DPHPC: Prefix Scan
Recitation session, 14.11.2019
First project reports have to be sent by tomorrow! Please send directly to your supervisor. And cc Timo 😊

Agenda for today:
- Prefix scan
- Homework
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”

```c
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}
```
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”

```cpp
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}

int findmin(int n, a[n]) {
    for(int i=1; i<n; i++)
        if(a[i]<a[0]) a[0] = a[i];
}
```
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”

```c
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}

int findmin(int n, a[n]) {
    for(int i=1; i<n; i++)
        if(a[i]<a[0]) a[0] = a[i];
}

int finditem(list_t list)
    item = list.head;
    while(item.value!=0 && item.next!=NULL)
        item=item.next;
}
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”

```
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}

int findmin(int n, a[n]) {
    for(int i=1; i<n; i++)
        if(a[i]<a[0]) a[0] = a[i];
}

int finditem(list_t list)
    item = list.head;
    while(item.value!=0 && item.next!=NULL)
        item = item.next;
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LLᵀ)
- Gradient descent
- Model checking: exhaustive verification
An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs.

Recap: oblivious algorithms

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition ($A = LL^T$)
- Gradient descent
- Model checking: exhaustive verification
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs.”

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition ($A = LL^T$)
- Gradient descent
- Model checking: exhaustive verification

**NO.** Memory locations and instructions depend on data distribution.
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs.”

---

**NO.** Memory locations and instructions depend on data distribution.  
**YES.**

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LLᵀ)
- Gradient descent
- Model checking: exhaustive verification
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs.”

### Example Algorithms

- **Insert sort**
- **Prefix sum on an array**
- **Dense Cholesky decomposition** (\(A = LL^T\))
- **Gradient descent**
- **Model checking: exhaustive verification**

### Code Examples

```c
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}

int findmin(int n, a[n]) {
    for(int i=1; i<n; i++)
        if(a[i]<a[0]) a[0] = a[i];
}
```

### Algorithm Example

#### Dense Cholesky Decomposition

- **Partition** 
  \[
  A ightarrow \begin{bmatrix} \frac{A_{TL}}{A_{TR}} \\ \frac{A_{BL}}{A_{BR}} \end{bmatrix}
  \]

- **Repartition** 
  \[
  \begin{bmatrix} \frac{A_{TL}}{A_{TR}} \\ \frac{A_{BL}}{A_{BR}} \end{bmatrix} \rightarrow \begin{bmatrix} \frac{A_{00}}{\alpha_{11}} & \frac{a_{01}}{\alpha_{11}} & \frac{A_{22}}{\alpha_{11}} \\ \frac{\alpha_{11}}{\alpha_{11}} & \frac{a_{11}}{\alpha_{11}} & \frac{A_{22}}{\alpha_{11}} \\ \frac{\alpha_{11}}{\alpha_{11}} & \frac{\alpha_{11}}{\alpha_{11}} & \frac{A_{22}}{\alpha_{11}} \end{bmatrix}
  \]

- **Where** \(\alpha_{11}\) is a scalar

#### Gradient Descent

- **Initial values**
  - \(a_{01} := A_{00}^{-T}a_{01}\) (Trsv)
  - \(\alpha_{11} := \alpha_{11} - \frac{\alpha_{11}}{\alpha_{11}}a_{01}\) (Dot)
  - \(\alpha_{11} := \sqrt{\alpha_{11}}\)

#### Continue with

\[
\begin{bmatrix} \frac{A_{TL}}{A_{TR}} \\ \frac{A_{BL}}{A_{BR}} \end{bmatrix} \rightarrow \begin{bmatrix} \frac{A_{00}}{\alpha_{11}} & \frac{a_{01}}{\alpha_{11}} & \frac{A_{22}}{\alpha_{11}} \\ \frac{\alpha_{11}}{\alpha_{11}} & \frac{a_{11}}{\alpha_{11}} & \frac{A_{22}}{\alpha_{11}} \\ \frac{\alpha_{11}}{\alpha_{11}} & \frac{\alpha_{11}}{\alpha_{11}} & \frac{A_{22}}{\alpha_{11}} \end{bmatrix}
\]

