Chapter 3: Parallel Algorithmic Structures II

Elements of Parallel Computing

Eric Aubanel

Divide and Conquer

Three stages:

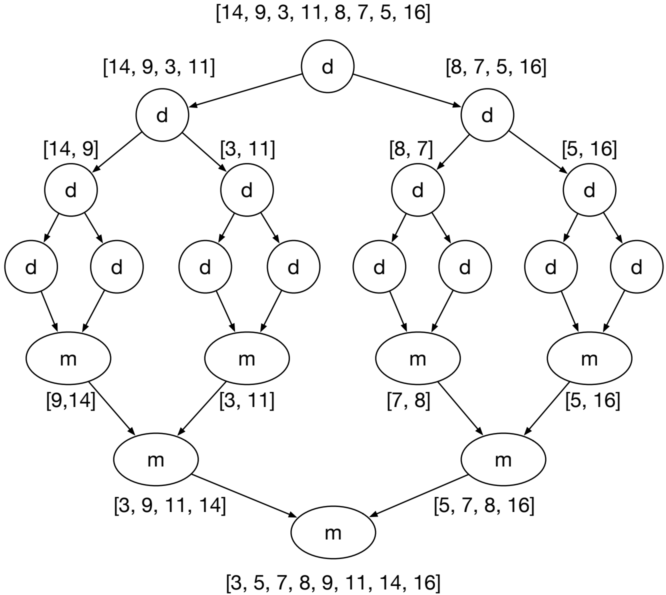
1. splitting into subproblems
2. solving bases cases
3. combining solutions

Input: array a of length n.
Output: array b, containing array a sorted. Array a overwritten.

```
arrayCopy(a, b) // copy array a to b
mergeSort(a, 0, n, b)

// sort elements with index i ∈ [lower..upper)
Procedure mergeSort(a, lower, upper, b)
    if (upper − lower) < 2 then
        return
    end
    mid ← ⌊(upper + lower)/2⌋
    mergeSort(b, lower, mid, a)
    mergeSort(b, mid, upper, a)
    // merge sorted sub-arrays i ∈ [lower..mid) and
        i ∈ [mid..upper) of a into b
    merge(a, lower, mid, upper, b)
    return
end
```

Initial Task Graph for Merge Sort



Input: array $a[lower..upper-1]$, with each subarray $i \in [lower..mid)$ and $i \in [mid..upper)$ sorted in ascending order.

Output: sorted array b

Procedure `merge`(a, $lower$, mid, $upper$, b)

