execution time (number of nodes at row 1 in this case)
- T_∞ = depth of the DAG
- T_p = steps an algorithm takes with p threads

Brent's Theorem

- T_1 = serial execution time (number of nodes at row 1 in this case)
- T_∞ = depth of the DAG
- T_p = steps an algorithm takes with p threads

Then Brent's Theorem states:

$$\frac{T_1}{p} \leq T_P \leq \frac{T_1}{p} + T_\infty$$

So for a reduce algorithm

$$T_p \leq \frac{T_1}{p} + T_\infty = \frac{N}{p} + \log_2(N)$$

Parallel Summation

Lets sum a million points (acutally 2^{20}).

Parallel Summation

Lets sum a million points (acutally 2^{20}).

```
float Bad_serial_reduce(const float *data, int N)
{
        float summ =0.0f;
        for(int i = 0; i < N; i++)
                summ+= data[i];
        return summ;
}
```

- this is bad
- we'll do two different ways in CUDA

Application Of Reduce

It is known that

$$\int_{-1}^{1} \sqrt{1 - x^2} dx = \frac{\pi}{2}$$

Application Of Reduce

It is known that

$$\int_{-1}^{1} \sqrt{1 - x^2} dx = \frac{\pi}{2}$$

- We can calculate more digits of pi this way

Application Of Reduce

It is known that

$$\int_{-1}^{1} \sqrt{1 - x^2} dx = \frac{\pi}{2}$$

- We can calculate more digits of pi this way
- We can test our reduction algorithm with this

Application Of Reduce

It is known that

$$\int_{-1}^{1} \sqrt{1 - x^2} dx = \frac{\pi}{2}$$

- We can calculate more digits of pi this way
- We can test our reduction algorithm with this
- How do we go from an integral to a sum?

Numerical Integration

Composite Trapezoidal Rule:

$$\int_a^b f(x)dx \approx \sum_1^N \left(f(x_{j-1}) + f(x_j)\right) \frac{\delta x}{2}$$

Numerical Integration

Composite Trapezoidal Rule:

$$\int_a^b f(x)dx \approx \sum_1^N \left(f(x_{j-1}) + f(x_j)\right)\frac{\delta x}{2}$$

where

$$[a,b] = \bigcup_{i=1}^N [x_{i-1}, x_i]$$

and $a = x_0$, and $b = x_N$.

Kernels

In our application we will set $N = 2^{20}$.

- This application is a map-reduce algorithm

Kernels

In our application we will set $N = 2^{20}$.

- This application is a map-reduce algorithm
- We must Map onto $f(x) = \sqrt{1 - x^2}$

Kernels

In our application we will set $N = 2^{20}$.

- This application is a map-reduce algorithm
- We must Map onto $f(x) = \sqrt{1 - x^2}$
- Then we have two stages of reduce.
- Lastly we will multiply the result by 2.

Scan

- Scan is a generalization of reduce to yield an array

Scan

- Scan is a generalization of reduce to yield an array
- Any binary operation can be used in a scan algorithm

Scan

- Scan is a generalization of reduce to yield an array
- Any binary operation can be used in a scan algorithm
- Notable applications: CDF calculation, sorting algorithms

A Short Example

$$\text{Input: } \{1, 2, 3, 4\}$$

$$\text{Operation: } +$$

$$\text{Output: } \{1, 3, 6, 10\}$$

Mathematical Representation of Scan

$$\mathcal{S}(\mathbf{x}, \oplus) = \mathbf{y}$$

Mathematical Representation of Scan

$$\mathcal{S}(\mathbf{x}, \oplus) = \mathbf{y}$$

- The operator $\oplus$ forms a group over the set of elements in $\mathbf{x}$

Mathematical Representation of Scan

$$\mathcal{S}(\mathbf{x}, \oplus) = \mathbf{y}$$

- The operator $\oplus$ forms a group over the set of elements in $\mathbf{x}$
- $\oplus$ is associative
- $\oplus$ is closed, i.e. $x \oplus y = z$ where x, y, z are of the same type
- There exists an identity element e, that is $e \oplus x = x$ for every element of type x

What does scan do?