### Verification

**NO.** Memory locations and instructions depend on data distribution.

**YES.**
Recap: oblivious algorithms

"An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs."

```
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}

int findmin(int n, a[n]) {
    for(int i=1; i<n; i++)
        if(a[i]<a[0]) a[0] = a[i];
}

int finditem(list_t list)
    item = list.head;
    while(item.value!=0 && item.next!=NULL)
        item=item.next;
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LLᵀ)
- Gradient descent
- Model checking: exhaustive verification

**NO.** Memory locations and instructions depend on data distribution.

**YES.**
Recap: oblivious algorithms

"An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs."

```c
int reduce(int n, arr[n]) {
  for(int i=0; i<n; ++i)
    sum += arr[i];
}

int findmin(int n, a[n]) {
  for(int i=1; i<n; i++)
    if(a[i]<a[0]) a[0] = a[i];
}

int finditem(list_t list)
  item = list.head;
  while(item.value!=0 && item.next!=NULL)
    item = item.next;
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LLᵀ)
- Gradient descent
- Model checking: exhaustive verification

**NO.** Memory locations and instructions depend on data distribution.

**YES.**

**YES.**

**NO.** Stopping condition depends on numerical result.
Recap: oblivious algorithms

“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs.”

```
int reduce(int n, arr[n]) {
    for(int i=0; i<n; ++i)
        sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
    for(int i=1; i<n; i++)
        if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
    item = list.head;
    while(item.value!=0 && item.next!=NULL)
        item=item.next;
}
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LLᵀ)
- Gradient descent
- Model checking: exhaustive verification

**NO.** Memory locations and instructions depend on data distribution.
**YES.**
**YES.**
**NO.** Stopping condition depends on numerical result.
**NO.** Counterexample position depends on model.
Prefix scan

Known as “scan”, “prefix scan”, “prefix sum”

\[ W(n) = n - 1 \]
\[ D(n) = n - 1 \]
Exclusive scan

How to make an exclusive scan from an inclusive?
How to make an inclusive scan from an exclusive one?
Blelloch (tree-based) scan

```plaintext
procedure down-sweep(A)
    a[n - 1] ← 0
    % Set the identity
    for d from (lg n) - 1 downto 0
        in parallel for i from 0 to n - 1 by 2^{d+1}
            t ← a[i + 2^d - 1]
            % Save in temporary
            a[i + 2^d - 1] ← a[i + 2^{d+1} - 1]
            % Set left child
            a[i + 2^{d+1} - 1] ← t + a[i + 2^{d+1} - 1]
            % Set right child
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Vector in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[3 1 7 0 4 1 6 3]</td>
</tr>
<tr>
<td>up</td>
<td>[3 4 7 7 4 5 6 9]</td>
</tr>
<tr>
<td>2</td>
<td>[3 4 7 11 4 5 6 14]</td>
</tr>
<tr>
<td>3</td>
<td>[3 4 7 11 4 5 6 25]</td>
</tr>
<tr>
<td>clear</td>
<td>[3 4 7 11 4 5 6 0]</td>
</tr>
<tr>
<td>down</td>
<td>[3 4 7 0 4 5 6 11]</td>
</tr>
<tr>
<td>6</td>
<td>[3 0 7 4 4 11 6 16]</td>
</tr>
<tr>
<td>7</td>
<td>[0 3 4 11 11 15 16 22]</td>
</tr>
</tbody>
</table>
```

G. Blelloch, “Prefix Sums and Their Applications”

(b) Executing a +-prescan on a PRAM.
Blelloch (tree-based) scan