$i \leftarrow lower$

$j \leftarrow mid$

for $k \leftarrow lower$ *to* $upper - 1$ **do**

  if $i < mid \wedge (j \geq upper \vee a[i] \leq a[j])$ **then**

    $b[k] \leftarrow a[i]$

    $i \leftarrow i + 1$

  else

    $b[k] \leftarrow a[j]$

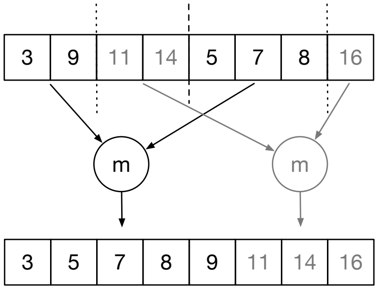
    $j \leftarrow j + 1$

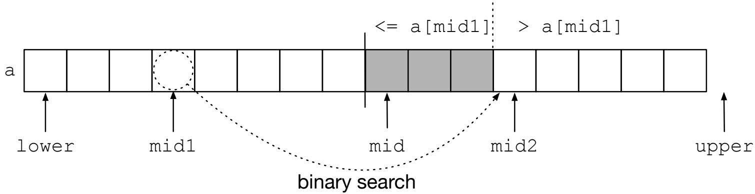
  end

end

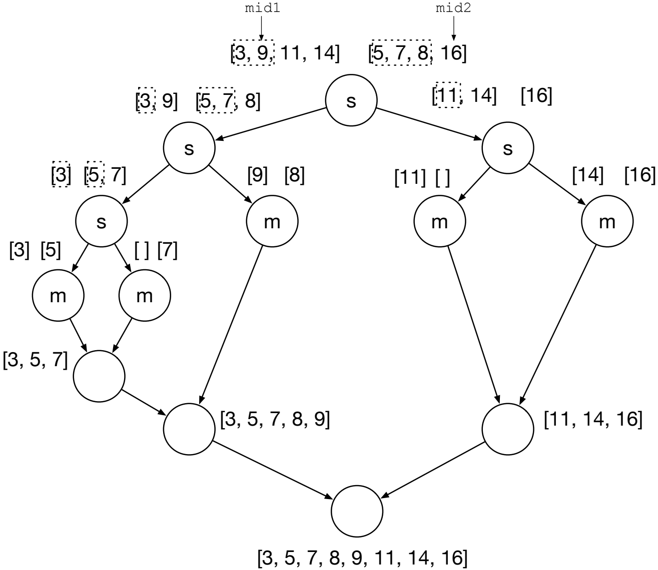
end

Merging With Two Tasks





Divide and Conquer Merge

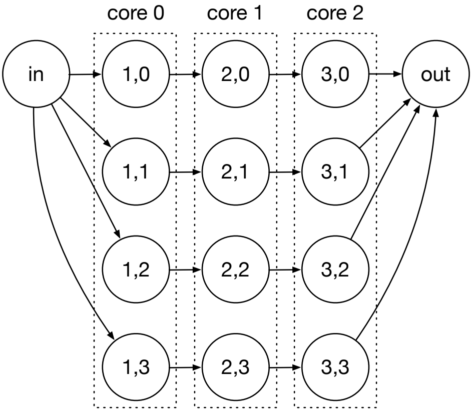


Pipeline

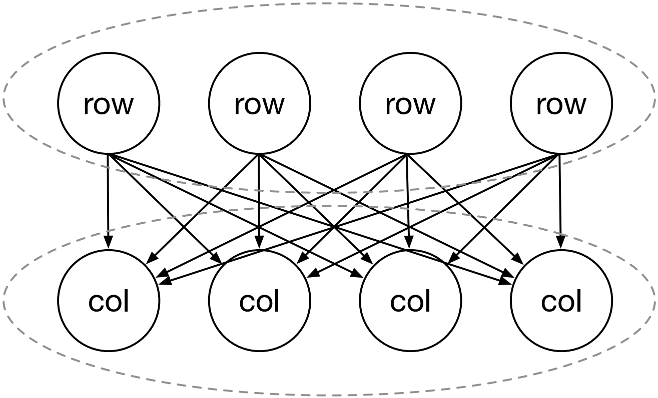
while token *in input* **do**
    $x \leftarrow \texttt{phase1(token)}$
    $y \leftarrow \texttt{phase2}(x)$
    $z \leftarrow \texttt{phase3}(y)$
    output z
end

Pipeline

```
while token in input do
    x ← phase1(token)
    y ← phase2(x)
    z ← phase3(y)
    output z
end
```

