CME 213, ME 339 Spring 2023
Introduction to parallel computing using MPI, openMP, and CUDA

Eric Darve, ICME, Stanford

Group Activity, May 22, 2023

Stanford University

Group activity

- These are instructions for the group activity.
- The goal of the activity is to calculate a prefix sum in parallel using a human computer.
- Let’s review first how to calculate a reduction in parallel.

Reduction

Reduction is a classic problem in parallel computing.

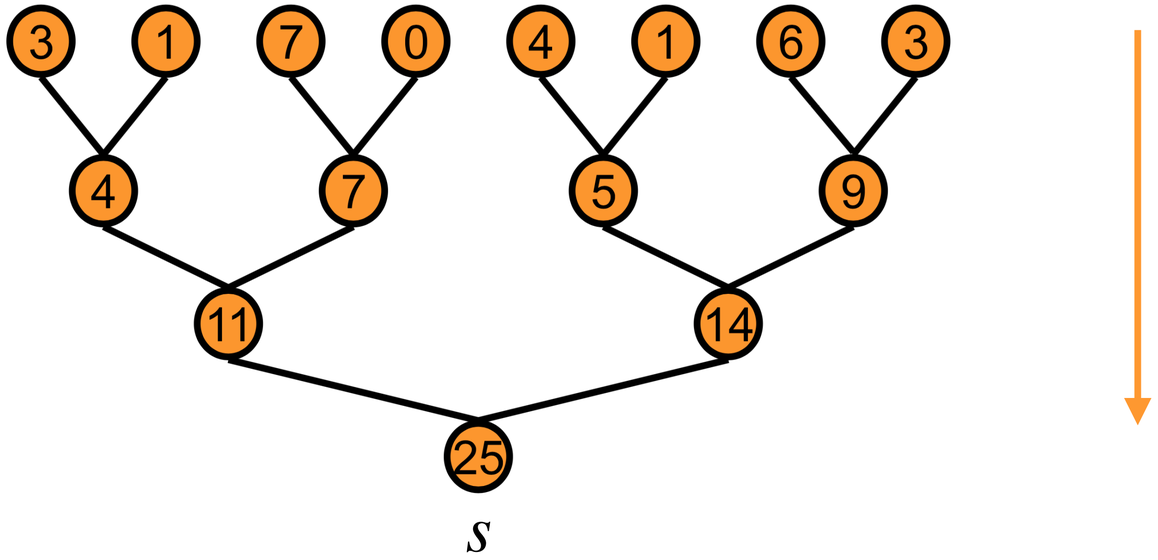
A reduction is defined as the following computation:

$$s = \sum_{i=1}^{n} x_i$$

The goal is to compute this efficiently in parallel.

Binary tree

- The best strategy uses a binary tree.
- We start adding entries at the leaf and progressively go down the tree adding partial sums until we get the final result s.



Running time

- This approach is optimal in the sense that it minimizes the overall running time.
- For n entries, the total time is $O(\ln_2 n)$.

Prefix sum or scan

- The goal of the activity is to compute a prefix sum in parallel.
- Here is an example.
- `Input:    3    5    6    2    4`
- `Output:   3    8   14   16   20`

Mathematical definition of scan: $y_i = \sum_{j=1}^{i} x_j$

Algorithm 1

Algorithm 1 has 2 phases:

