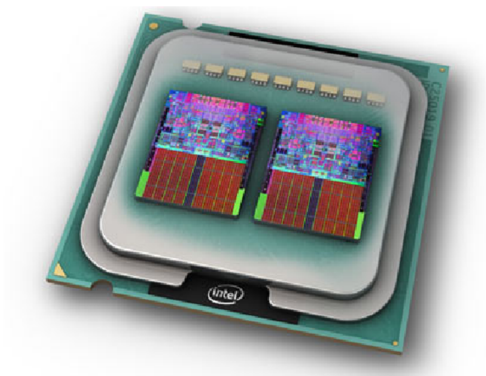
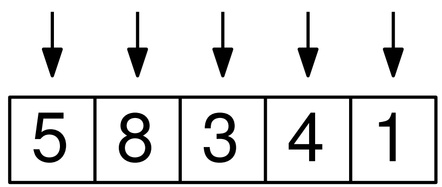
INFORMATION & COMPUTER SCIENCES
UNIVERSITY of HAWAIʻI at MĀNOA

AlgoPARC

ICS 621: Analysis of Algorithms

Prof. Nodari Sitchinava





Parallel Algorithms

Slides vs Hand-writing

Simple example

for i = 1 to n **do**
    $a[i] = a[i] + 1$

5	8	3	4	1

Simple example

for $i = 1$ to n **do**

$\quad a[i] = a[i] + 1$

5	8	3	4	1

$\qquad i = 1$

L: $a[i] = a[i] + 1$

$\qquad i = i + 1$

$\qquad$ **if** $i \leq n$: $\textsc{JumpTo}$ L

Simple example

$i = 1$

5	8	3	4	1

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$$\begin{aligned} & i = 1 \\ \text{L:}\quad & a[i] = a[i] + 1 \\ & i = i + 1 \\ & \textbf{if } i \leq n\text{: } \textsc{JumpTo L} \end{aligned}$$

Simple example

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i = 1$

6	8	3	4	1

$i = 1$

L: $a[i] = a[i] + 1$

$i = i + 1$

if $i \leq n$: JUMPTO L

Simple example

for i = 1 to n **do**
  $a[i] = a[i] + 1$

i = 2

6	8	3	4	1

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

for $i = 1$ to n **do**
$\quad a[i] = a[i] + 1$

$i =$ 2

6	9	3	4	1

$\quad i = 1$
L: $a[i] = a[i] + 1$
$\quad i = i + 1$
$\quad$ **if** $i \leq n$: JUMPTO L

Simple example

for $i = 1$ to n **do**

$a[i] = a[i] + 1$

$i =$ 3

6	9	3	4	1

$i = 1$

L: $a[i] = a[i] + 1$

$i = i + 1$

if $i \leq n$: JUMPTO L

Simple example

$i =$ 3

6	9	4	4	1

for $i = 1$ to n **do**
    $a[i] = a[i] + 1$

```
    i = 1
L:  a[i] = a[i] + 1
    i = i + 1
    if i ≤ n: JUMPTO L
```

Simple example

for i = 1 to n **do**
  $a[i] = a[i] + 1$

i = 4

6	9	4	4	1

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i =$ 4

6	9	4	5	1

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

i = 5

6	9	4	5	1

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

   $i = 1$
L: $a[i] = a[i] + 1$
   $i = i + 1$
   if $i \leq n$: JUMPTO L

Simple example

$i =$ 5

6	9	4	5	2

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

for i = 1 to n **do**
$a[i] = a[i] + 1$

$i =$ 5

6	9	4	5	2

Time

$O(n)$

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

Time

for i = 1 to n **do**
  $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

for i = 1 to n **in parallel do**
  $a[i] = a[i] + 1$

Simple example

Time

for $i = 1$ to n **do**
    $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

for $i = 1$ to n **in parallel do**
    $a[i] = a[i] + 1$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
    $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

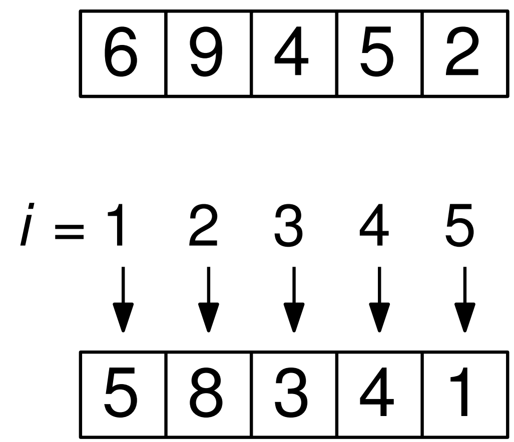
5	8	3	4	1

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
    $a[i] = a[i] + 1$



$O(n)$

for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
    $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

$i = 1 \quad 2 \quad 3 \quad 4 \quad 5$

6	9	4	5	2

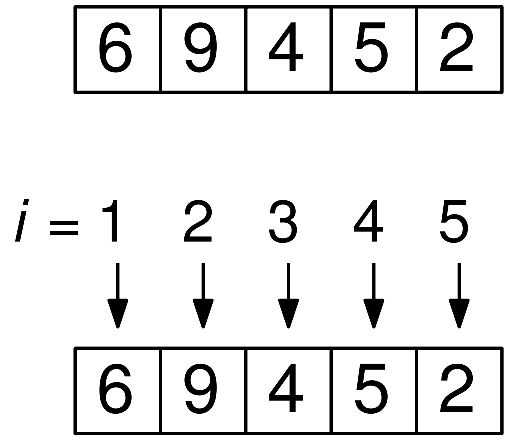
for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
    $a[i] = a[i] + 1$



$O(n)$

for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

$O(1)$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
    $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

i = 1 2 3 4 5

6	9	4	5	2

$O(1)$

for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Parallel Time = time of the slowest thread

More complex example: Prefix Sums

for $i = 2$ to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$

5	8	3	4	1

More complex example: Prefix Sums

for $i = 2$ to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$

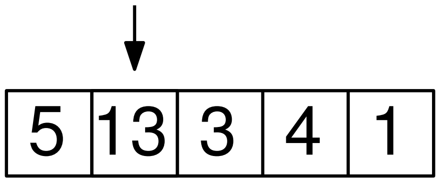
5	8	3	4	1

More complex example: Prefix Sums

for $i = 2$ to n **do**

$\quad a[i] = a[i] + a[i-1]$

return $a[n]$



More complex example: Prefix Sums

for i = 2 to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$

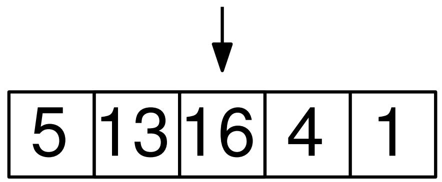
5	13	3	4	1

More complex example: Prefix Sums

for $i = 2$ to n **do**

$\quad a[i] = a[i] + a[i-1]$

return $a[n]$



More complex example: Prefix Sums

for i = 2 to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	4	1

More complex example: Prefix Sums

for $i = 2$ to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$

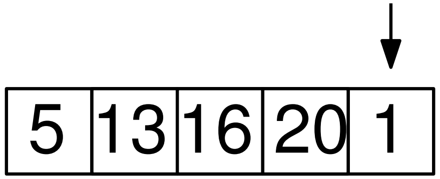
5	13	16	20	1

More complex example: Prefix Sums

for i = 2 to n **do**

$\quad a[i] = a[i] + a[i-1]$

return $a[n]$

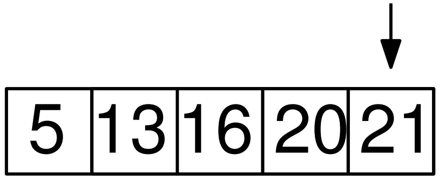


More complex example: Prefix Sums

for $i = 2$ to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$



More complex example: Prefix Sums

for $i = 2$ to n **do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

Time

$O(n)$

More complex example: Prefix Sums

Time

for $i = 2$ to n **do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$O(n)$

for $i = 2$ to n **in parallel do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	8	3	4	1

Start $n - 1$ threads $t_2, \ldots t_n$
Each thread t_i (where $i = 2, \ldots, n$) **do**:
  $a[i] = a[i] + a[i-1]$