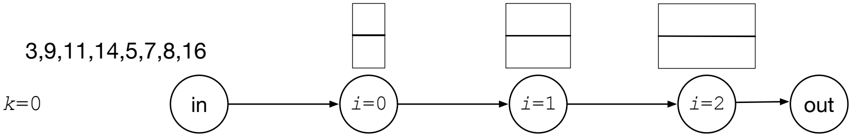


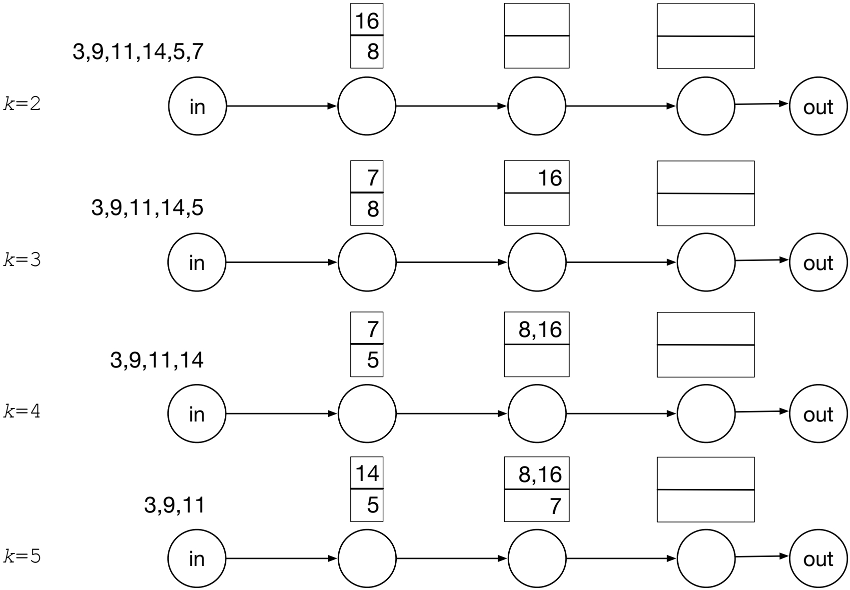
2D FFT in two stages

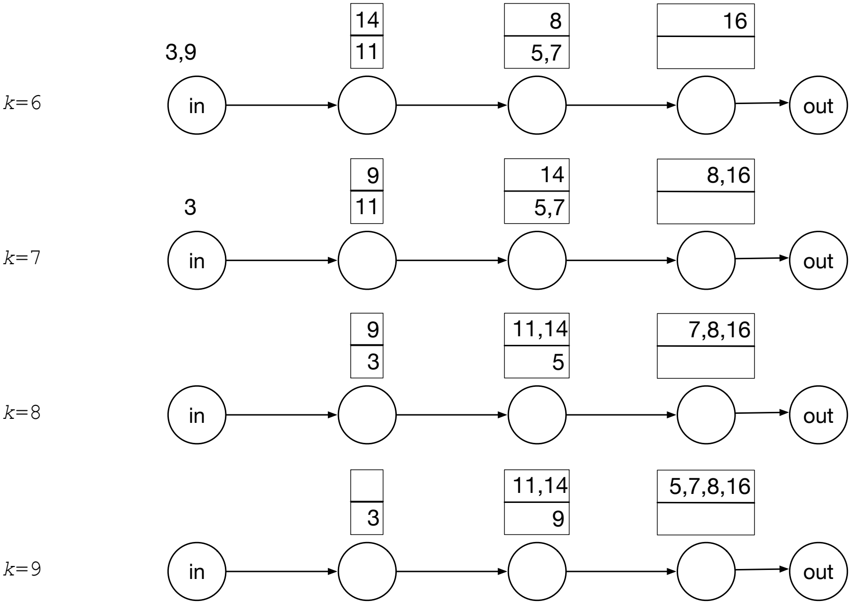


Pipelined Merge Sort

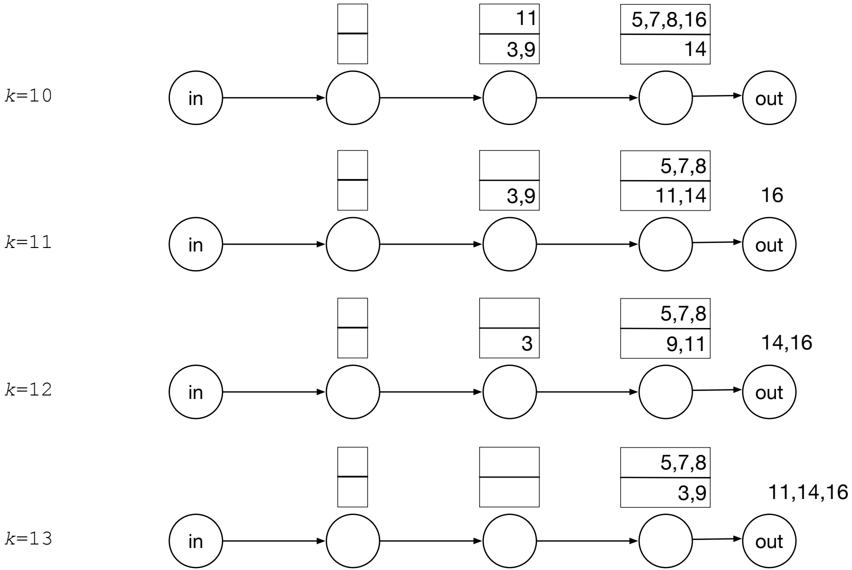
- Agglomerates tasks in each level of merge sort into one task
- merges are sequential
- $\log n$ merges taking place simultaneously when the pipeline is full




k=2
3,9,11,14,5,7
16
8
in
out
k=3
3,9,11,14,5
7
8
16
in
out
k=4
3,9,11,14
7
5
8,16
in
out
k=5
3,9,11
14
5
8,16
7
in
out



k=6
3,9
in
14
11
8
5,7
16
out
k=7
3
in
9
11
14
5,7
8,16
out
k=8
in
9
3
11,14
5
7,8,16
out
k=9
in
3
11,14
9
5,7,8,16
out