Let $\mathcal{S}$ be the scan primitive, and $\oplus$ be a binary operator for the data type, then for an inclusive scan

$$\left[a_0, a_1, a_2, \cdots, a_{n-1}\right] \text{ :input}$$

$$\left[a_0, a_0 \oplus a_1, a_0 \oplus a_1 \oplus a_2, \cdots, \bigoplus_{j=0}^{n-1} a_j\right] \text{ :output}$$

What does scan do?

Let $\mathcal{S}$ be the scan primitive, and $\oplus$ be a binary operator for the data type, then for an inclusive scan

$$\left[a_0, a_1, a_2, \cdots, a_{n-1}\right] \text{ :input}$$

$$\left[a_0, a_0 \oplus a_1, a_0 \oplus a_1 \oplus a_2, \cdots, \bigoplus_{j=0}^{n-1} a_j\right] \text{ :output}$$

and for an exclusive scan

$$\left[a_0, a_1, a_2, \cdots, a_{n-1}\right] \text{ :input}$$

$$\left[e, a_0, a_0 \oplus a_1, a_0 \oplus a_1 \oplus a_2, \cdots, \bigoplus_{j=0}^{n-2} a_j\right] \text{ :output}$$

Implementation of scan

```
int acc = identity; //for op + identity = 0.0;
for (int = 0; i < elements.length(); i++)
        {
                acc = acc op element[i] // acc + element[i]
                    or max(acc, element);
                out[i] = acc;
        }
```

Hillis and Steele

- Danny Hillis And Guy Steele 1986

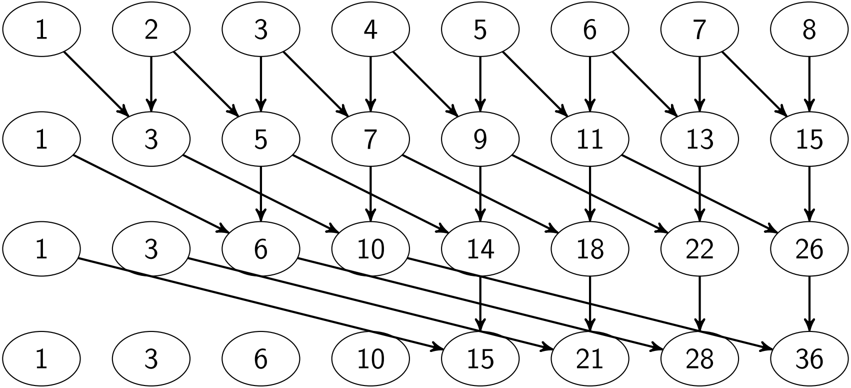
Hillis and Steele

- Danny Hillis And Guy Steele 1986
- Thinking Machines

Hillis and Steele

- Danny Hillis And Guy Steele 1986
- Thinking Machines
- It's best to observe the graph of this algorithm

Hillis And Steele Scan



Properties of the Hillis and Steele Algorithm

- Has Step complexity $\mathcal{O}(\log(n))$

Properties of the Hillis and Steele Algorithm

- Has Step complexity $\mathcal{O}(\log(n))$
- However has $\mathcal{O}(n \log(n))$ work complexity

Properties of the Hillis and Steele Algorithm

- Has Step complexity $\mathcal{O}(\log(n))$
- However has $\mathcal{O}(n\log(n))$ work complexity
- Essentially doing n reductions

Properties of the Hillis and Steele Algorithm

- Has Step complexity $\mathcal{O}(\log(n))$
- However has $\mathcal{O}(n\log(n))$ work complexity
- Essentially doing n reductions
- Is an *inclusive* scan

Properties of the Hillis and Steele Algorithm

- Has Step complexity $\mathcal{O}(\log(n))$
- However has $\mathcal{O}(n\log(n))$ work complexity
- Essentially doing n reductions
- Is an *inclusive* scan
- Best for small arrays where the number of processors is equal to or greater than the number of array elements.

A more work efficient scan?

- Serial scan has a work complexity of n

A more work efficient scan?

- Serial scan has a work complexity of n
- The Hillis and Steele algorithm is *more* work complex than serial

A more work efficient scan?

- Serial scan has a work complexity of n
- The Hillis and Steele algorithm is *more* work complex than serial
- If the number of size of the data array is larger than the number of threads, we seek a more work efficient algorithm than H&S

A more work efficient scan?

