## COMP 322: Parallel and Concurrent Programming

# Lecture 36: Algorithms Based on Parallel Prefix (Scan) Operations 

Mack Joyner<br>mjoyner@rice.edu<br>http://comp322.rice.edu

## Formalizing Parallel Prefix: Scan operations

- The $i$-scan operation is an inclusive parallel prefix sum operation.
- The scan operator was introduced in APL in the 1960's, and has been popularized recently in more modern languages, most notably the NESL project in CMU


## Formalizing Parallel Prefix: Scan operations

- The e-scan operation is an exclusive parallel prefix sum operation. It takes a binary associative operator $\oplus$ with identity $I$, and a vector of $n$ elements, $\left[a_{0}, a_{1}, \ldots, a_{n-1}\right]$, and returns the vector $\left[l, a_{0},\left(a_{0} \oplus a_{1}\right), \ldots,\left(a_{0} \oplus a_{1} \oplus \ldots \oplus a_{n-2}\right)\right]$.
- An e-scan can be generated from a i-scan by shifting the vector right by one and inserting the identity. Similarly, the i-scan can be generated from the e-scan by shifting left, and inserting at the end the sum of the last element of the e-scan and the last element of the original vector.


## Line-of-Sight Problem

- Problem Statement: given a terrain map in the form of a grid of altitudes and an observation point, X , on the grid, find which points are visible along a ray originating at the observation point. Note that a point on a ray is visible if and only if no other point between it and the observation point has a greater vertical angle.
- Define angle[i] = angle of point i on ray relative to observation point, $X$ (can be computed from altitudes of $X$ and $i$ )
- A max e-scan on angle[*] returns to each point the maximum previous angle.
- Each point can compare its angle with its max e-scan value to determine if it will be visible or not



## Segmented Inclusive Scan

Goal: Given a data vector and a flag vector as inputs, compute independent i-scans on segments of the data vector specified by the flag vector.

$$
x_{i}=\left\{\begin{array}{lll}
a_{0} & & i=0 \\
\begin{cases}a_{i} & f_{i}=1 \\
\left(x_{i-1} \oplus a_{i}\right) & f_{i}=0\end{cases} & 0<i<n
\end{array}\right.
$$

$\left.\begin{array}{|llcccccccc|}\hline a & = & {[5} & 1 & 3 & 4 & 3 & 9 & 2 & 6] \\ f & = & {[1} & 0 & 1 & 0 & 0 & 0 & 1 & 0\end{array}\right]$

## Binary Addition

This is the pen and paper addition of two 4-bit binary numbers $x$ and $y$. c represents the generated carries. s represents the produced sum bits.

A stage of the addition is the set of $\mathbf{x}$ and $\mathbf{y}$ bits being used to produce the appropriate sum and carry bits. For example the highlighted bits $x_{2}$, $y_{2}$ constitute stage 2 which generates carry $\mathrm{c}_{2}$ and sum $\mathrm{s}_{2}$.

Each stage $i$ adds bits $a_{i}, b_{i}, c_{i-1}$ and produces bits $s_{i}, c_{i}$ The following hold:

| $\mathrm{a}_{\mathrm{i}}$ | $\mathrm{b}_{\mathrm{i}}$ | $\mathrm{c}_{\mathrm{i}}$ | Comment: | Formal definition: |  |
| :--- | :--- | :--- | :--- | :--- | :--- |
| 0 | 0 | 0 | The stage "kills" an incoming carry. | "Kill" bit: | $k_{i}=\overline{x_{i}+y_{i}}$ |
| 0 | 1 | $\mathrm{c}_{\mathrm{i}-1}$ | The stage "propagates" an incoming carry | "Propagate" bit: | $p_{i}=x_{i} \oplus y_{i}$ |
| 1 | 0 | $\mathrm{c}_{\mathrm{i}-1}$ | The stage "propagates" an incoming carry |  |  |
| 1 | 1 | 1 | The stage "generates" a carry out | "Generate" bit: | $g_{i}=x_{i} \bullet y_{i}$ |

## Binary Addition

| $\mathrm{a}_{\mathrm{i}}$ | $\mathrm{b}_{\mathrm{i}}$ | $\mathrm{c}_{\mathrm{i}}$ | Comment: | Formal definition: |  |
| :--- | :--- | :--- | :--- | :--- | :--- |
| 0 | 0 | 0 | The stage "kills" an incoming carry. | "Kill" bit: | $k_{i}=\overline{x_{i}+y_{i}}$ |
| 0 | 1 | $\mathrm{c}_{\mathrm{i}-1}$ | The stage "propagates" an incoming carry | "Propagate" bit: | $p_{i}=x_{i} \oplus y_{i}$ |
| 1 | 0 | $\mathrm{c}_{\mathrm{i}-1}$ | The stage "propagates" an incoming carry |  |  |
| 1 | 1 | 1 | The stage "generates" a carry out | "Generate" bit: | $g_{i}=x_{i} \bullet y_{i}$ |

The carry $c_{i}$ generated by a stage $i$ is given by the equation:

$$
c_{i}=\xi_{i}+p_{i} \cdot \mathcal{C}_{i-1}=x_{i} \cdot y_{i}+\left(x_{i} \oplus y_{i}\right) \cdot c_{i-1}
$$

This equation can be simplified to:

$$
c_{i}=x_{i} \cdot y_{i}+\left(x_{i}+y_{i}\right) \cdot c_{i-1}=\xi_{i}+a_{i} \cdot c_{i-1}
$$

The " $a_{i}$ " term in the equation being the "alive" bit.
The later form of the equation uses an OR gate instead of an XOR which is a more efficient gate when implemented in CMOS technology. Note that:

$$
a_{i}=\overline{k_{i}}
$$

Where $k_{i}$ is the "kill" bit defined in the table above.

## Binary addition as a prefix sum problem.

- We define a new operator:
- Input is a vector of pairs of 'propagate' and 'generate' bits:

$$
\left(g_{n}, p_{n}\right)\left(g_{n-1}, p_{n-1}\right) \ldots\left(g_{0}, p_{0}\right)
$$

- Output is a new vector of pairs:

$$
\left(G_{n}, P_{n}\right)\left(G_{n-1}, P_{n-1}\right) \ldots\left(G_{0}, P_{0}\right)
$$

- Each pair of the output vector is calculated by the following definition:

$$
\left(G_{i}, P_{i}\right)=\left(g_{i}, p_{i}\right) \circ\left(G_{i-1}, P_{i-1}\right)
$$

Where:

$$
\begin{aligned}
& \left(G_{0}, P_{0}\right)=\left(g_{0}, p_{0}\right) \\
& \left(g_{x}, p_{x}\right) \circ\left(g_{y}, p_{y}\right)=\left(g_{x}+p_{x} \cdot g_{y}, p_{x} \cdot p_{y}\right) \\
& \text { with }+, \quad \text { being the OR, AND operations }
\end{aligned}
$$

## 1973: Kogge-Stone adder



- The Kogge-Stone adder has:
$\square$ Low depth
$\square$ High node count (implies more area).
$\square$ Minimal fan-out of 1 at each node (implies faster performance).


## Summary

- A parallel prefix adder can be seen as a 3-stage process:

- There exist various architectures for the carry calculation part.
- Trade-offs in these architectures involve the
$\square$ area of the adder
$\square$ its depth
$\square$ the fan-out of the nodes
$\square \quad$ the overall wiring network.