k=10
in
out
11
3,9
5,7,8,16
14
k=11
in
out
3,9
5,7,8
11,14
16
k=12
in
out
3
5,7,8
9,11
14,16
k=13
in
out
5,7,8
3,9
11,14,16

```
// m = log n synchronous tasks with index i ∈ [0..m)
Procedure Task i ()
    dir ← 1 // 1:  upper queue, -1:  lower queue
    start ← 0 // can't start merging until enough
        values
    count ← 0, nup ← 0, ndown ← 0
    while value in input or queues not empty do
        // Send head of upper or lower queue to
            ouput (next slide)
        if value exists then
            put value in queue dir
            count ← count + 1
            if count = 2^i then
                dir ← −dir
                count ← 0
            end
        end
    end
end
```

```
// Send head of upper or lower queue to ouput
```

if $(start = 0) \wedge$ (*upper has* 2^i *elements and lower has* 1) **then**

  $start \leftarrow 1$

end

if $start = 1$ **then**

  if $nup = 2^i \wedge ndown = 2^i$ **then** $nup \leftarrow 0$, $ndown \leftarrow 0$

  if $nup = 2^i$ **then**

    send head of lower queue to output

    $ndown \leftarrow ndown + 1$

  else if $ndown = 2^i$ **then**

    send head of upper queue to output

    $nup \leftarrow nup + 1$

  else if *head of upper queue* $\geq$ *than head of lower queue* **then**

    send head of upper queue to output

    $nup \leftarrow nup + 1$

  else

    send head of lower queue to output

    $ndown \leftarrow ndown + 1$

  end

Comparing Two Merges

Divide and conquer and pipeline parallel decompositions merge the same groups of elements.

- Divide and conquer merges groups of the same size in parallel
- pipeline merges groups of different size in parallel, one element at a time.

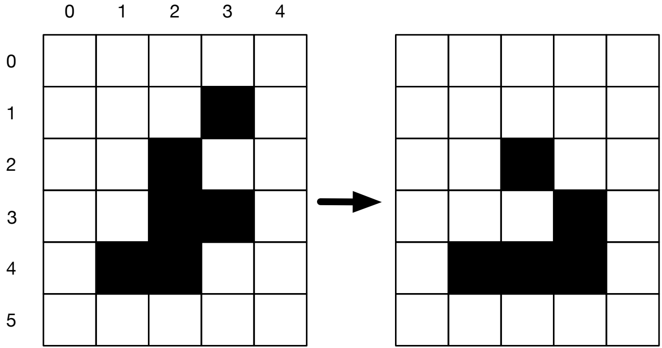
Data Decomposition

Decompose the data structures, then associate a task with each portion.

Four examples:

1. grid application
2. matrix-vector multiplication
3. bottom-up dynamic programming
4. merge sort (next chapter)

Grid Application: Game of Life



- Each cell has eight neighbours
  - Use periodic boundary conditions

Rules

- Each cell visited and its neighbourhood examined to determine whether its state will change in the next generation
- A cell that is alive will only still be alive in the next generation if 2 or 3 neighbours are alive.
- A cell that is dead will be alive in the next generation only if it has three neighbours that are alive.

Game of Life

Input: $n \times n$ grid of cells, each with a state of alive (1) or dead (0).
Output: evolution of grid for a given number of generations

Allocate empty *newGrid*
for *a number of generations* **do**
    Display *grid*
    foreach *cell at coordinate* (i, j) **do**
        `updateGridCell`(*grid*, *newGrid*, *i*, *j*)
    end
    swap references to *newGrid* and *grid*
end

Procedure `updateGridCell`(*grid*, *newGrid*, *i*, *j*)

$sumAlive \leftarrow grid[(i-1+n) \bmod n, (j-1+n) \bmod n]$
$+grid[(i-1+n) \bmod n, j] + grid[(i-1+$
$n) \bmod n, (j+1) \bmod n] + grid[i, (j-1+$
$n) \bmod n] + grid[i, (j+1) \bmod n]$
$+grid[(i+1) \bmod n, (j-1+n) \bmod n] +$
$grid[(i+1) \bmod n, j]$
$+grid[(i+1) \bmod n, (j+1) \bmod n]$

