# 15-210
# Parallel and Sequential Algorithms and Data Structures

## Lecture 2

## Algorithmic Cost Models

# Synopsis

- Cost Models
- Parallelism
- Scheduling
- Cost Analysis for the Shortest Super String Problem
  - The Brute Force Algorithm
  - The Greedy Algorithm

# Cost Models

- Sequential: the Random Access Machine (RAM) model
- Parallel: the Parallel RAM model
- Parallel: the 15-210 model
  - Tied to high-level programming constructs – operational semantics
  - Think parallel!

# 15-210 Cost Model

- $W(e)$: Work needed to evaluate $e$
- $S(e)$: Span of the evaluation of $e$

- Parameterized with relevant problem size measures.

- Asymptotic Models
  - How do algorithms scale to large problems!

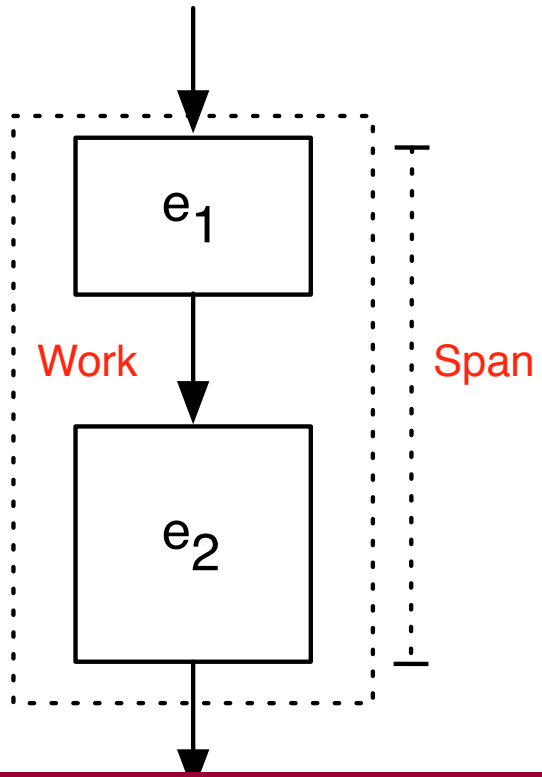
# Parameterization

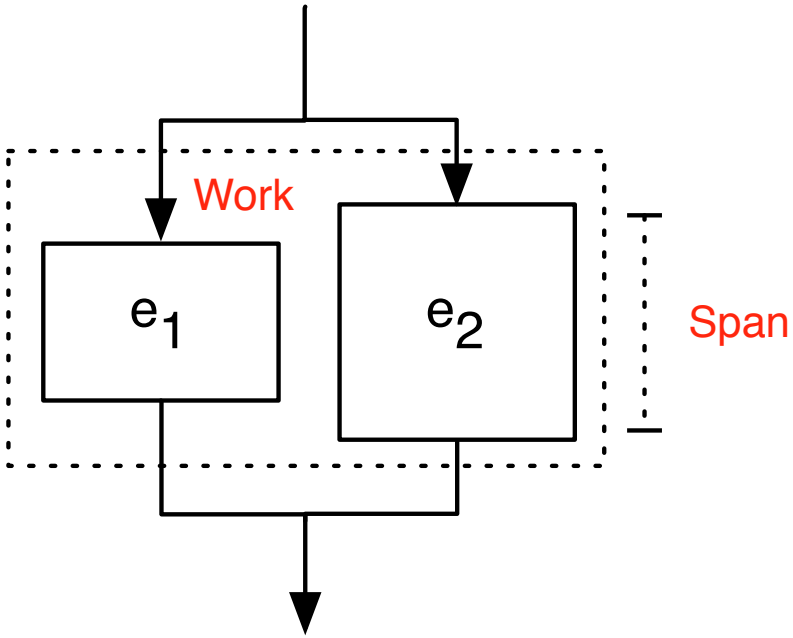
- We measure the size of **representation** of the input.

- Sorting: Number of items to sort
- Map, Reduce: Number of items in the sequence
- Graph Problems: Number of Nodes, Edges
- Searching: Number of items in the database
- Matrix operations: Number of rows and columns
- Prime number testing: Size – number of bits to represent the number (not the value!)
- Computing $n^{th}$ Fibonacci number: Size – number of bits to represent the number (not the value!)

