Week 6

Prefix Sum and Parallel Sort

1

Prefix Sum

2

A *prefix sum* is an algorithm that is applied to the elements of an n-element array of numbers, and its output is an n-element array that's obtained by adding up the elements "preceding" the current element.

3

Example

- 5 1 6 2 4
- 5 6 12 14 18

4

Sequential Code

```
int serial_prefixsum(double* a, int n) {
for (int i = 1; i < n; i++) {
a[i] = a[i] + a[i - 1];
}
    return 1;
}
```

- Is there any dependence?
- How can you parallelize this?
- Have we seen anything similar?

5

Exclusive prefix sum, element 0 is 0, and the other elements are obtained by adding the preceding element of the array to the most recently computed element of the prefix sum array

6

Example

- 5 1 6 2 4
- 5 6 12 14 18
- 0 5 6 12 14

7

Parallel Prefix: A pictorial view – down step

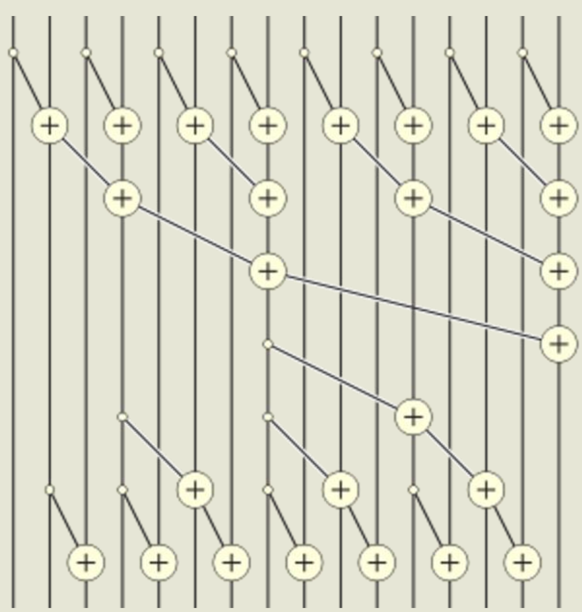
3	1	7	0	4	1	6	3
	+		+		+		+
3	4	7	7	4	5	6	9
			+				+
3	4	7	11	4	5	6	14
							+
3	4	7	11	4	5	6	25

8

Parallel Prefix: A pictorial view – up step

3	4	7	11	4	5	6	25
					+		
3	4	7	11	4	16	6	25
		+		+		+	
3	4	11	11	15	16	22	25

9



Parallel Prefix Sum – Brent Kung Algorithm

10

It is left as self-study assignment to read Kogge-Stone algorithm and compare with Brent Kung Algorithm.

11

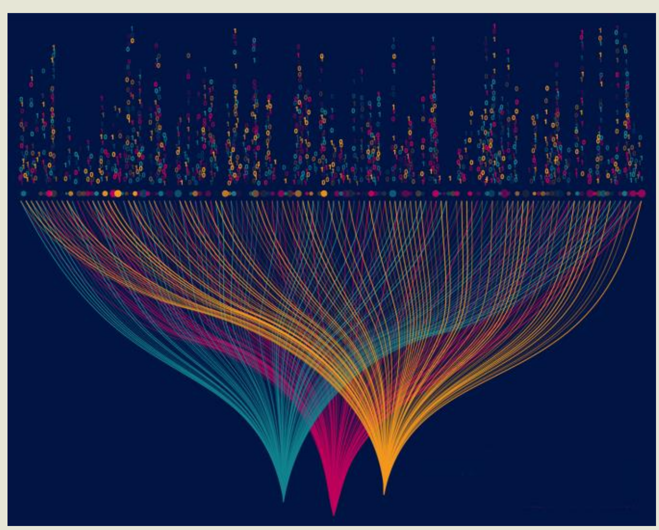
A more efficient Prefix Sum

12

Any problem with Brent Kung Algorithm?

