PARALLEL PROCESSING

Mohammed Alabdulkareem

kareem@ksu.edu.sa

Office 2247

6

PARALLEL ALGORITHM COMPLEXITY

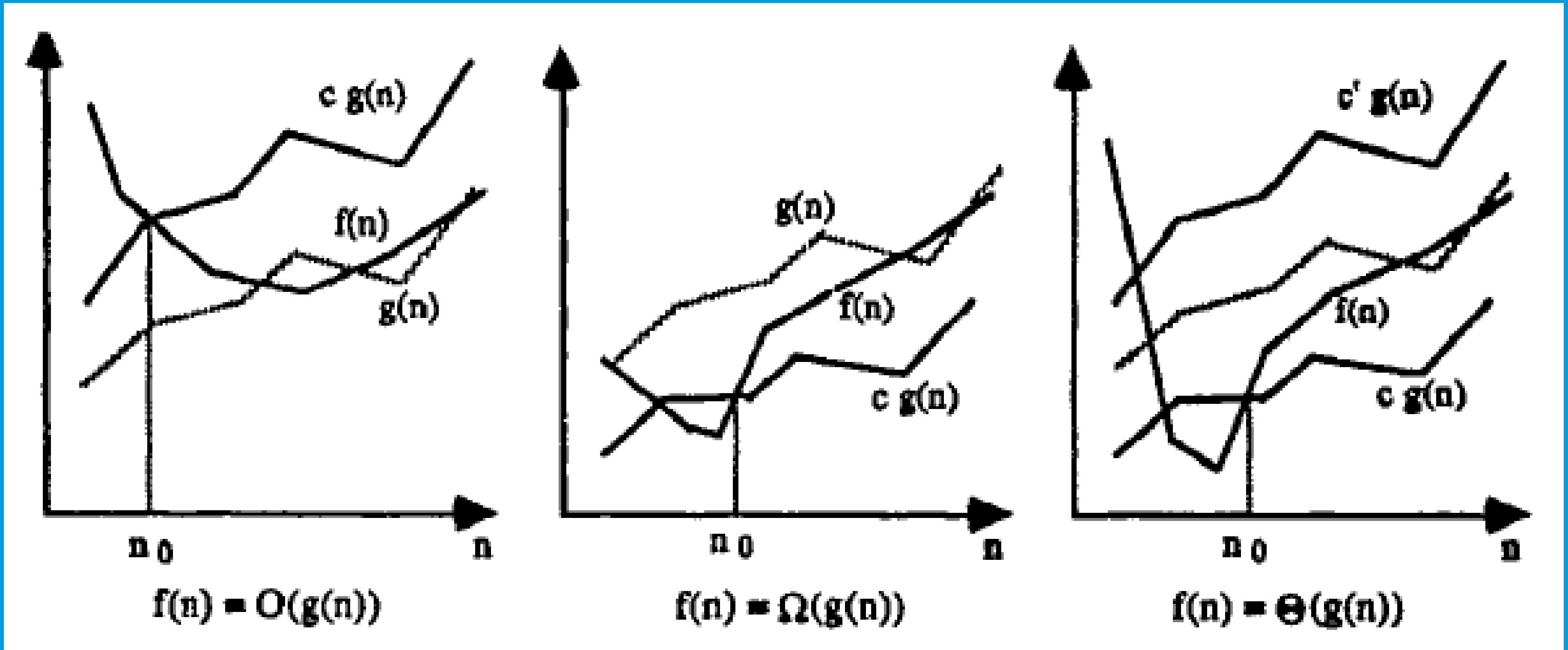
- For large problem size *n* we use asymptotic complexity to analyze and compare parallel algorithms.
- An algorithm of complexity (log2 *n*) ² and another algorithm of complexity 500 log2 *n* the later is taking on the order of log2 *n.*

$f(n) = O(g(n))$ if $\exists c, n_0$ such that $\forall n > n_0$ we have $f(n) < c\, g(n)$

$f(n) = \Omega(g(n))$ if $\exists c, n_0$ such that $\forall n > n_0$ we have $f(n) > c\, g(n)$

$f(n) = \Theta(g(n))$ if $\exists c, c', n_0$ such that $\forall n > n_0$ we have $c\, g(n) < f(n) < c'\, g(n)$

PARALLEL ALGORITHM COMPLEXITY



PARALLEL ALGORITHM COMPLEXITY

- Some methods used in devising parallel algorithms
  - Divide and Conquer.
  - Randomization.
  - Approximation

PARALLEL ALGORITHM COMPLEXITY

- **Divide and Conquer.**
  - Some problems in P can be solved in parallel by decomposing the problem of size *n* into two or more "smaller" sub-problems.
  - For a problem of size *n* ,
    - $Td(n)$ is decomposing time.
    - Ts is the time to solve sub-problem.
    - $Tc(n)$ *the time to combine.*
    - *The total time* $T(n) = Td(n) + Ts + Tc(n)$

PARALLEL ALGORITHM COMPLEXITY

- **Divide and Conquer example**
  - sorting a list of *n* keys, we can decompose the list into two halves each of $^{n}/_{2}$ elements.
  - Time to sort is $\mathrm{T}\left(\frac{n}{2}\right)$ each sub list will be sorted in the same manner (decomposed, sorted, and merged).
  - Time to decompose and combine is 2 log(n)
  - $T(n) = T(n/2) + 2 \log n$.