# Rules of Composition

- $(e_1, e_2)$: Sequential Composition
  - Add work and span
- $e_1 || e_2$: Parallel Composition
  - Add work but take the maximum span



# Rules of Composition

| **e** | **W(e)** | **S(e)** |
|---|---|---|
| $c$ | $1$ | $1$ |
| op $e$ | $1$ | $1$ |
| $(e_1, e_2)$ | $1 + W(e_1) + W(e_2)$ | $1 + S(e_1) + S(e_2)$ |
| $(e_1 \| \| e_2)$ | $1 + W(e_1) + W(e_2)$ | $1 + \max(S(e_1), S(e_2))$ |
| `let val` $x = e_1$ `in` $e_2$ `end` | $1 + W(e_1) + W(e_2[\text{Eval}(e_1)/x])$ | $1 + S(e_1) + S(e_2[\text{Eval}(e_1)/x])$ |
| $\{f(x) \mid x \in A\}$ | $1 + \sum_{x \in A} W(f(x))$ | $1 + \max_{x \in A} S(f(x))$ |

# Rules of Composition

- $\{f(x) \mid x \in A\} \equiv \texttt{map}\ f\ A$
  - $W(\texttt{map}\ f\ \langle s_0, \ldots, s_{n-1} \rangle) = 1 + \sum_{i=0}^{n-1} W(f(s_i))$
  - $S(\texttt{map}\ f\ \langle s_0, \ldots, s_{n-1} \rangle) = 1 + \max_{i=0}^{n-1} S(f(s_i))$

# Upper and Lower Bounds

- Upper bound: The maximum asymptotic work (and span) that a given algorithm needs for all inputs of size $n$.

- Lower bound: The minimum asymptotic work (and span) that any algorithm for a problem needs for all inputs of size $n$.

# Synopsis

- Cost Models
- Parallelism
- Scheduling
- Cost Analysis for the Shortest Super String Problem
  - The Brute Force Algorithm
  - The Greedy Algorithm

# Parallelism

- For a given $W$ and $S$, what is the maximum number of processors you can utilize?
  - $\mathbb{P} = \frac{W}{S}$
- Why?
- Mergesort has $W = \theta(n \log n)$ and $S = \theta(\log^2 n)$
- $\mathbb{P} = \theta(\frac{n}{\log n})$
  - The larger the problem is, the higher the parallelism

# Designing Parallel Algorithms

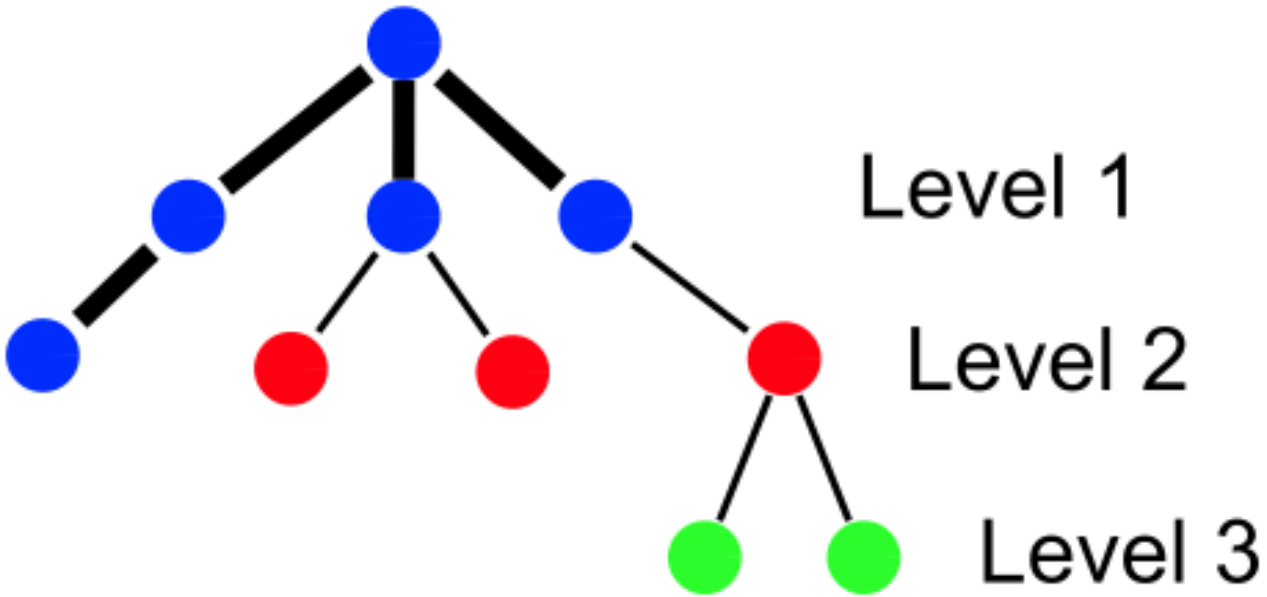
- Keep work as low as possible
  - No unnecessary computation
- Keep span as low as possible
  - Hence get high-parallelism