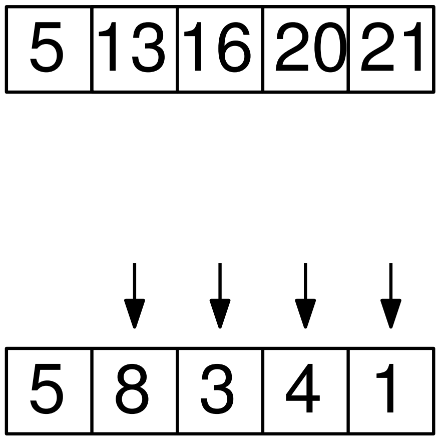
More complex example: Prefix Sums

Time

for i = 2 to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$



$\boxed{O(n)}$

for i = 2 to n **in parallel do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$

Start $n-1$ threads $t_2, \ldots t_n$

Each thread t_i (where $i = 2, \ldots, n$) **do**:

  $a[i] = a[i] + a[i-1]$

More complex example: Prefix Sums

Time

for $i = 2$ to n **do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$O(n)$

for $i = 2$ to n **in parallel do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	11	7	5

Start $n-1$ threads $t_2, \ldots t_n$
Each thread t_i (where $i = 2, \ldots, n$) **do**:
  $a[i] = a[i] + a[i-1]$

More complex example: Prefix Sums

Time

for $i = 2$ to n **do**

$a[i] = a[i] + a[i-1]$

return $a[n]$

5	13	16	20	21

$O(n)$

for $i = 2$ to n **in parallel do**

$a[i] = a[i] + a[i-1]$

return $a[n]$

5	13	11	7	5

$O(1)$

Start $n-1$ threads $t_2, \ldots t_n$

Each thread t_i (where $i = 2, \ldots, n$) **do**:

$a[i] = a[i] + a[i-1]$

Parallel Prefix Sums

5	8	3	4	1

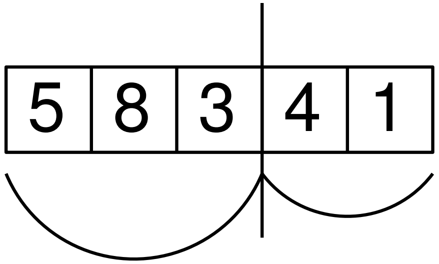
Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums



Parallel Prefix Sums

5	8	3	4	1

5	8	3

4	1

Parallel Prefix Sums

5	8	3	4	1

5	13	16

4	5

Parallel Prefix Sums

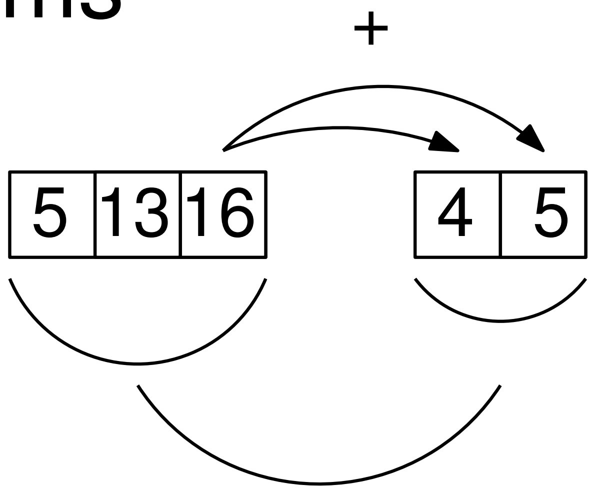
5	8	3	4	1

5	13	16

4	5

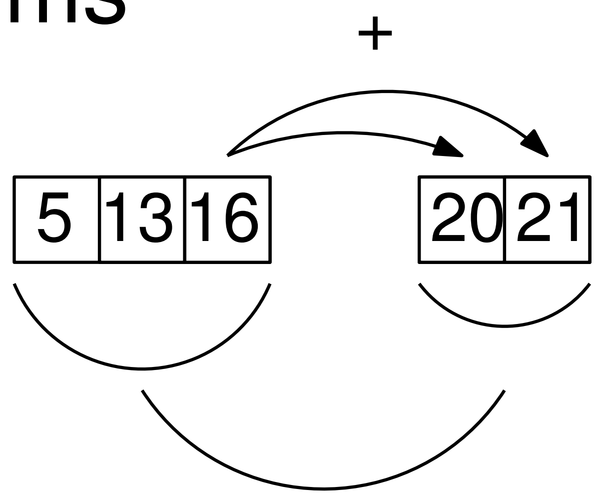
Parallel Prefix Sums

5	8	3	4	1

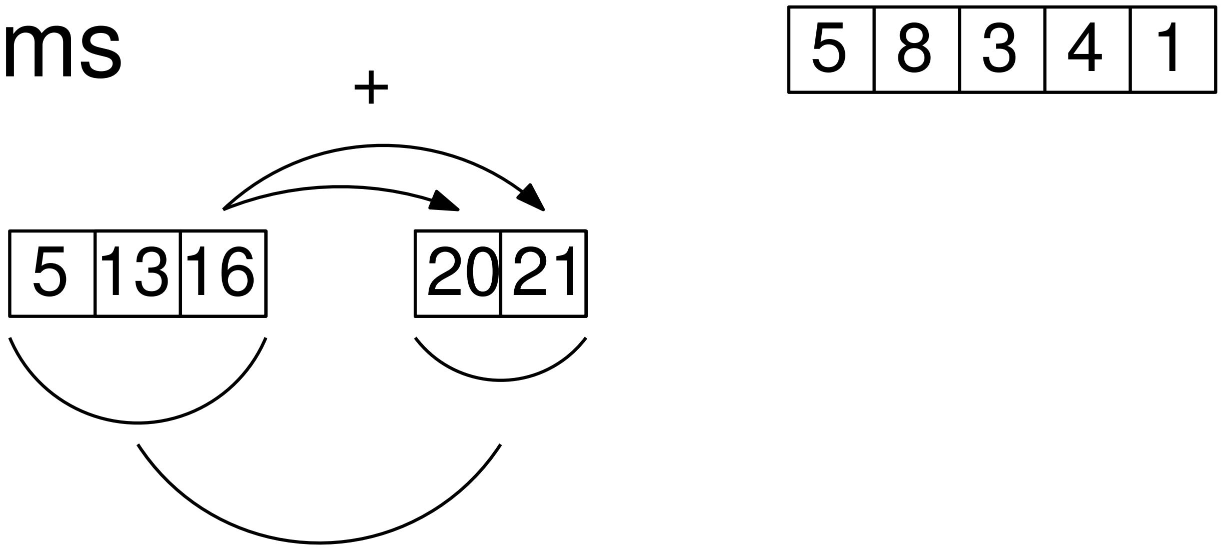


Parallel Prefix Sums

5	8	3	4	1



Parallel Prefix Sums

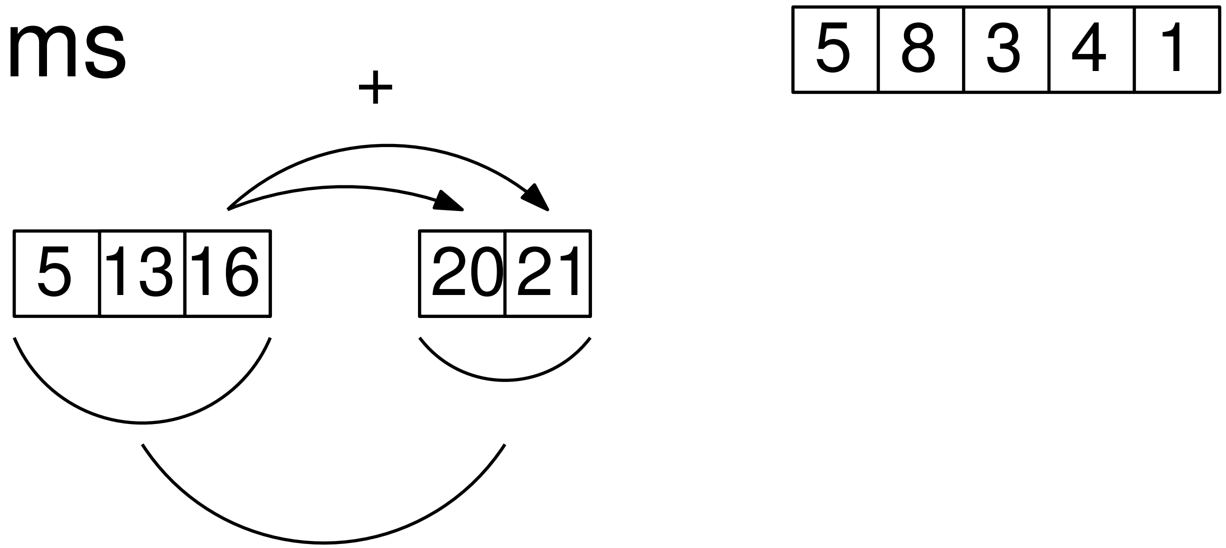


function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  PREFIX-SUMS(A, i, mid)
  PREFIX-SUMS($A, mid + 1, j$)

  for $k = mid + 1$ to j **do**
    $A[k] = A[k] + A[mid]$

Parallel Prefix Sums



function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  PREFIX-SUMS(A, i, mid)