13

Data >>> CPUs (cores) available



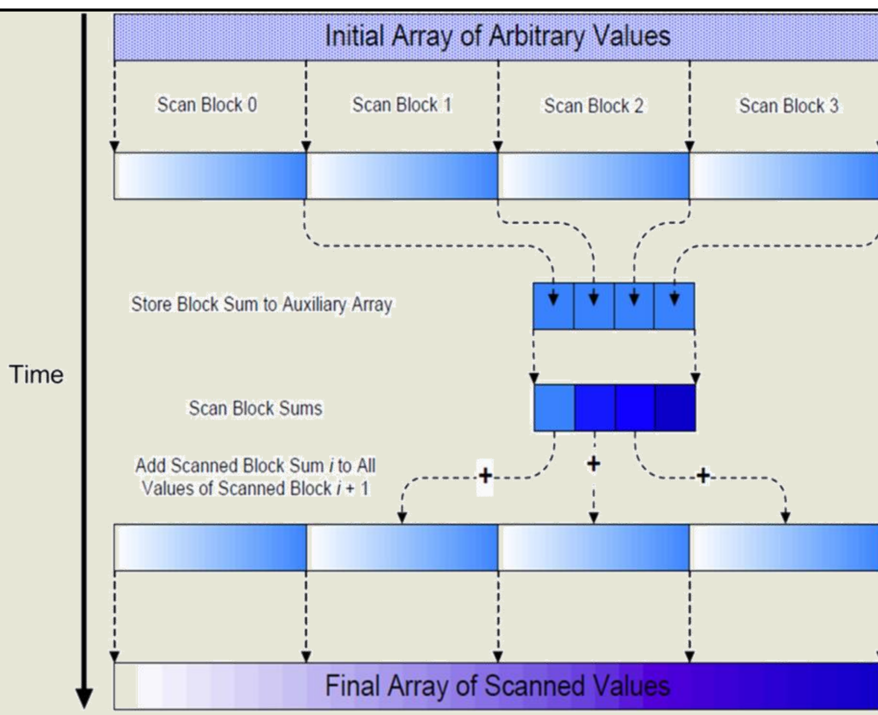
14

Thre Phase Prefix Adder

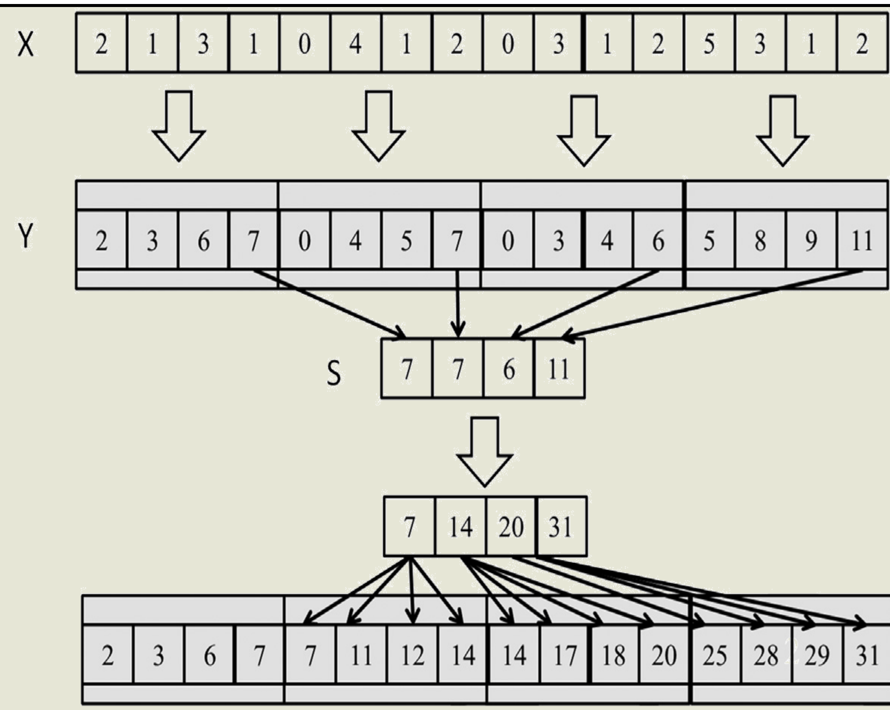
2	1	3	1	0	4	1	2	0	3	1	2	5	3	1	2

Phase 1: Scan sub arrays on individual CPUs

15



16



17

Prefix Sum in MPI

- MPI_Scan is an inclusive scan:
  - Performs a prefix reduction across all MPI processes in the given **communicator**.

18

Prefix Sum in MPI

- In other words, each MPI process receives the result of the reduction operation on the values passed by that MPI process and all MPI processes with a lower rank.

19

Prefix Sum in MPI

- MPI_Scan is a collective operation;
  - it must be called by all MPI processes in the communicator concerned.
- The variant of MPI_Scan is the exclusive version *MPI_Exscan*.