- Serial scan has a work complexity of n
- The Hillis and Steele algorithm is *more* work complex than serial
- If the number of size of the data array is larger than the number of threads, we seek a more work efficient algorithm than H&S
- To this effect we look to the Blelloch Scan

Blelloch

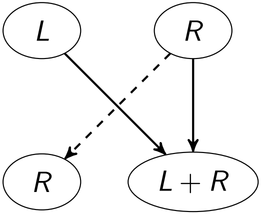
- Formulated by Guy Blelloch in 1990

Blelloch

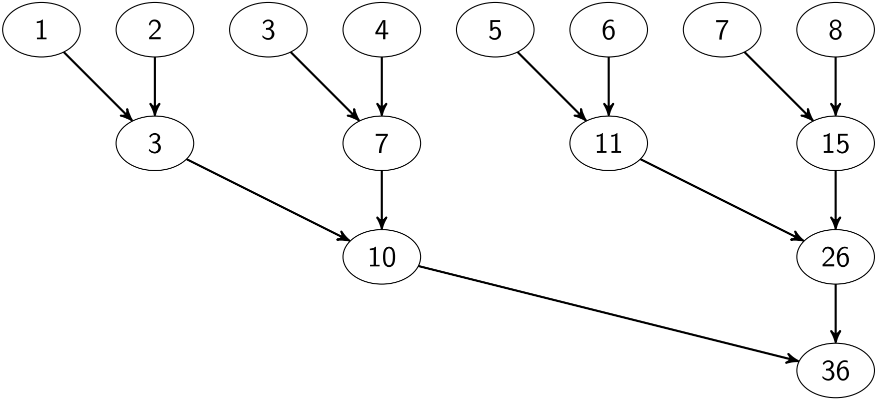
- Formulated by Guy Blelloch in 1990
- Has two stages
  - Reduce
  - Downsweep

Blelloch

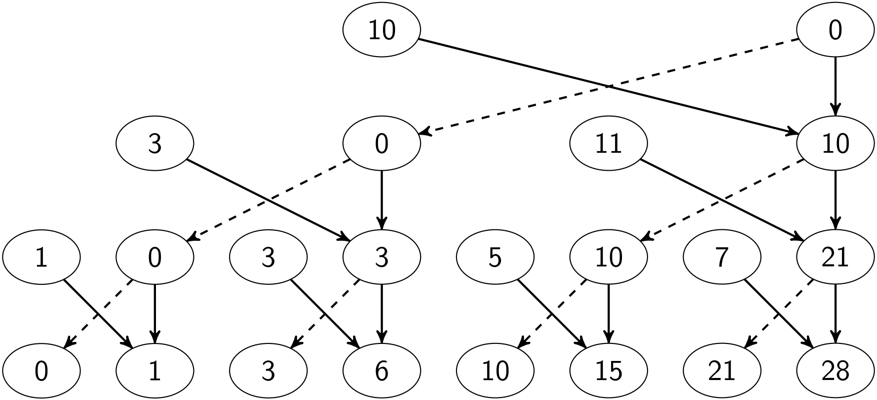
- Formulated by Guy Blelloch in 1990
- Has two stages
  - Reduce
  - Downsweep
- Requires the downseep "operator"



Reduce Phase



Downseep Phase



Properties of the Blelloch Scan

- The Step complexity of the reduce phase is $\mathcal{O}(\log(n))$

Properties of the Blelloch Scan

- The Step complexity of the reduce phase is $\mathcal{O}(\log(n))$
- The Work complexity is of $\mathcal{O}(n)$.

Properties of the Blelloch Scan

- The Step complexity of the reduce phase is $\mathcal{O}(\log(n))$
- The Work complexity is of $\mathcal{O}(n)$.
- The communication pattern of the downsweep mirrors reduce
- Thus the step and work complexity are the same

Properties of the Blelloch Scan

- The Step complexity of the reduce phase is $\mathcal{O}(\log(n))$
- The Work complexity is of $\mathcal{O}(n)$.
- The communication pattern of the downsweep mirrors reduce
- Thus the step and work complexity are the same
- So the Blelloch scan has $2\log(n)$ steps, but $\mathcal{O}(n)$ work

Properties of the Blelloch Scan

- The Step complexity of the reduce phase is $\mathcal{O}(\log(n))$
- The Work complexity is of $\mathcal{O}(n)$.
- The communication pattern of the downsweep mirrors reduce
- Thus the step and work complexity are the same
- So the Blelloch scan has $2\log(n)$ steps, but $\mathcal{O}(n)$ work
- Note that the Blelloch Scan is *exclusive*

Mix and match

- What if we want a work efficient inclusive scan?
- Or a step efficient exclusive scan?