  PREFIX-SUMS($A, mid + 1, j$)

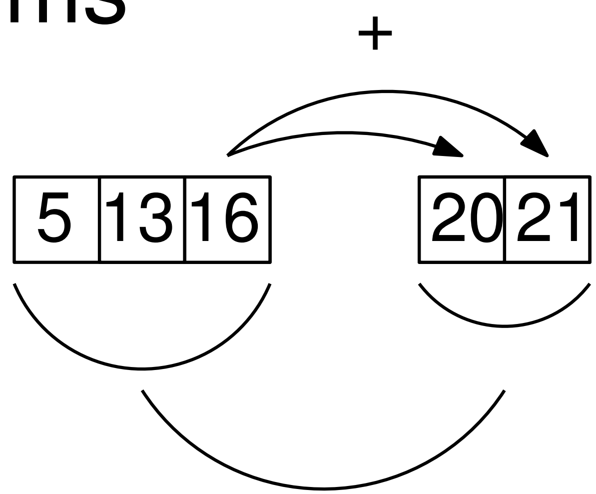
  for $k = mid + 1$ to j **do**

    $A[k] = A[k] + A[mid]$

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Prefix Sums

5	8	3	4	1



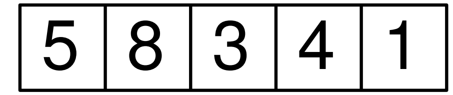
function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

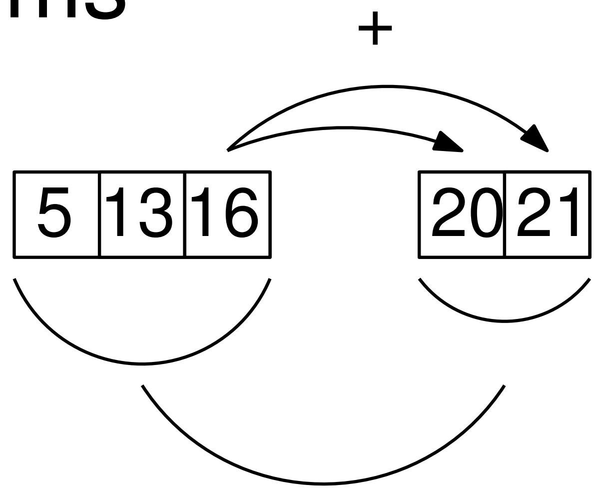
  PREFIX-SUMS(A, i, mid)
  PREFIX-SUMS($A, mid + 1, j$)

  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Prefix Sums





function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  PREFIX-SUMS(A, i, mid)

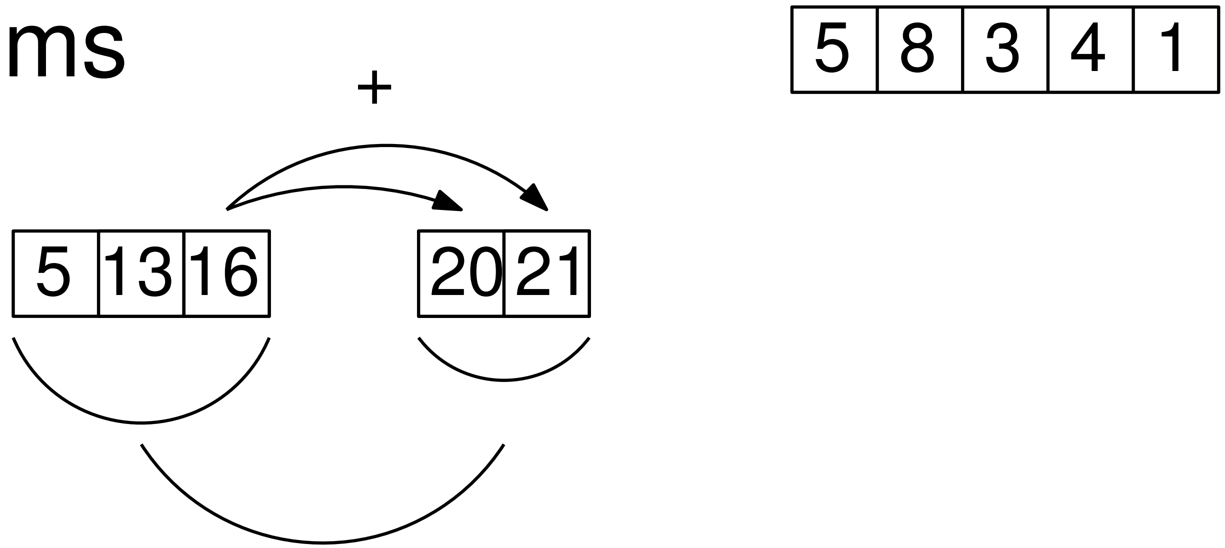
  PREFIX-SUMS($A, mid+1, j$)

  for $k = mid + 1$ to j **in parallel do**

    $A[k] = A[k] + A[mid]$

$$\begin{aligned} T(n) &= 2T(n/2) + O(1) \\ &= O(n) \end{aligned}$$

Parallel Prefix Sums

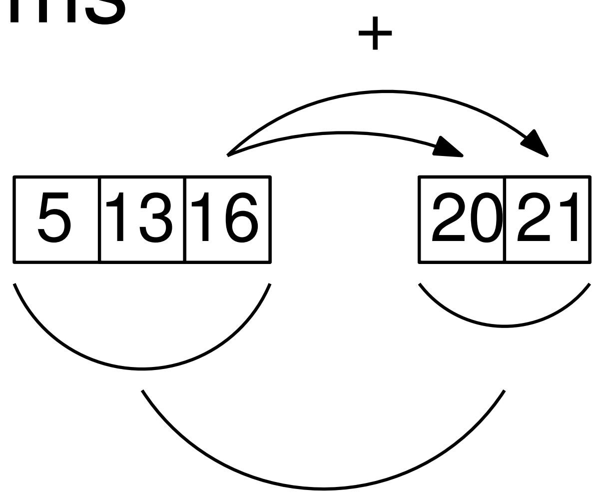


function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    spawn
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS$(A, mid + 1, j)$
    sync
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

$$\begin{aligned} T(n) &= 2T(n/2) + O(1) \\ &= O(n) \end{aligned}$$

Parallel Prefix Sums

5	8	3	4	1



function $\text{PREFIX-SUMS}(A, i, j)$

  if $i \geq j$ **then return**

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  spawn

    $\text{PREFIX-SUMS}(A, i, mid)$

    $\text{PREFIX-SUMS}(A, mid + 1, j)$

  sync

  for $k = mid + 1$ to j **in parallel do**

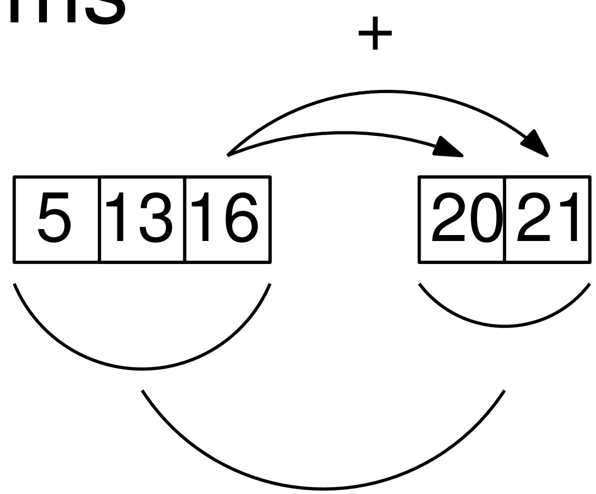
    $A[k] = A[k] + A[mid]$

$(t_1, t_2) = \text{STARTTWOTHREADS}()$
t_1 **do**: $\text{PREFIX-SUMS}(A, i, mid)$
t_2 **do**: $\text{PREFIX-SUMS}(A, mid + 1, j)$
$\text{WAITUNTILFINISHED}(t_1, t_2)$