20

```
int MPI_Scan(
  _In_  void          *sendbuf,
  _Out_ void          *recvbuf,
        int           count,
        MPI_Datatype  datatype,
        MPI_Op        op,
        MPI_Comm      comm
);
```

21

Parameters

- ***sendbuf*** [in]: Starting address of send buffer.
- ***recvbuf*** [out]: Starting address of receive buffer.
- ***count***: Number of elements in input buffer.
- ***datatype***: Datatype of elements of input buffer.
- ***op***: Operation.
- ***comm***: Communicator.

22

Return Value

- Returns MPI_SUCCESS on success. Otherwise, the return value is an error code.

23

Exclusive Scan

24

Adders and Algorithms

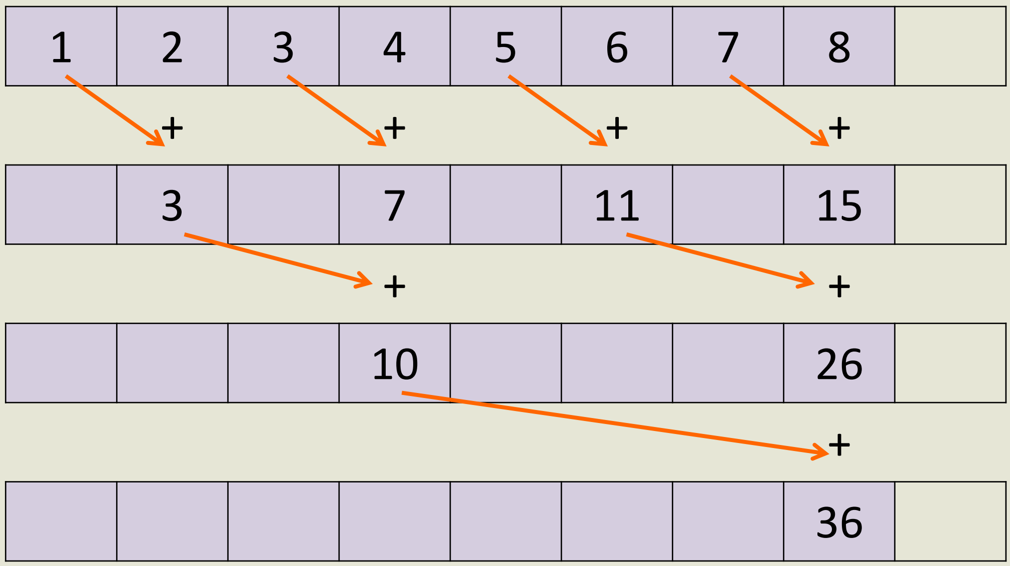
- Kogge-Stone Adder →
- Hillis Steele algorithm
- Brent-Kung Adder →
- Blelloch

Recall exclusive scan

- ***Exclusive prefix sum***, element 0 is 0, and the other elements are obtained by adding the preceding element of the array to the most recently computed element of the prefix sum array