```
procedure down-sweep(A)
    a[n - 1] ← 0  % Set the identity

for d from (lg n) - 1 downto 0
    in parallel for i from 0 to n - 1 by 2^d+1
        t ← a[i + 2^d - 1]  % Save in temporary
        a[i + 2^d - 1] ← a[i + 2^d+1 - 1]  % Set left child
        a[i + 2^d+1 - 1] ← t + a[i + 2^d+1 - 1]  % Set right child
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Vector in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[3 1 7 0 4 1 6 3]</td>
</tr>
<tr>
<td>up</td>
<td>1 [3 4 7 7 4 5 6 9]</td>
</tr>
<tr>
<td></td>
<td>2 [3 4 7 11 4 5 6 14]</td>
</tr>
<tr>
<td></td>
<td>3 [3 4 7 11 4 5 6 25]</td>
</tr>
<tr>
<td>clear</td>
<td>4 [3 4 7 11 4 5 6 0]</td>
</tr>
<tr>
<td>down</td>
<td>5 [3 4 7 0 4 5 6 11]</td>
</tr>
<tr>
<td></td>
<td>6 [3 0 7 4 4 11 6 16]</td>
</tr>
<tr>
<td></td>
<td>7 [0 3 4 11 11 15 16 22]</td>
</tr>
</tbody>
</table>

Downsweep uses:
- a) aggregated result
- b) last left child result
Ladner-Fischer scan

Efficient work for any depth $D(n) = \log(n) + k$

N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”
Ladner-Fischer scan

Efficient work for any depth $D(n) = \log(n) + k$

N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”
Ladner-Fischer scan

Efficient work for any depth $D(n) = \log(n) + k$

N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”
Ladner-Fischer scan

Efficient work for any depth $D(n) = \log(n) + k$

Broadcast

N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”
Parallel scan

What if $p \ll n$?

1. Buy more processors? $\log(n)$ runtime 😐
Parallel scan

What if $p << n$?

1. Buy more processors? $\log(n)$ runtime 😞

2. Run multiple iterations?
Parallel scan

What if $p \ll n$?

1. Buy more processors? $\log(n)$ runtime 😞

2. Run multiple iterations?

3. Split work and minimize global exchange to $\log(p)$ 😊
Parallel scan
Parallel scan

Prefix Scan Operator

Local Phase Reduce
Parallel scan

1. Local Phase Reduce
2. Global Phase Prefix Scan

Prefix Scan Operator

MPI
Parallel scan

1. Local Phase Reduce
   - Prefix Scan Operator

2. Global Phase Prefix Scan

3. Local Phase Prefix Scan
Parallel scan

Scan—then—reduce also possible!

Which one is better?

1. Local Phase Reduce

2. Global Phase Prefix Scan

3. Local Phase Prefix Scan
Scan in message passing

**Pipelined** binary tree – split message of size $n$ into $k$ packets.
Scan in message passing

**Pipelined** binary tree – split message of size \( n \) into \( k \) packets.

Communication time dominates for simple operators!

\[
(2 \log_2 p - 2) \left( T_{\text{start}} + n \times T_{\text{byte}} \right)
\]
**Scan in message passing**

**Pipelined** binary tree – split message of size $n$ into $k$ packets.

Communication time dominates for simple operators!

\[
(2 \log_2 p - 2)(T_{\text{start}} + n \times T_{\text{byte}})
\]

\[
(4 \times \log_2 p - 2 + 6(k - 1))(T_{\text{start}} + \frac{n}{k} \times T_{\text{byte}})
\]
Parallel filter

Given an array, produce an array containing only elements for which predicate is true.

input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]

f(e): true if e > 10

output [17, 11, 13, 19, 24]

Can we parallelize that?
Parallel filter

- Parallel map with filter $f$
  
  ```python
def f(i):
    if bitsum[i]:
      output[bitsum[i] - 1] = input[i]
```

- Parallel prefix sum on bit vector
  
  ```
  input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]
  bits [1, 0, 0, 0, 1, 0, 1, 1, 0, 1]
  bitsum [1, 1, 1, 1, 2, 2, 3, 4, 4, 5]
  output [17, 11, 13, 19, 24]
  ```

- Parallel map with filter $f$
  
  ```
  $D(n) = 1$
  ```
Parallel lexical analysis

INPUT  (  f  o  o  space  +  "  s  "  )