$$\begin{aligned} T(n) &= 2T(n/2) + O(1) \\ &= O(n) \end{aligned}$$

Parallel Prefix Sums

5	8	3	4	1



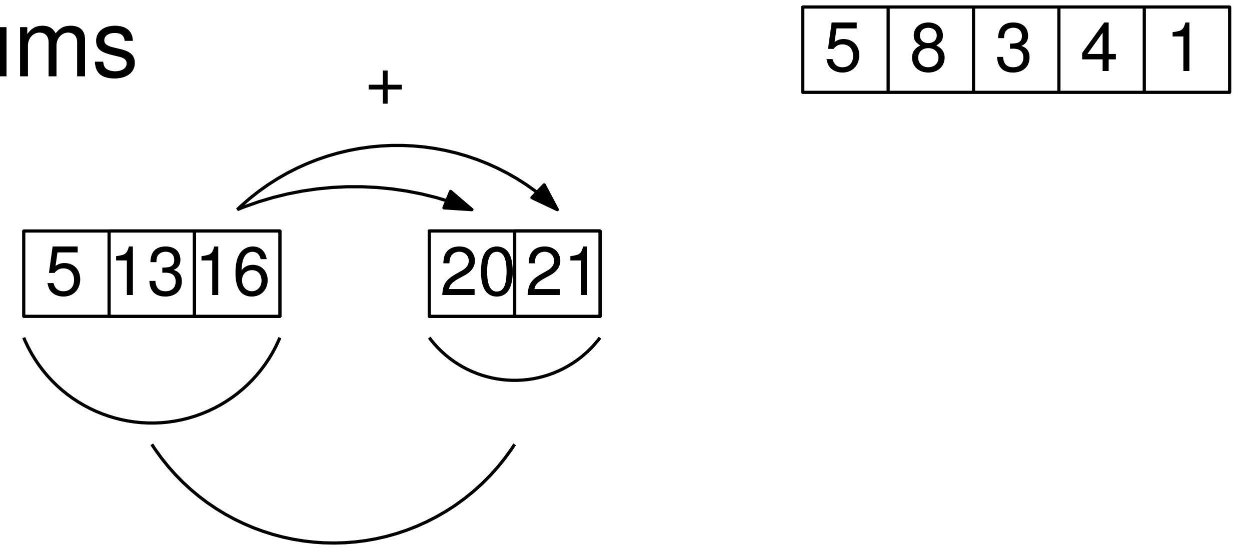
function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return**
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    spawn
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS$(A, mid + 1, j)$
    sync
    for $k = mid + 1$ to j **in parallel**
        $A[k] = A[k] + A[mid]$

$(t_1, t_2) =$ STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS$(A, mid + 1, j)$
WAITUNTILFINISHED(t_1, t_2)

$$T(n) = \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} + O(1)$$

$$\leq \underline{\quad} T(n/2) + O(1)$$

Parallel Prefix Sums



function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return**
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS$(A, mid + 1, j)$
  sync
  for $k = mid + 1$ to j **in parallel**
    $A[k] = A[k] + A[mid]$

(t_1, t_2) = STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS$(A, mid + 1, j)$
WAITUNTILFINISHED(t_1, t_2)

$$T(n) = \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} + O(1)$$

$$\leq \_T(n/2) + O(1)$$

$$= O(\log n)$$

Recursion vs. parallel **for** loop

Recursion vs. parallel **for** loop

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync

  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

Recursion vs. parallel **for** loop

function $\text{Prefix-Sums}(A, i, j)$
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    for k = 1 to 2 **in parallel do**
        if k = 1 **then**
            $\text{Prefix-Sums}(A, i, mid)$
        else
            $\text{Prefix-Sums}(A, mid + 1, j)$
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

Parallel Sorting

Parallel Sorting

function MergeSort(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  MergeSort(A, i, mid)
  MergeSort($A, mid + 1, j$)

  Merge(A, i, mid, j)

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  MERGESORT(A, i, mid)
  MERGESORT($A, mid + 1, j$)

  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= 2T(n/2) + T_{\text{MERGE}} \\ &= 2T(n/2) + O(n) \end{aligned}$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  MERGESORT(A, i, mid)
  MERGESORT($A, mid + 1, j$)

  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= 2T(n/2) + T_{\text{MERGE}} \\ &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** $\triangleright$ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do $\{$
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  $\}$
  MERGE(A, i, mid, j)

$$
\begin{aligned}
T(n) &= 2T(n/2) + T_{\textsc{Merge}} \\
&= 2T(n/2) + O(n) \\
&= O(n \log n)
\end{aligned}
$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do $\{$
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  $\}$
  MERGE(A, i, mid, j)

$$T(n) = \_T(n/2) + T_{\text{MERGE}}$$
$$= \_T(n/2) + O(n)$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= \_\,T(n/2) + T_{\text{MERGE}} \\ &= \_\,T(n/2) + O(n) \\ &= O(n) \end{aligned}$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$T(n) = \_\,T(n/2) + T_{\text{MERGE}}$$
$$= \_\,T(n/2) + O(\log n)$$

With parallel merging

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= \_\, T(n/2) + T_{\text{MERGE}} \\ &= \_\, T(n/2) + O(\log n) \\ &= O(\log^2 n) \end{aligned}$$