Mix and match

- What if we want a work efficient inclusive scan?
- Or a step efficient exclusive scan?
- For inclusive to exclusive:
  - Shift all elements to the right, drop last element
  - Store the identity in the first entry

Mix and match

- What if we want a work efficient inclusive scan?
- Or a step efficient exclusive scan?
- For inclusive to exclusive:
  - Shift all elements to the right, drop last element
  - Store the identity in the first entry
- Exclusive to Inclusive:
  - Shift all elements to the left, drop first element
  - Perform operation on last element of scan and last of original array
  - Or – store the reduced element in a temporary variable at the end of the ruduce phase

CDF

- To illustrate the use of scan we will compute a Cummulative Distribution Function

CDF

- To illustrate the use of scan we will compute a Cummulative Distribution Function
- Our underlying Probability Density Function will be the normal distribution.

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]$$

The CDF for the Normal distribution is

$$\Phi(x|\mu,\sigma^2) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]$$

The CDF for the Normal distribution is

$$\Phi(x|\mu,\sigma^2) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]$$

- Cannot describe the error function as a combination of elementary functions

The CDF for the Normal distribution is

$$\Phi(x|\mu,\sigma^2) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]$$

- Cannot describe the error function as a combination of elementary functions
- Must find it numerically.

Definition of CDF

Note that

$$\Phi(x|\mu,\sigma^2) = \int_{-\infty}^{x} f(x'|\mu,\sigma^2)dx'$$

Definition of CDF

Note that

$$\Phi(x|\mu,\sigma^2) = \int_{-\infty}^{x} f(x'|\mu,\sigma^2)dx'$$

- Computers can't do infinity

Definition of CDF

Note that

$$\Phi(x|\mu,\sigma^2) = \int_{-\infty}^{x} f(x'|\mu,\sigma^2)dx'$$

- Computers can't do infinity
- However, f is rapidly descreasing
- That is, $f \to 0$ as $|x| \to \infty$ "faster" than any polynomial

Definition of CDF

Note that

$$\Phi(x|\mu,\sigma^2) = \int_{-\infty}^{x} f(x'|\mu,\sigma^2)dx'$$

- Computers can't do infinity
- However, f is rapidly descreasing
- That is, $f \to 0$ as $|x| \to \infty$ "faster" than any polynomial
- And by the empirical rule, 99.7% of the probability is contained within 3 standard deviations from the mean

$$\Phi(x|\mu,\sigma^2) \approx \int_{x-5\mu}^{x} f(x'|\mu,\sigma^2)dx'$$

Numerical Plan

1. Use Trapezoidal Rule to discretize the integral

Numerical Plan

1. Use Trapezoidal Rule to discretize the integral
2. Use a Stencil + Map to generate array to be scanned
3. Use device function to apply normal distribution to x

Numerical Plan

1. Use Trapezoidal Rule to discretize the integral
2. Use a Stencil + Map to generate array to be scanned
3. Use device function to apply normal distribution to x
4. Use work efficient Blelloch Scan to perform the numerical integration
5. Perform the shift to turn exclusive scan to inclusive

Stencil + Map Kernel

Blelloch Scan Kernel

Shift Kernel

Generalizing Scan to Larger Arrays

1. Perform scan on all m thread blocks

Generalizing Scan to Larger Arrays

1. Perform scan on all m thread blocks
2. Create a second array to contain the last element from each scan

Generalizing Scan to Larger Arrays

1. Perform scan on all m thread blocks
2. Create a second array to contain the last element from each scan
3. Scan this array

Generalizing Scan to Larger Arrays

1. Perform scan on all m thread blocks
2. Create a second array to contain the last element from each scan
3. Scan this array
4. Use newly updated second array to correct original scan by thread block