Whitespace
Identifier
Character token
Quote strings

Finite-State Machine
Parallel lexical analysis

Map appropriate transition for each character

<table>
<thead>
<tr>
<th>Letter</th>
<th>Character Token</th>
<th>Quote</th>
<th>White Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>A</td>
<td>Q</td>
<td>N</td>
</tr>
<tr>
<td>A</td>
<td>Z</td>
<td>Q</td>
<td>N</td>
</tr>
<tr>
<td>Z</td>
<td>*</td>
<td>Q</td>
<td>N</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>E</td>
<td>N</td>
</tr>
<tr>
<td>E</td>
<td>*</td>
<td>S</td>
<td>N</td>
</tr>
</tbody>
</table>

Fig. 3. The state of the processors after the input `(foo + "s")` has been initialised.
Parallel lexical analysis

Compose transitions: $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ produces $\alpha \rightarrow \gamma$

<table>
<thead>
<tr>
<th>f</th>
<th>O</th>
<th>O</th>
<th>&lt;space</th>
<th>+</th>
<th>&quot;&quot;</th>
<th>S</th>
<th>&quot;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>N → *, N → A</td>
<td>N → A</td>
<td>N → A</td>
<td>N → N</td>
<td>N → *</td>
<td>N → Q</td>
<td>N → A</td>
<td>N → C</td>
</tr>
<tr>
<td>A → Z</td>
<td>A → Z</td>
<td>A → Z</td>
<td>Z → N</td>
<td>Z → *</td>
<td>Z → Q</td>
<td>Z → Z</td>
<td>Z → S</td>
</tr>
<tr>
<td>* → *</td>
<td>* → A</td>
<td>* → A</td>
<td>* → N</td>
<td>* → *</td>
<td>* → A</td>
<td>* → A</td>
<td>* → S</td>
</tr>
<tr>
<td>Q → S</td>
<td>Q → S</td>
<td>Q → S</td>
<td>Q → S</td>
<td>Q → E</td>
<td>Q → S</td>
<td>Q → S</td>
<td>Q → S</td>
</tr>
<tr>
<td>E → E</td>
<td>E → E</td>
<td>E → E</td>
<td>E → N</td>
<td>E → *</td>
<td>E → S</td>
<td>E → E</td>
<td>E → S</td>
</tr>
</tbody>
</table>

Data initialised from column 1 of figure 2

The left and right processing elements compose their state transition tables and the result is stored in the right of the two elements. This result signifies that the machine would be in state A or S after reading the previous and current characters (i.e "(f)")
Parallel lexical analysis

Parallel prefix sum!

Last value is FSM state after reading the entire input

During the fourth and final iteration, the last processor will combine its contents with the processor eight places to its left.

Fig. 5. The fourth and final iteration of the parallel prefix technique.

| INPUT | ( | f | o | o | space | + | " | S | " | ) |
|-------|----|----|----|------|----|----|----|----|----|
| Result State | * | A | Z | Z | N | * | Q | S | E | * |
| Comment | Single char. token | 'A' denotes the start of a identifier token , and Z' corresponds to the continuation of that token | Single char token | Single char token | 'Q' and 'E' denote the start & end of a quotes token, & S denotes the sentence within the quotes |

"Parallel lexical analysis and parsing on the AMT distributed array processor." Parallel computing 18.6 (1992)
Radix sort

• Idea: split keys into multiple digits according to the radix

| 101 | 111 | 010 | 011 | 110 | 001 |
Radix sort

- Idea: split keys into multiple digits according to the radix

<table>
<thead>
<tr>
<th></th>
<th>101</th>
<th>111</th>
<th>010</th>
<th>011</th>
<th>110</th>
<th>001</th>
</tr>
</thead>
</table>

- In each iteration sort according to a single digit.

|   | 101 | 111 | 010 | 011 | 110 | 001 |
Radix sort

• Idea: split keys into multiple digits according to the radix

```
101  111  010  011  110  001
```

• In each iteration sort according to a single digit.

```
101  111  010  011  110  001
```

```
010  110
```
Radix sort

• Idea: split keys into multiple digits according to the radix

```
101  111  010  011  110  001
```