With parallel merging

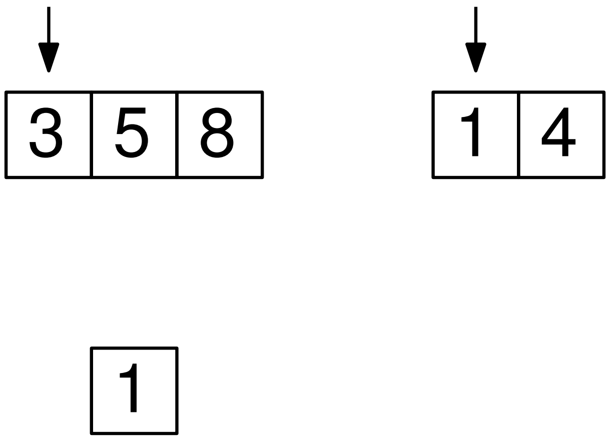
Parallel Merging

3	5	8

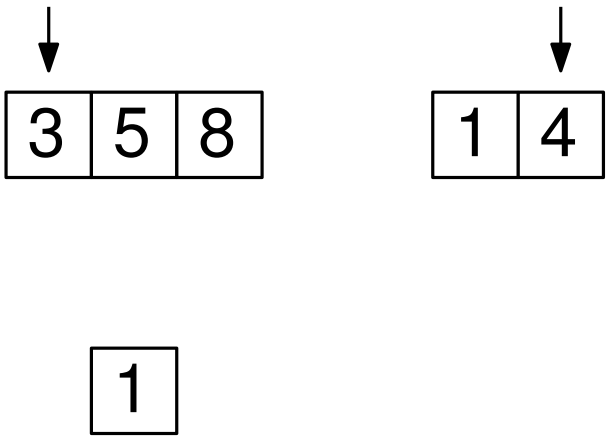
1	4

Parallel Merging

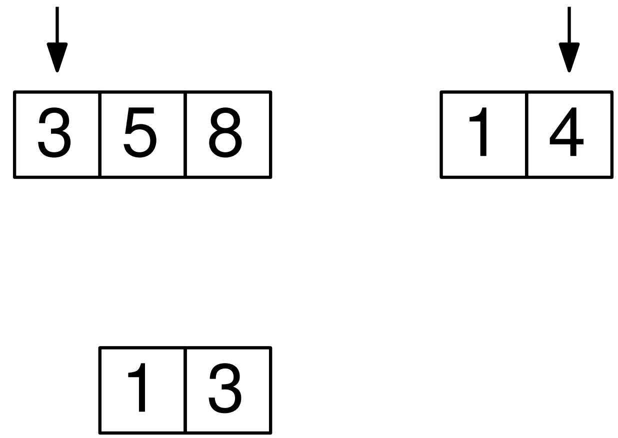
Parallel Merging



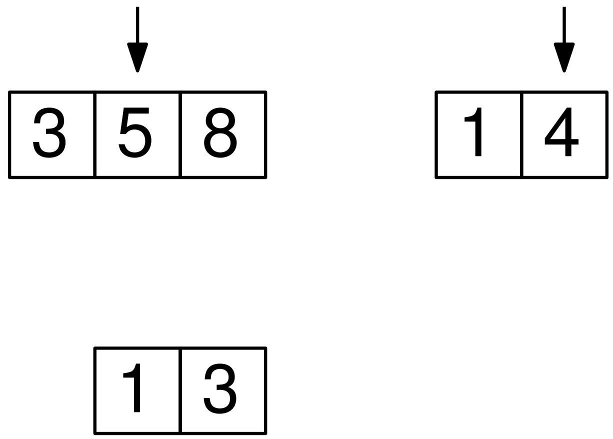
Parallel Merging



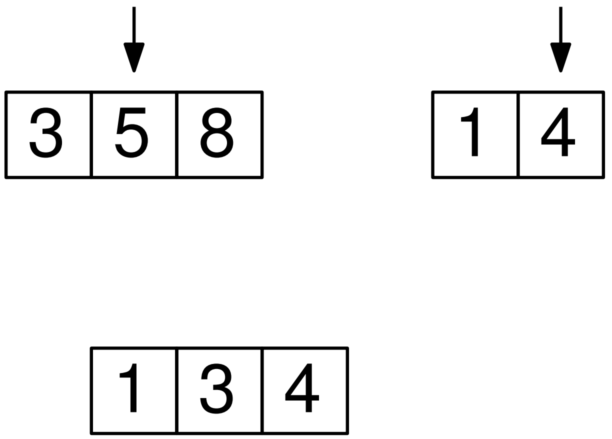
Parallel Merging



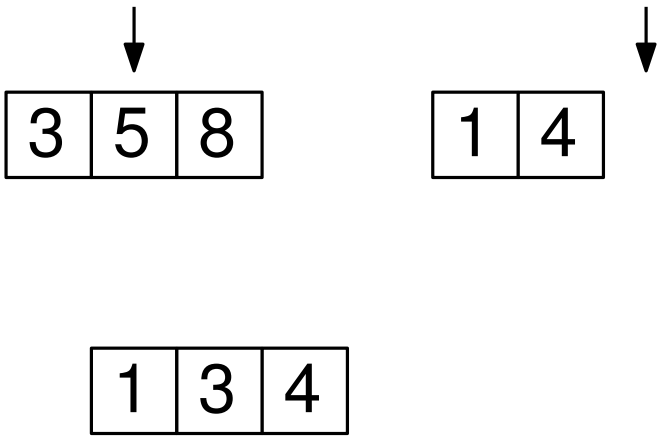
Parallel Merging



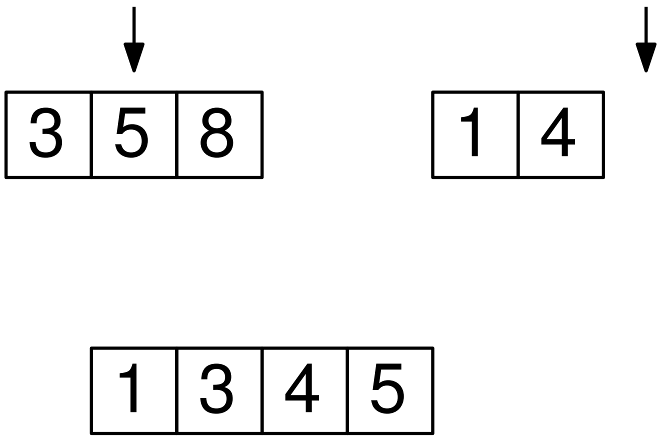
Parallel Merging



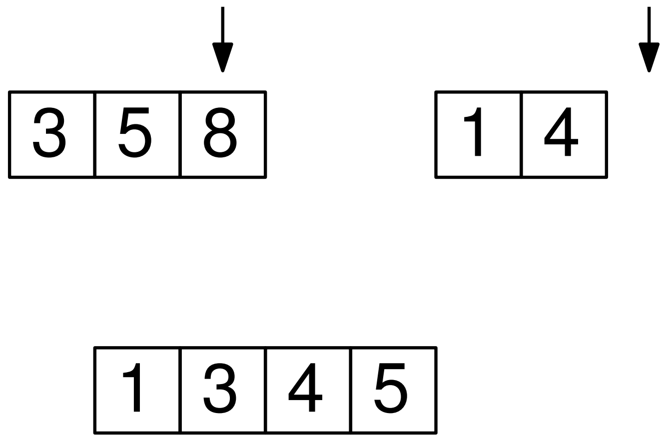
Parallel Merging



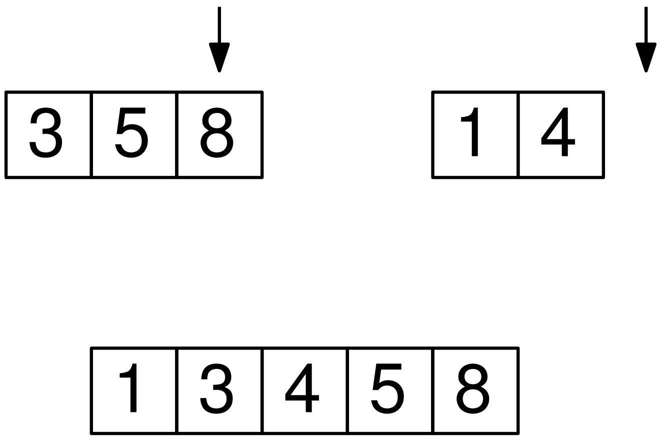
Parallel Merging



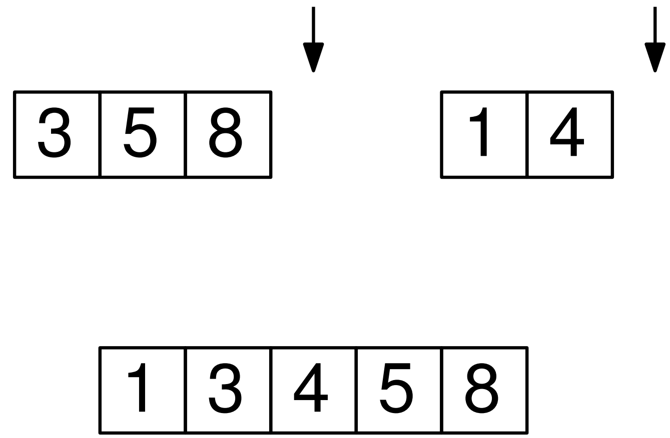
Parallel Merging