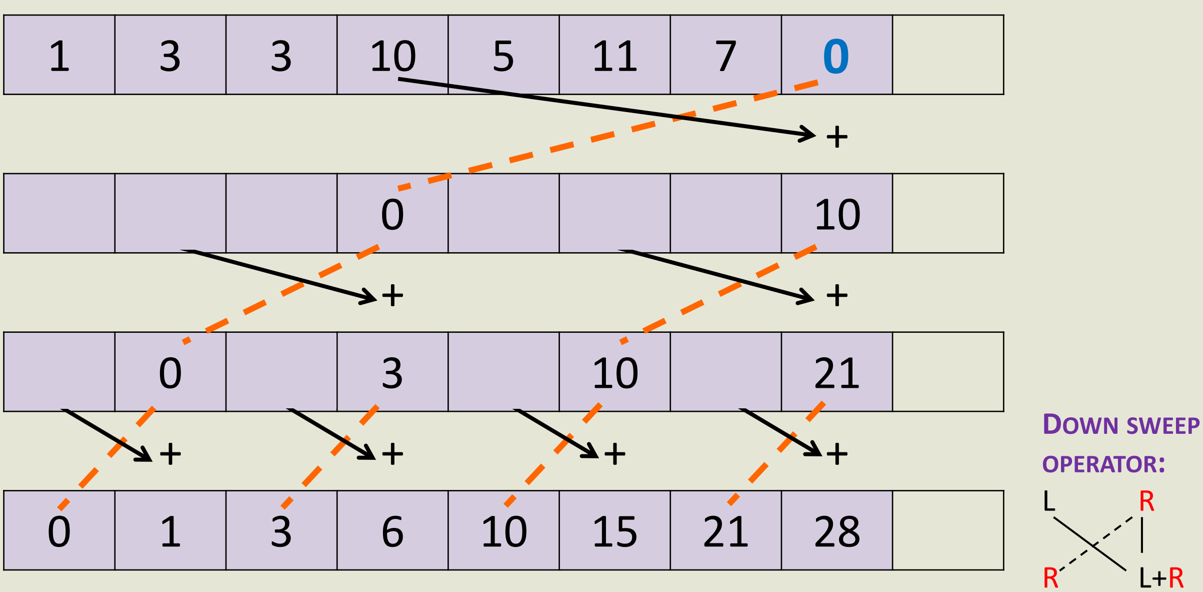
- 5 1 6 2 4
- 0 5 6 12 14

Blelloch Parallel Exclusive Scan: A pictorial view – reduction step



27

Blelloch Parallel Exclusive Scan : A pictorial view – down sweep



28

Complexity Analysis

Hillis-Steel vs Blelloch

29

Some definitions

- **Depth D(n)** = "# of iterations/steps" = parallel running time $T_p(n)$
- **Work W(n)** = total number of operations performed by all threads together
  - With sequential algorithms, work complexity = time complexity
- **Work-efficient:**
  - A parallel algorithm is called work-efficient, if it performs no more work than the sequential one

30

Hillis-Steele Parallel **Inclusive** Scan: A pictorial view

1	2	3	4	5	6	7	8	
|	+	+	+	+	+	+	+	
1	3	5	7	9	11	13	15	
|	|	+	+	+	+	+	+	
1	3	6	10	14	18	22	26	
				+	+	+	+	
1	3	6	10	15	21	28	36	

31

Hillis-Steele Parallel **Inclusive** Scan: Complexity

1	2	3	4	5	6	7	8	
|	+	+	+	+	+	+	+	
1	3	5	7	9	11	13	15	$N-2^0$
|	|	+	+	+	+	+	+	
1	3	6	10	14	18	22	26	$N-2^1$
				+	+	+	+	
1	3	6	10	15	21	28	36	$N-2^2$

NUMBER OF STEPS:
log(n)

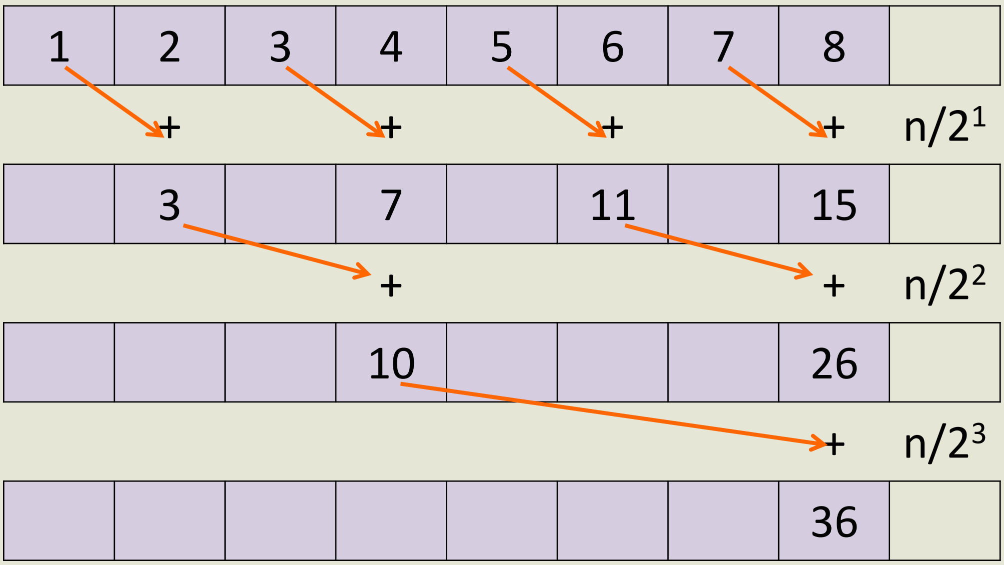
AMOUNT OF WORK
n-2^0 +
n-2^1 +
n-2^2 +
…
½ N

Around N work in each step.

nlog(n)

32

Blelloch Parallel Scan: Complexity



Number of Steps:

2xlog(n) = O(logn)

Amount of Work

2xn = O(n)

Note: once for reduction step and then for down-sweep

33