# Synopsis

- Cost Models
- Parallelism
- Scheduling
- Cost Analysis for the Shortest Super String Problem
  - The Brute Force Algorithm
  - The Greedy Algorithm

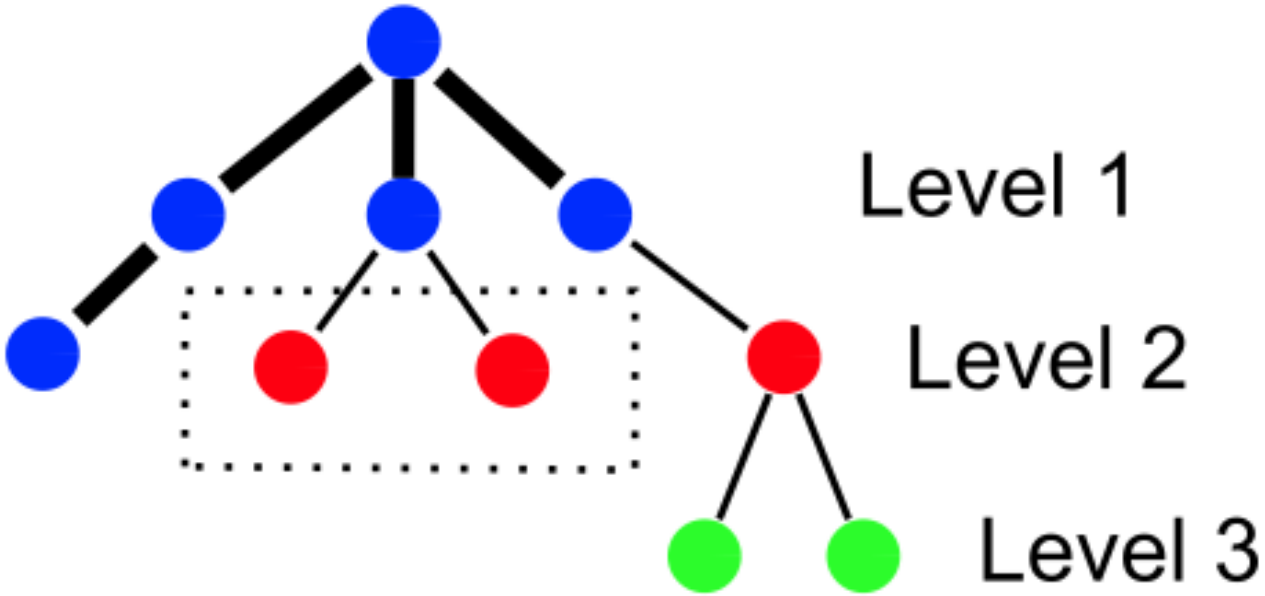
# Under the Hood: Task Scheduling

- Mapping from a computation graph to processors



# Greedy Scheduling

- A greedy scheduler will schedule a ready task on an available processor.



# A Lower Bound

- Let $T_p$ be the "time" needed when using $p$ processors,

$$\max(\frac{W}{p}, S) \leq T_p$$

- Why?

# An Upper Bound

- With $p$ processors

$$T_p < \frac{W}{p} + S$$

- Why?

# Tying things together

- Speed-up is $\frac{W}{T_p}$
  - Maximum possible speed-up is $p$.

$$\begin{aligned} T_p &< \frac{W}{p} + S \\ &= \frac{W}{p} + \frac{W}{\mathbb{P}} \\ &= \frac{W}{p}\left(1 + \frac{p}{\mathbb{P}}\right) \end{aligned}$$

- $\mathbb{P} \gg p \rightarrow$ near perfect parallelism

# Synopsis

- Cost Models
- Parallelism
- Scheduling
- Cost Analysis for the Shortest Super String Problem
  - The Brute Force Algorithm
  - The Greedy Algorithm

# Costs for the Brute Force SS Algorithm

- The brute-force algorithm
  - For each permutation
    - Remove overlaps
    - Stitch strings
  - Output (one of) the shortest string(s)

- $\text{overlap}(s_i, s_j)$ will be needed many times.
  - Preprocess $S$ once and store overlaps as a table
    - What prefix to remove
    - Increase in length