if $grid[i,j] = 0 \wedge sumAlive = 3$ **then**

$newGrid[i,j] \leftarrow 1$

else if $grid[i,j] = 1 \wedge (sumAlive = 2 \vee sumAlive = 3)$
then

$newGrid[i,j] \leftarrow 1$

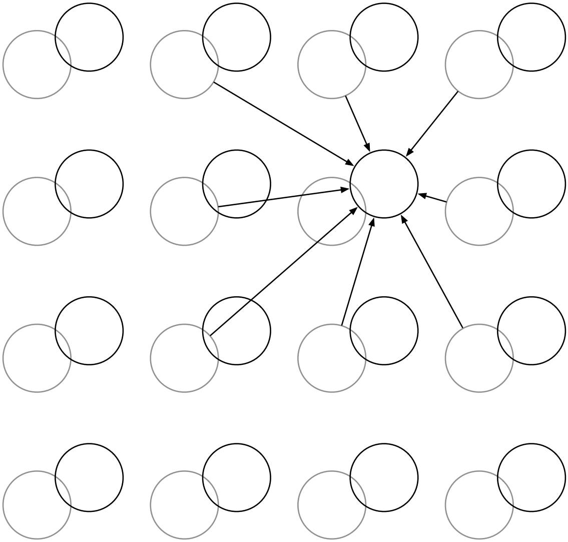
else

$newGrid[i,j] \leftarrow 0$

end

end

Game of Life Task Graph



Matrix-Vector Multiplication

foreach *row i of matrix A* **do**
  $b[i] \leftarrow 0$
  foreach *column j of A* **do**
    $b[i] \leftarrow b[i] + A[i,j] * x[j]$
  end
end

- series of inner products between b and a row of A (inner loop above).
  - suggests a decomposition of A into rows
  - each tasks computes inner product
- go further by decomposing inner product
  - decompose matrix into its elements

Bottom-up Dynamic Programming

- Dynamic programming algorithms formulated as recurrence relations, and can be solved recursively
- Usually solved bottom-up, where solution is built up from smaller to larger problems

Subset Sum Problem

Given a set of positive integers $\{s_1..s_n\}$, does there exist a subset whose sum is equal to a desired value S?

$$F[i,j] = \begin{cases} F[i-1,j] \vee F[i-1,j-s_i] & \text{if } i > 0\,; \\ 1 & \text{if } j = 0\,; \\ 0 & \text{otherwise.} \end{cases}$$

Example

$s = \{5, 3, 2, 8\}$ and $S = 7$

i \ j	0	1	2	3	4	5	6	7	
4	1	0	1	1	0	1	0	1	s_4=8
3	1	0	1	1	0	1	0	1	s_3=2
2	1	0	0	1	0	1	0	0	s_2=3
1	1	0	0	0	0	1	0	0	s_1=5

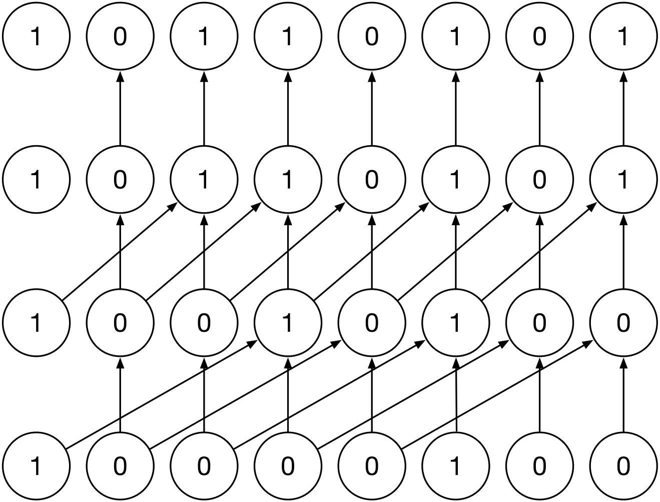
Input: Array $s[1..n]$ of n positive integers, target sum S
Output: returns 1 if a subset that sums to S exists, 0 otherwise
Data: Array $F[1..n, 0..S]$ initialized to 0

// subset sum always true for $j = 0$
for $i \leftarrow 1$ *to* n **do**
  $F[i, 0] \leftarrow 1$
end

$F[1, s[1]] \leftarrow 1$ // first integer summing to itself

for $i \leftarrow 2$ *to* n **do**
  for $j \leftarrow 1$ *to* S **do**
    $F[i, j] \leftarrow F[i - 1, j]$
    if $j \geq s[i]$ **then**
      $F[i, j] \leftarrow F[i, j] \vee F[i - 1, j - s[i]]$
    end
  end
end
return $F[n, S]$

Task Graph



List Ranking with Pointer Jumping