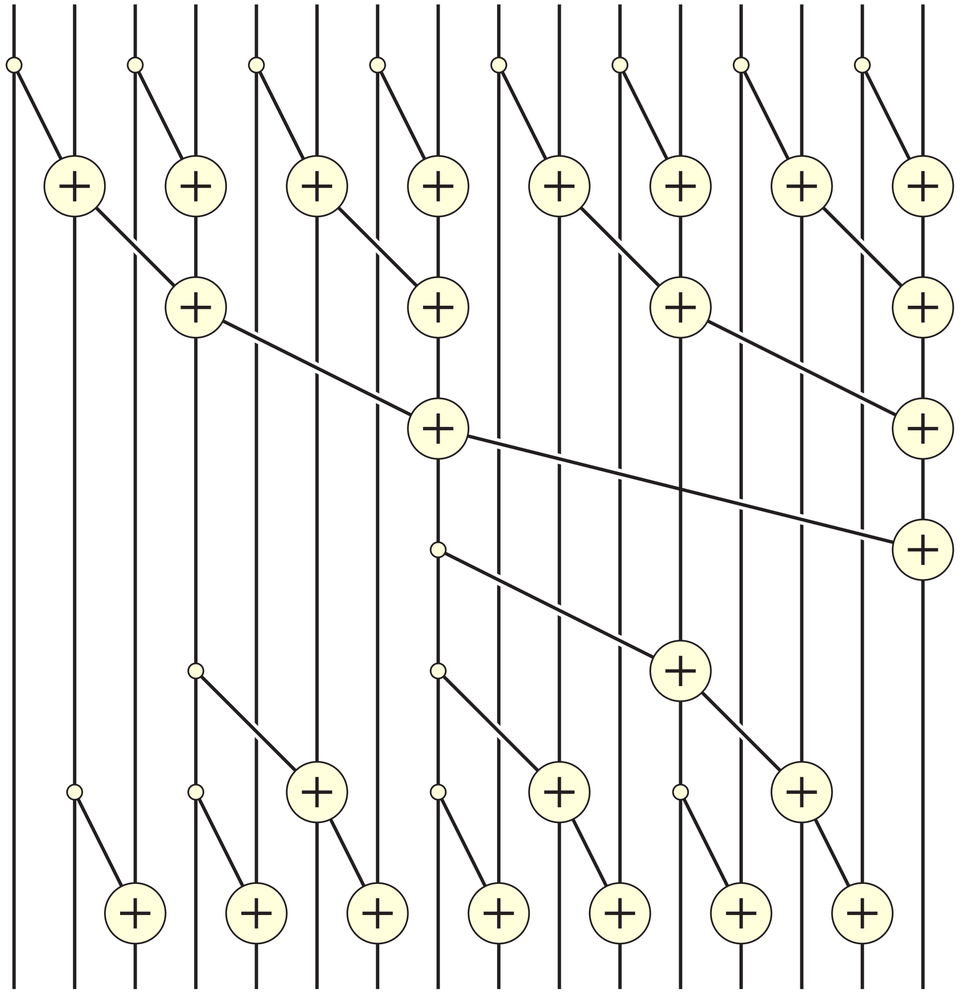
1. Collect or reduce partial results.