Parallel Merging



Parallel Merging



Parallel Merging

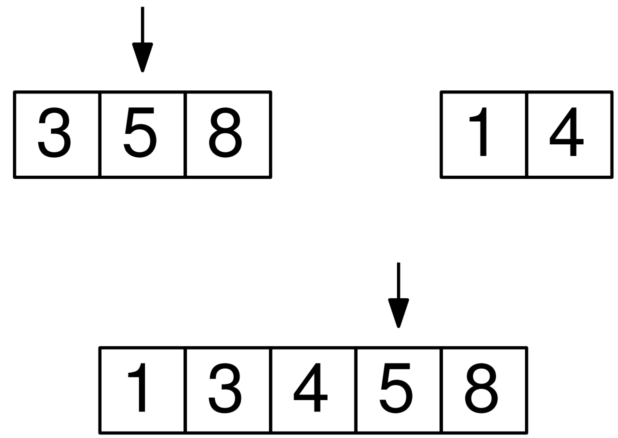
3	5	8

1	4

$O(n)$

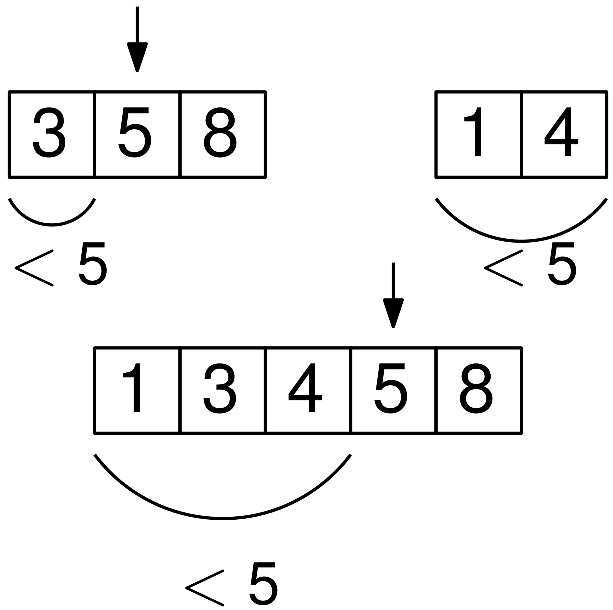
1	3	4	5	8

Parallel Merging



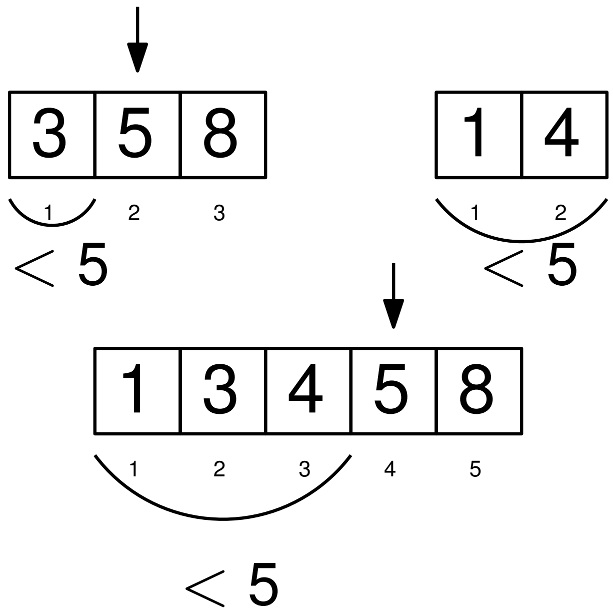
$O(n)$

Parallel Merging



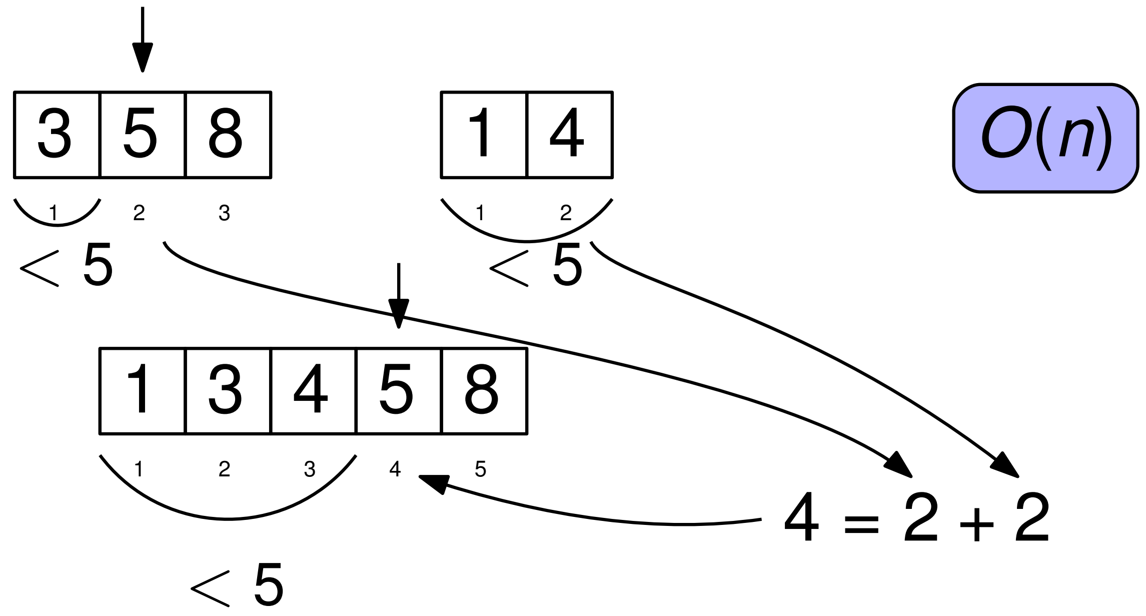
$O(n)$

Parallel Merging

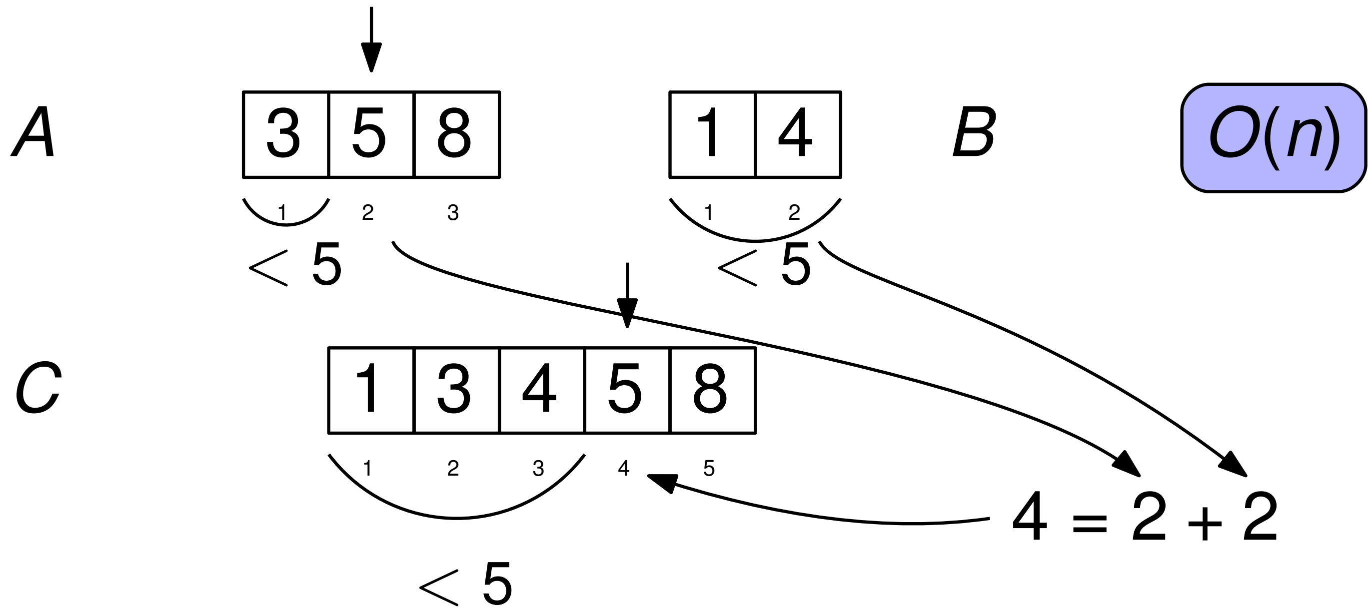


$O(n)$

Parallel Merging



Parallel Merging



function MERGE(A, B, C)

  for i = 1 to $|A|$ **in parallel do**

    $k = i +$ PREDECESSOR($A[i], B$)

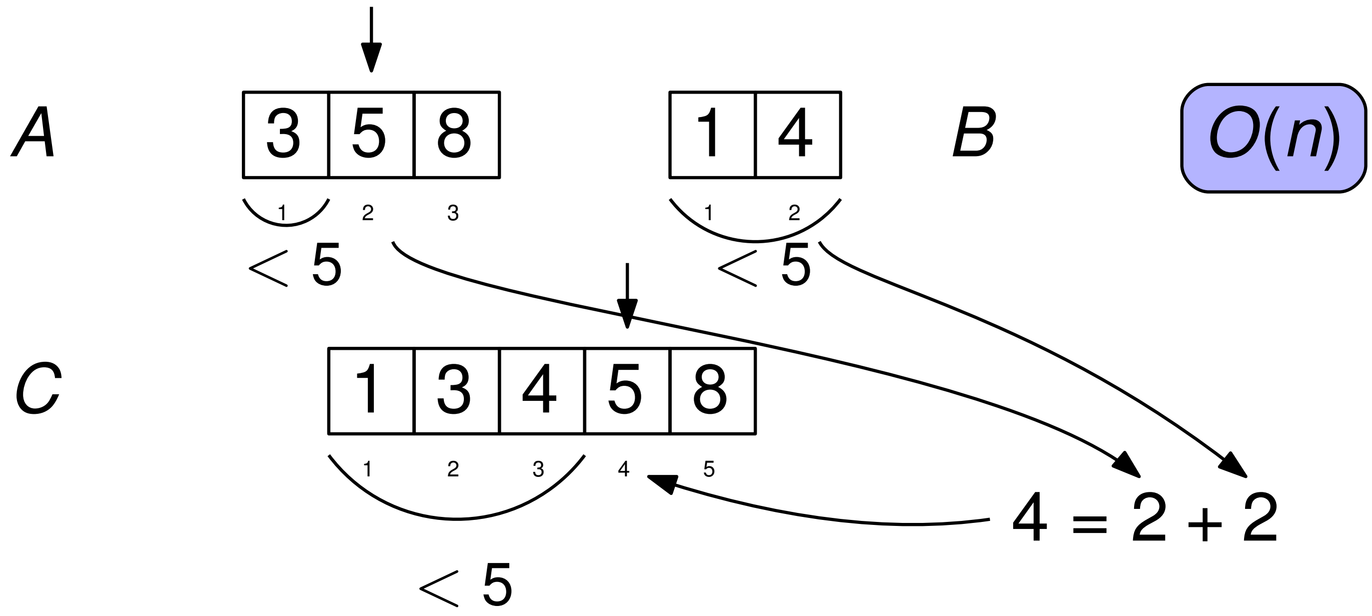
    $C[k] = A[i]$

  for j = 1 to $|B|$ **in parallel do**

    $k = j +$ PREDECESSOR($B[j], A$)