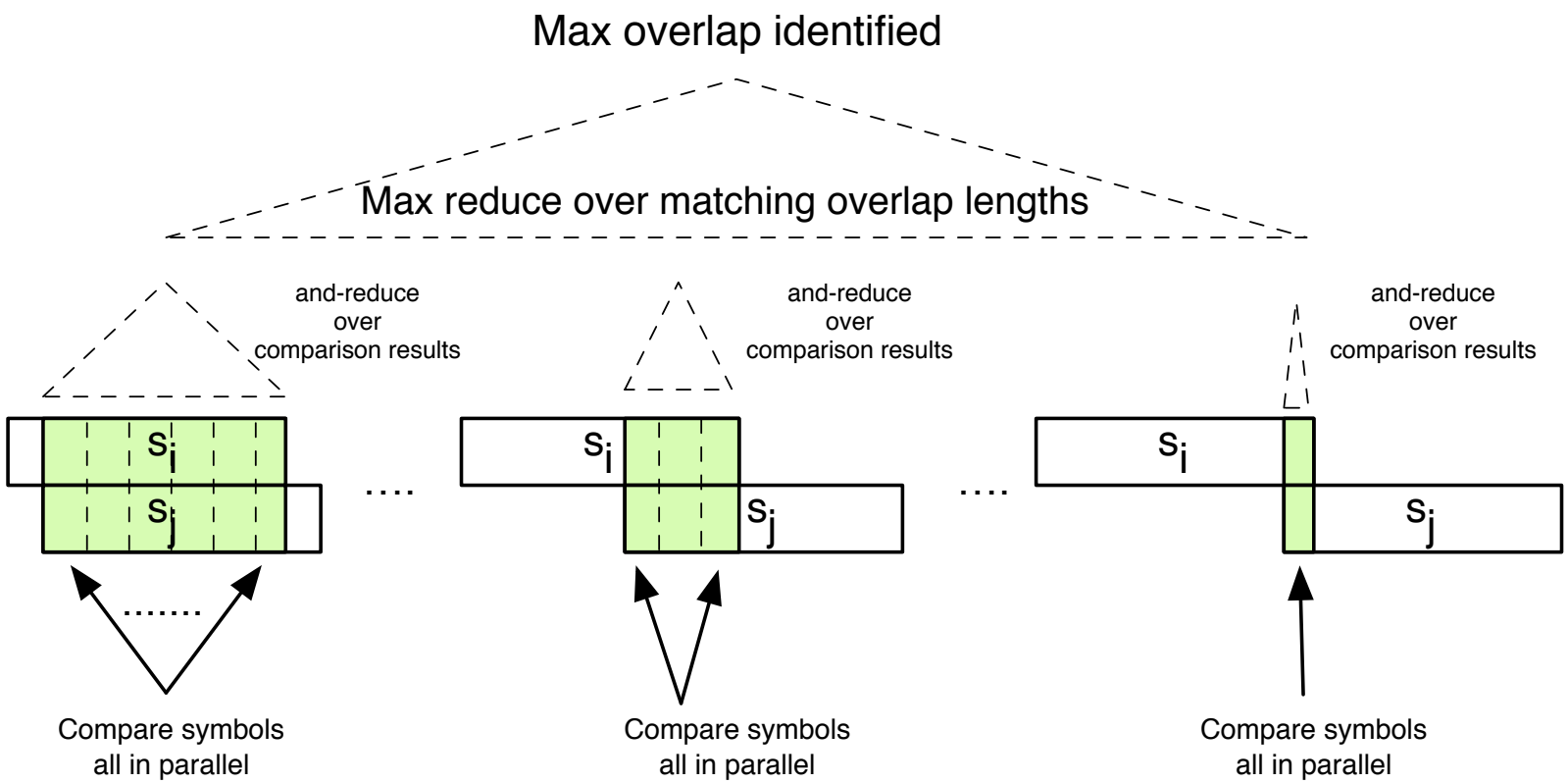
# Preprocessing – Inputs

- A set $S$ is $n$ strings, $s_1, s_2, \cdots, s_n$
- Define

$$m = \sum_{i=1}^{n} |s_i|$$

and observe $n \leq m$.

# Preprocessing a pair




- Work and span for preprocessing one pair, $s_i$ and $s_j$?
  - $W = O(|s_i| \cdot |s_j|)$ Why?
  - $S = O(\log(|s_i| + |s_j|))$ Why?