2. Distribute the partial sums in order to get the final result.

Algorithm 1

Perform partial reductions

Distribute the results to final locations



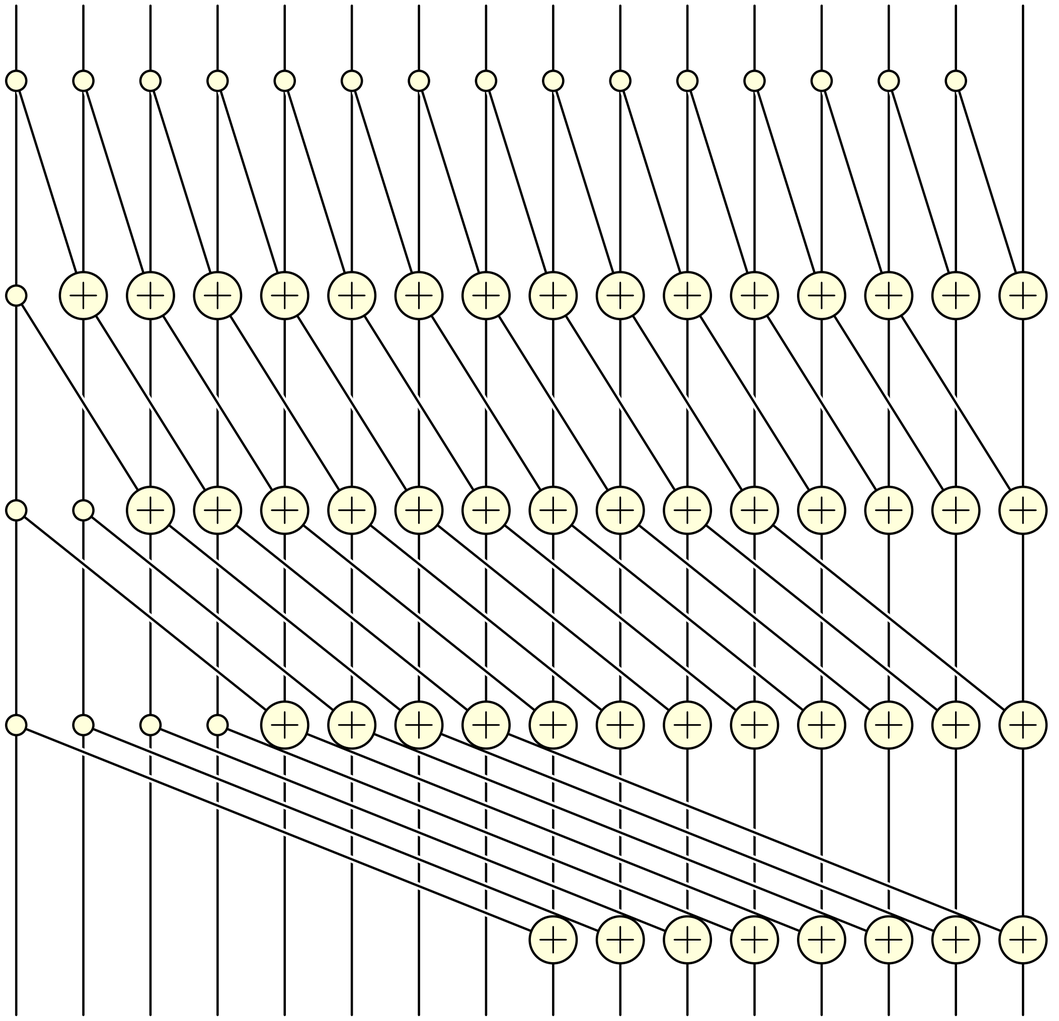
Complexity

- Number of passes: $2 \ln_2 n - 2$
- Flops is approximately 2x the amount of flops for a sequential algorithm.
- This algorithm is **work efficient.** It performs a small amount of extra work compared to the sequential algorithm.
- This extra work is required to make the algorithm parallel.
- This is a common tradeoff.

Algorithm 2

The Hillis and Steele algorithm performs more work by computing many reductions in parallel, but it can run faster in some cases.

Algorithm 2



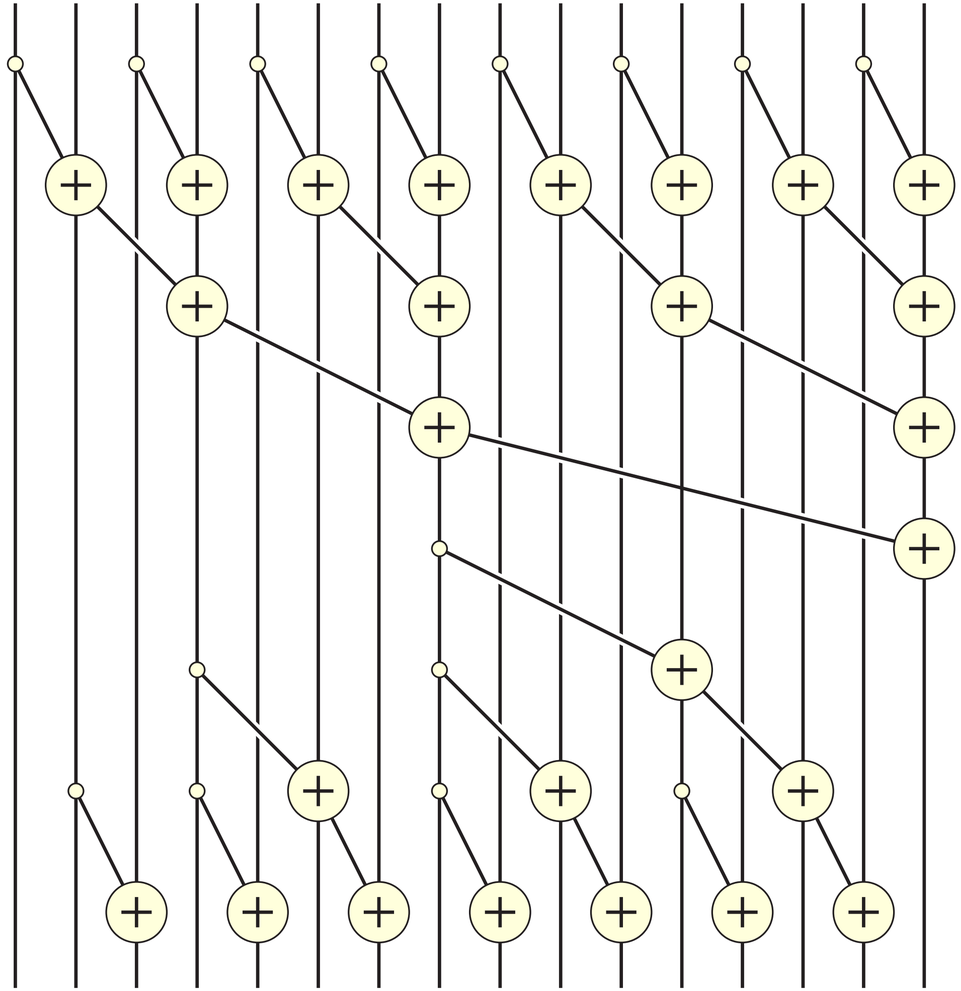
Many reductions are computed in parallel