• In each iteration sort according to a single digit.

```
101  111  010  011  110  001
```

```
010  110  101  111  011  001
```
Radix sort

- Idea: split keys into multiple digits according to the radix

```
101  111  010  011  110  001
```

- In each iteration sort according to a single digit.

```
101  111  010  011  110  001
```
```
010  110  101  111  011  001
```

- Stability (preserve the order inside group) + process each digit => full sort!

```
101  001  010  110  111  011
```
```
001  010  011  101  110  111
```
Radix sort

- Idea: split keys into multiple digits according to the radix

| 101 | 111 | 010 | 011 | 110 | 001 |

- In each iteration sort according to a single digit.

| 101 | 111 | 010 | 011 | 110 | 001 |

- Stability (preserve the order inside group) + process each digit => full sort!

| 101 | 001 | 010 | 110 | 111 | 011 |
| 001 | 010 | 011 | 101 | 110 | 111 |
Radix sort

<table>
<thead>
<tr>
<th>101</th>
<th>111</th>
<th>010</th>
<th>011</th>
<th>110</th>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Radix sort

1. Build k-element histogram of each digit.

<table>
<thead>
<tr>
<th>Digit</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

k digits
k elements
Radix sort

1. Build \( k \)-element histogram of each digit.

<table>
<thead>
<tr>
<th>Digit</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

2. Exclusive prefix scan on histogram

<table>
<thead>
<tr>
<th>Digit</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Radix sort

1. Build k-element histogram of each digit.

2. Exclusive prefix scan on histogram

3. Exclusive prefix scan on each group of digits
Radix sort

1. Build $k$-element histogram of each digit.

<table>
<thead>
<tr>
<th>Digit</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

2. Exclusive prefix scan on histogram

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

3. Exclusive prefix scan on each group of digits

| 0 | 1 | 0 | 2 | 1 | 3 |

4. Summarize to get new indices

```
0+2
1+2
0+0
2+2
1+0
3+2
```
Image Registration

Electron microscopy frame $f_0$

The image after deformation (magnification). $\tilde{f}_{25} \cdot \phi_{0,25}$
Image Registration
Image Registration

Not every prefix scan problem is...

... trivial to compute
... stable
... perfectly associative
Linked list prefix scan

Recap: breaking symmetry

• We can run parallel computation on independent sets. How to find them?

• Introduce randomness to create local differences!
  • Each node tosses a coin → 0 or 1
  • Let $I$ be the set of nodes such that $v$ drew 1 and $v.next$ drew 0!
    • What is the probability that $v \in I$?
Linked list prefix scan

Recap: breaking symmetry

• We can run parallel computation on independent sets. How to find them?

• Introduce randomness to create local differences!
  • Each node tosses a coin \( \rightarrow 0 \text{ or } 1 \)
  • Let \( I \) be the set of nodes such that \( v \) drew 1 and \( v.\text{next} \) drew 0!
    • What is the probability that \( v \in I \)?
      \[
      P(v \in I) = \frac{1}{4}
      \]
Linked list prefix scan - upward

P0    P1    P2    P3    P4
Linked list prefix scan - upward

P0  P1  P2  P3  P4

0   1   1   1   0

5   1   6   3   7
Linked list prefix scan - upward

P0  P1  P2  P3  P4

5  1  6  3  7

5  1  11  10  7

☺  1  1  0  ☺
Linked list prefix scan - upward

P0    P1    P2    P3    P4

5     1     6     3     7

5     1     11    10    7

5     1     21    10    7

☺     0     1     ☺     ☺
Linked list prefix scan - upward
Linked list prefix scan - downward

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>
Linked list prefix scan - downward

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>

5 -> 1 -> 22 -> 10 -> 7
## Linked list prefix scan - downward

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>11</td>
<td>7</td>
</tr>
</tbody>
</table>
Linked list prefix scan - downward

P0  P1  P2  P3  P4

5 → 1 → 22 → 10 → 7

5 → 1 → 22 → 11 → 8
Linked list prefix scan - downward

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>22</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>22</td>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>