    $C[k] = B[j]$

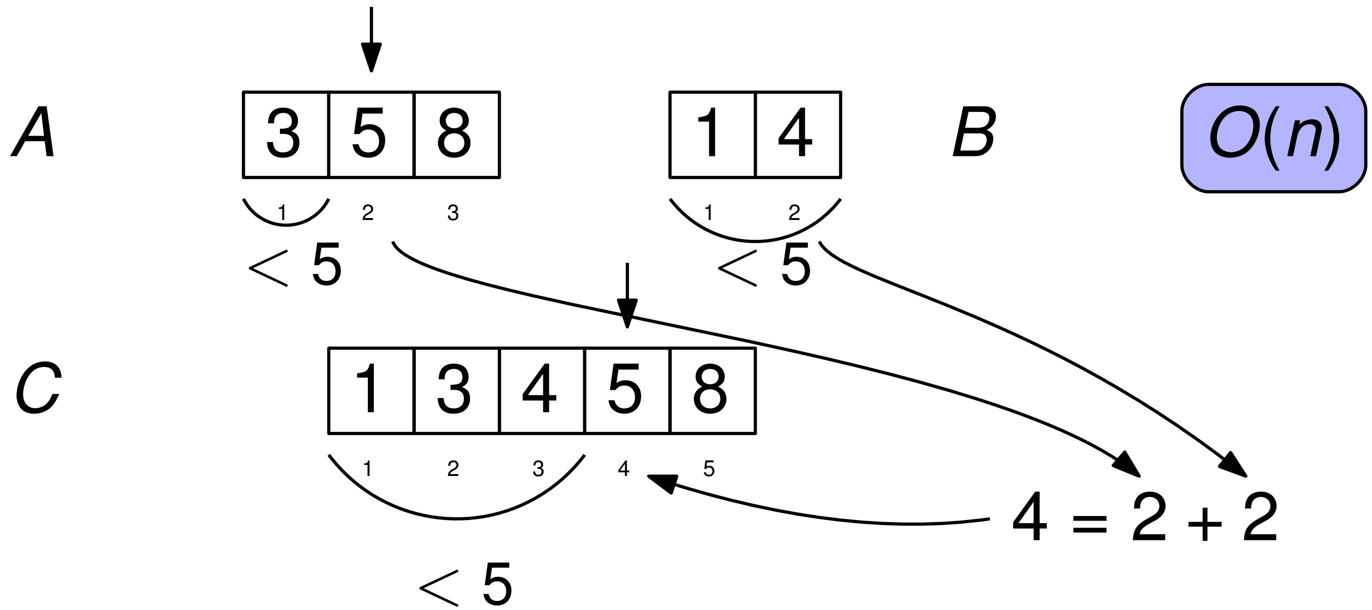
Parallel Merging



function MERGE(A, B, C)
 for i = 1 to $|A|$ **in parallel do**
  $k = i +$ PREDECESSOR($A[i]$, B)
  $C[k] = A[i]$
 for j = 1 to $|B|$ **in parallel do**
  $k = j +$ PREDECESSOR($B[j]$, A)
  $C[k] = B[j]$

function PREDECESSOR(x, A)
 for i = 1 to $|A|$ **do**
  if $A[i] > x$ **then**
   return $i - 1$
 return $|A|$

Parallel Merging

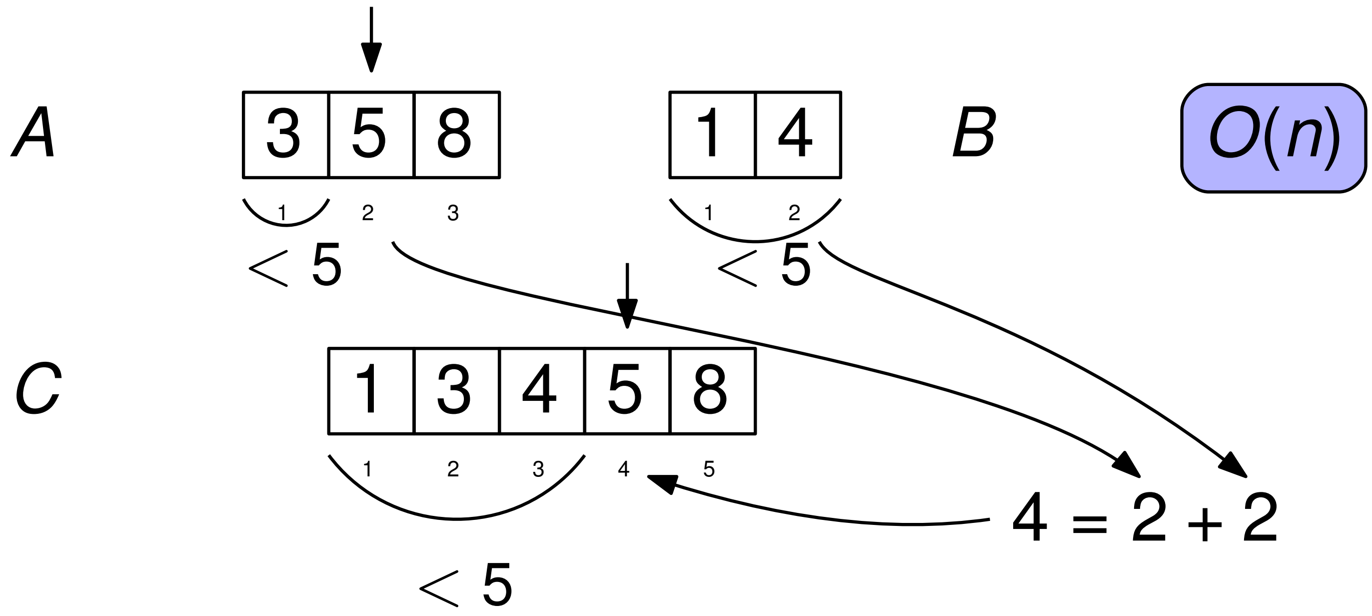


function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i +$ PREDECESSOR$(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j +$ PREDECESSOR$(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  for $i = 1$ to $|A|$ **do**
    if $A[i] > x$ **then**
      return $i - 1$
  return $|A|$