# Preprocessing – Work

$$
\begin{aligned}
W_{ov} &\leq \sum_{i=1}^{n}\sum_{j=1}^{n} W(\texttt{overlap}(s_i, s_j)) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} O(|s_i||s_j|) \\
&\leq \sum_{i=1}^{n}\sum_{j=1}^{n} (k_1 + k_2|s_i||s_j|) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} k_1 + \sum_{i=1}^{n}\sum_{j=1}^{n} (k_2|s_i||s_j|) \\
&= k_1 n^2 + k_2 \sum_{j=1}^{n} |s_j| (\sum_{i=1}^{n} |s_i|) = k_1 n^2 + k_2 m^2 \in \mathbf{O(m^2)}
\end{aligned}
$$

# Preprocessing – Span

- All $s_i$, $s_j$ pairs can be processed in parallel.

$$\begin{aligned} S_{ov} &\leq \max_{i=1}^{n} \max_{j=1}^{n} S(\texttt{overlap}(s_i, s_j))) \\ &\in \mathbf{O}(\log \mathbf{m}) \end{aligned}$$

# Brute Force SS Algorithm

- Work:
  - $O(n)$ lookups each with $O(1)$ work. Why?
  - $n!$ permutations
  - $O(n \cdot n!) = \mathbf{O((n+1)!)}$
  - $W_{ov}$ can be ignored!
- Span:
  - All permutations can be done in parallel, but!

```
func permutations S =
    if |S| = 1 then {S}
    else
      {append([s], p) :
          s in S, p in permutations(S\s)}
```

  - This has span $O(n)$. Why?
  - $S_{ov}$ can be ignored.

# Synopsis

- Cost Models
- Parallelism
- Scheduling
- Cost Analysis for the Shortest Super String Problem
  - The Brute Force Algorithm
  - The Greedy Algorithm

# The Greedy SS Algorithm

```
1   fun greedyApproxSS(S) =
2     if |S| = 1 then s_0
3     else let
4         O = {(overlap(s_i, s_j), s_i, s_j) : s_i ∈ S, s_j ∈ S, s_i ≠ s_j}
5         (o, s_i, s_j) = maxval <_#1 O
6         s_k = join(s_i, s_j)
7         S' = ({s_k} ∪ S)\{s_i, s_j}
8     in
9       greedyApproxSS(S')
10    end
```

# The Greedy SS Algorithm

```
1   fun greedyApproxSS(S) =
2     if |S| = 1 then s_0
3     else let
4         O = {(overlap(s_i, s_j), s_i, s_j) : s_i ∈ S, s_j ∈ S, s_i ≠ s_j}
5         (o, s_i, s_j) = maxval <_#1 O
6         s_k = join(s_i, s_j)
7         S' = ({s_k} ∪ S)\{s_i, s_j}
8     in
9       greedyApproxSS(S')
10    end
```

- $W_{ov} = O(m^2),\ S_{ov} = O(\log m)$

# The Greedy SS Algorithm

```
1   fun greedyApproxSS(S) =
2     if |S| = 1 then s_0
3     else let
4         O = {(overlap(s_i, s_j), s_i, s_j) : s_i ∈ S, s_j ∈ S, s_i ≠ s_j}
5         (o, s_i, s_j) = maxval <_#1 O
6         s_k = join(s_i, s_j)
7         S' = ({s_k} ∪ S)\{s_i, s_j}
8     in
9       greedyApproxSS(S')
10    end
```

- $W_{maxval} = O(m^2),\ S_{maxval} = O(\log m)$
- Why?

# The Greedy SS Algorithm

```
1   fun greedyApproxSS(S) =
2     if |S| = 1 then s_0
3     else let
4         O = {(overlap(s_i, s_j), s_i, s_j) : s_i ∈ S, s_j ∈ S, s_i ≠ s_j}
5         (o, s_i, s_j) = maxval <_#1 O
6         s_k = join(s_i, s_j)
7         S' = ({s_k} ∪ S)\{s_i, s_j}
8     in
9       greedyApproxSS(S')
10    end
```

- No more than $W = O(m^2),\ S = O(\log m)$
- Why?

# The Greedy SS Algorithm

```
1  fun greedyApproxSS(S) =
2    if |S| = 1 then s_0
3    else let
4      O = {(overlap(s_i, s_j), s_i, s_j) : s_i ∈ S, s_j ∈ S, s_i ≠ s_j}
5      (o, s_i, s_j) = maxval <_#1 O
6      s_k = join(s_i, s_j)
7      S' = ({s_k} ∪ S)\{s_i, s_j}
8    in
9      greedyApproxSS(S')
10   end
```

- At most $n$ (sequential) calls to `greedyApproxSS`
  - Each with $W = O(m^2),\ S = O(\log m)$
- $W_{greedy} = O(nm^2)$ and $S_{greedy} = O(n \log m)$
- Why?

# Summary

- Cost Models: Rules of Composition
- Parallelism and Scheduling
- Cost Analysis for the Shortest Super String Problem
  - Preprocessing for overlaps
  - The Brute Force Algorithm
  - The Greedy Algorithm