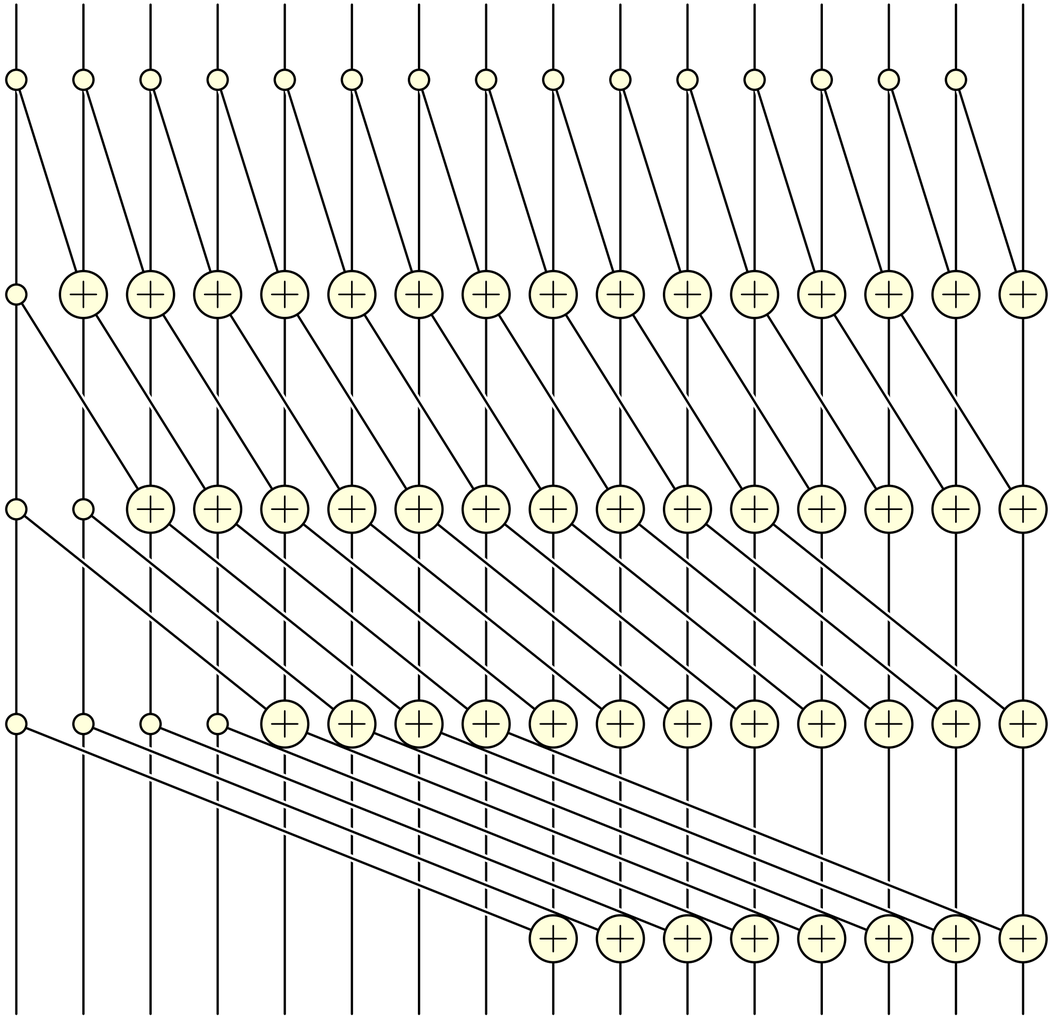
Complexity

- Number of passes: $\ln_2 n$ vs. $2 \ln_2 n - 2$ for Algorithm 1.
- **Algorithm 2 is faster when many cores are available.**
- Flops is approximately $\ln_2 n$ the amount of flops for a sequential algorithm.
- **More work** is required compared to Algorithm 1.