Still $O(n)$

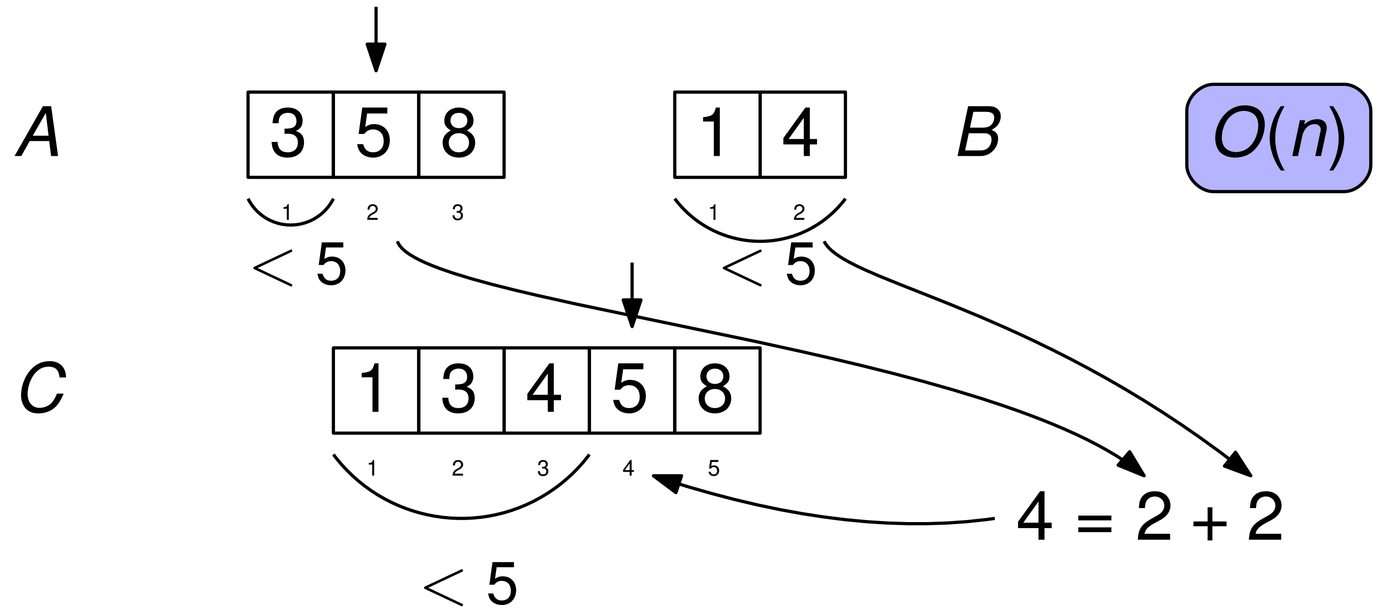
Parallel Merging




```
function MERGE(A, B, C)
    for i = 1 to |A| in parallel do
        k = i + PREDECESSOR(A[i], B)
        C[k] = A[i]
    for j = 1 to |B| in parallel do
        k = j + PREDECESSOR(B[j], A)
        C[k] = B[j]
```

```
function PREDECESSOR(x, A)
    return BINARYSEARCH(x, A)
```

Parallel Merging

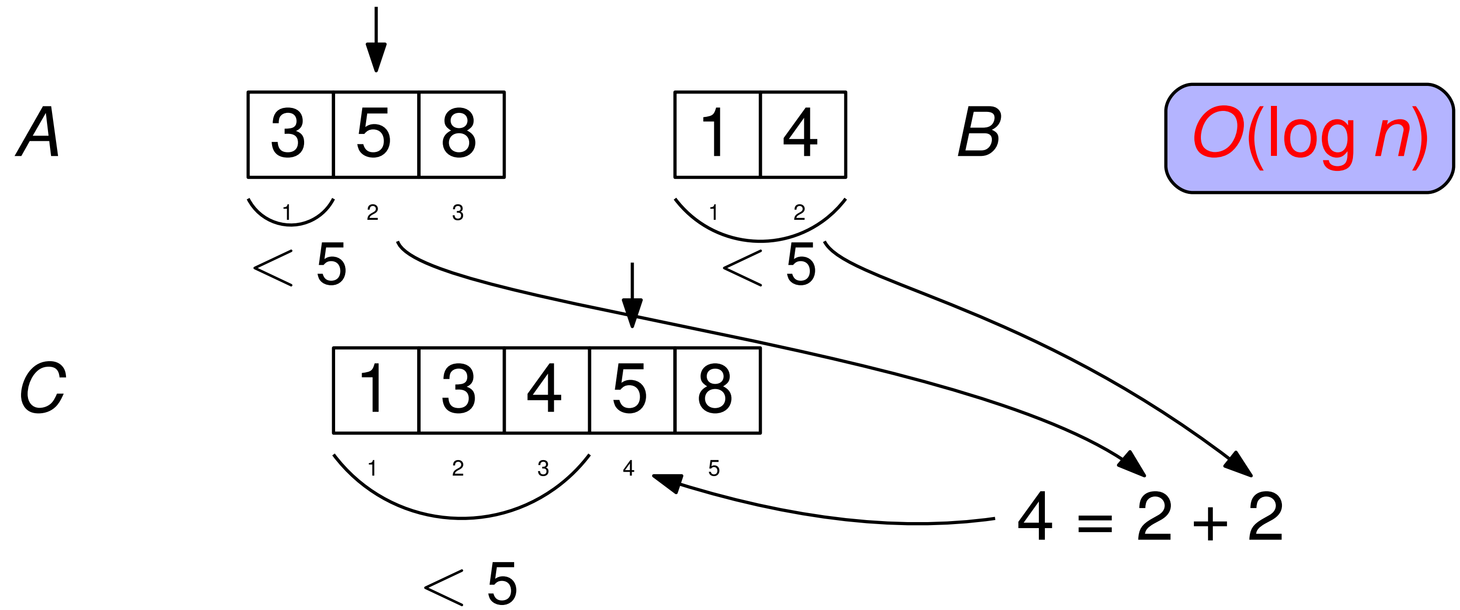


function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

$O(\log n)$

Parallel Merging



function MERGE(A, B, C)
    for $i = 1$ to $|A|$ **in parallel do**
        $k = i +$ PREDECESSOR$(A[i], B)$
        $C[k] = A[i]$
    for $j = 1$ to $|B|$ **in parallel do**
        $k = j +$ PREDECESSOR$(B[j], A)$
        $C[k] = B[j]$

function PREDECESSOR(x, A)
    return BINARYSEARCH(x, A)

$O(\log n)$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

for $i = 2$ to n **in parallel do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

for $i = 2$ to n **in parallel do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

$W = O(n)$
$T_\infty = O(1)$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync
  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS$(A, mid + 1, j)$
  sync
  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

$$W = 2W(n/2) + O(n)$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function $\textsc{Prefix-Sums}(A, i, j)$
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    $\textsc{Prefix-Sums}(A, i, mid)$
    $\textsc{Prefix-Sums}(A, mid + 1, j)$
  sync
  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

$$\boxed{\begin{aligned} W &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}}$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return** $\triangleright$ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    spawn
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS($A, mid + 1, j$)
    sync
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

$$W = 2W(n/2) + O(n) = O(n \log n)$$

$$T_\infty(n) = T_\infty(n/2) + O(1)$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function Prefix-Sums(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    spawn
        Prefix-Sums(A, i, mid)
        Prefix-Sums($A, mid + 1, j$)
    sync
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

$$W = 2W(n/2) + O(n) = O(n \log n)$$

$$T_\infty(n) = T_\infty(n/2) + O(1) = O(\log n)$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

$W(n) = O(\log n)$
$T_\infty(n) = O(\log n)$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

$W(n) = O(n \log n)$
$T_\infty(n) = O(\log n)$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

$W(n) = O(\log n)$
$T_\infty(n) = O(\log n)$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function $\textsc{MergeSort}(A, i, j)$
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do $\{$
        $\textsc{MergeSort}(A, i, mid)$
        $\textsc{MergeSort}(A, mid + 1, j)$
    $\}$
    $\textsc{Merge}(A, i, mid, j)$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function $\textsc{MergeSort}(A, i, j)$
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do $\{$
    $\textsc{MergeSort}(A, i, mid)$
    $\textsc{MergeSort}(A, mid + 1, j)$
  $\}$
  $\textsc{Merge}(A, i, mid, j)$

$$W(n) = 2W(n) + O(n \log n)$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$\boxed{\begin{aligned} W(n) &= 2W(n) + O(n \log n) \\ &= O(n \log^2 n) \end{aligned}}$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function $\textsc{MergeSort}(A, i, j)$
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    $\textsc{MergeSort}(A, i, mid)$
    $\textsc{MergeSort}(A, mid + 1, j)$
  }
  $\textsc{Merge}(A, i, mid, j)$

$$W(n) = 2W(n) + O(n \log n)$$
$$= O(n \log^2 n)$$

$$T_\infty(n) = T_\infty(n/2) + O(\log n)$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$W(n) = 2W(n) + O(n \log n) = O(n \log^2 n)$$

$$T_\infty(n) = T_\infty(n/2) + O(\log n) = O(\log^2 n)$$

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

Work vs Parallel Time

- Work: Total # of operations = Sequential runtime: $W = T_1$
- (Parallel) Time: # of operations of slowest thread: T_∞

$$\boxed{T_p = O\left(\frac{W}{P} + T_\infty\right)}$$