PARALLEL ALGORITHM COMPLEXITY

- **Randomization**
  - Sometimes it is impossible, or computationally difficult, to decompose a large problem into smaller problems in such a way that the solution times of the sub-problems are roughly equal.
  - Large decomposition and combining overheads, or wide variations in the solution times of the sub-problems, <u>may reduce the effective speed-up</u> achievable by the divide-and-conquer method.
  - it might be possible to use random decisions that lead to good results with very high probability.

PARALLEL ALGORITHM COMPLEXITY

- Randomization
  - Sorting Algorithm Example:
    - Suppose that each of p processors begins with a sub-list of size n/p.
    - First each processor selects a random sample of size k from its local sub-list.
    - The kp samples from all processors form a smaller list that can be readily sorted.
    - This sorted list of samples is now divided into p equal segments.
    - The beginning values in the p segments used as thresholds to divide the original list of n keys into p sub-lists.
    - The lengths of these latter sub-lists will be approximately balanced.
    - The sub-lists are sorted within each processor.
    - Depending on the threshold permutation between processors.
    - Local sorting at each processor needed to sort the n keys.

PARALLEL ALGORITHM COMPLEXITY

- **Randomization Practices**
- Other useful practices of randomization:
  - **Random Search:** When searching large space for an element that exist, random search can <u>lead to very good average-case</u> performance.
  - **Control randomization:** To <u>avoid consistently experiencing close to worst-case</u> because unfortunate distribution of inputs, the inputs can be chosen at random.
  - **Symmetry breaking:** Interacting deterministic processes may exhibit a <u>cyclic behavior that leads to deadlock</u>. Randomization can be used to break the symmetry and thus the deadlock.

PARALLEL ALGORITHM COMPLEXITY

- **Approximation**
  - Iterative numerical methods often use approximation to arrive at the solution(s).
  - For example, to solve a system of *n* linear equations, one can begin with some rough estimates for the answers and then successively refine these estimates using parallel numerical calculations.
  - Jacobi relaxation is an example. The analysis of complexity is somewhat more difficult here as the number of iterations required often cannot be expressed as a simple function of the desired accuracy and/or the problem size.

PARALLEL ALGORITHM COMPLEXITY

- **Approximation example**

- Suppose we are given the following linear system:

$$
\begin{aligned}
10x_1 - x_2 + 2x_3 &= 6, \\
-x_1 + 11x_2 - x_3 + 3x_4 &= 25, \\
2x_1 - x_2 + 10x_3 - x_4 &= -11, \\
3x_2 - x_3 + 8x_4 &= 15.
\end{aligned}
$$

PARALLEL ALGORITHM COMPLEXITY

- **Approximation example**

If we choose (0, 0, 0, 0) as the initial approximation, then the first approximate solution is:

$$
\begin{aligned}
x_1 &= (6 + 0 - (2 * 0))/10 = 0.6, \\
x_2 &= (25 + 0 + 0 - (3 * 0))/11 = 25/11 = 2.2727, \\
x_3 &= (-11 - (2 * 0) + 0 + 0)/10 = -1.1, \\
x_4 &= (15 - (3 * 0) + 0)/8 = 1.875.
\end{aligned}
$$

PARALLEL ALGORITHM COMPLEXITY

- **Approximation example**

Using the approximations obtained, the iterative procedure is repeated until the desired accuracy has been reached.

x_1	x_2	x_3	x_4
0.6	2.27272	-1.1	1.875
1.04727	1.7159	-0.80522	0.88522
0.93263	2.05330	-1.0493	1.13088
1.01519	1.95369	-0.9681	0.97384
0.98899	2.0114	-1.0102	1.02135

The exact solution is (1,2,-1,1)

PARALLEL ALGORITHM COMPLEXITY

- **SOLVING RECURRENCES**
  - *We may solve recurrence by unrolling:*

$$
\begin{aligned}
f(n) &= f(n-1) + n \qquad \{\text{rewrite } f(n-1) \text{ as } f((n-1)-1) + n - 1\} \\
&= f(n-2) + n - 1 + n \\
&= f(n-3) + n - 2 + n - 1 + n \\
&\;\;\cdots \\
&= f(1) + 2 + 3 + \cdots + n - 1 + n \\
&= n(n+1)/2 - 1 \\
&= \Theta(n^2)
\end{aligned}
$$