Which algorithm is best?

- If you have fewer cores, Algorithm 1 is the best.
- If you have many cores, Algorithm 2 completes faster.
- In practice, hybrid algorithms can be designed that represent tradeoffs between reducing the number of stages in the algorithm (shorter span) and the number of flops (work efficiency).





Game time

Teams

- You should form teams of about 10 players.
- **Bring a lot of small 3in x 3in pieces of paper to use as scratch paper.**
- Goal of activity: parallel prefix sum using “human” processors

Human computers

- We are going to build a computer using humans.
- There are 3 types of players in this game: **mem, net, pu.**
- Each player has only one type.
- You can have as many players of each type as you want.

Rules!

- A **memloc** is a piece of paper with a single number written on it. You can strike a number to write a new one. A memloc is full when it has a number on it, and empty otherwise.

- There are 3 types of players: **mem, net, pu.** Each player has only one type. You can have as many players of each type as you want.

- **mem player:** can copy and strikethrough numbers. mem players are allowed to use scratch paper to organize the calculation.

- **net player:** can take/give memlocs to mem and pu. Cannot hold more than 3 memlocs at a time.

- **pu player:** takes two full memlocs and one empty memloc, adds the numbers on the full memlocs (may use a calculator) and writes the result on the empty memloc. Can take/give memlocs to net. Cannot hold more than 3 memlocs at a time.

- pu players must be at least 20 meters away from the mem players.

Generating the prefix scan problem

We will use the following code to generate sequence:

```
generate_sequence.cpp
```

See the notebook:

https://colab.research.google.com/drive/1Zw8atH5szXmU6Uho--2fbWdzp7nwbGtJ?usp=sharing

- Run the notebook.
- Enter your team number. The code returns a sequence of random numbers.
- This is the sequence you will use for the prefix scan.

Example: team 1 output

```
1 !g++ generate_sequence.cpp
2 !group_number=1; echo $group_number | ./a.out
```

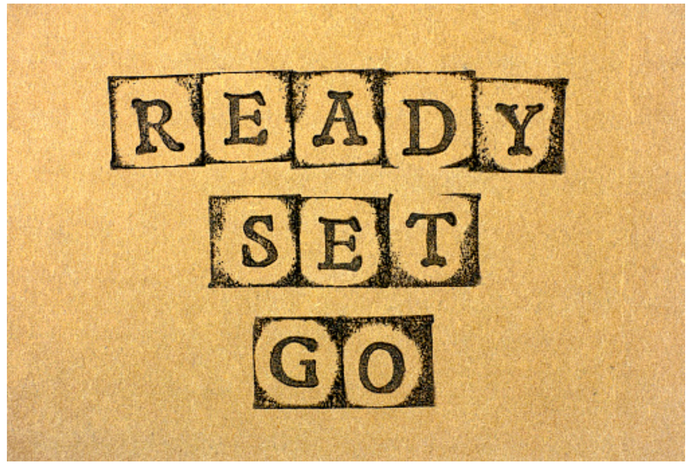
```
Enter your group number (an integer greater or equal to 1)
Selected group ID: 1
Row 1: index
Row 2: random value

Index  1 to 10
1            2            3            4            5            6            7            8            9            10
57           65           28           84           10           33           43           78           86           27

Index 11 to 20
11           12           13           14           15           16           17           18           19           20
61           63           63           14           69           20           66           65           92           67

Index 21 to 30
21           22           23           24           25           26           27           28           29           30
41           72           53           50           83           37           17           20           93           83

Index 31 to 40
31           32           33           34           35           36           37           38           39           40
41           83           67           13           62           35           89           70           57           18
```


READY
SET
GO

Follow-up discussion

- How did you organize your group?
- What was the best strategy?
- What were the main bottlenecks?
- What would you do differently?