CS4379: Parallel and Concurrent Programming
CS5379: Parallel Processing

Lecture 12

Dr. Yong Chen

Associate Professor

Computer Science Department

Texas Tech University

Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA:** Mr. Ghazanfar Ali, Ghazanfar.Ali@ttu.edu
- **TA Office hours:** Tue. and Fri., 2-3 p.m., or by appointment
- **TA Office:** EC 201 A
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

Outline

- Questions?

- Gaussian Elimination algorithm of solving linear equations
- Sorting, quicksort, and parallelizing quicksort
- Search, depth-first search (DFS), and parallelizing DFS
- Quiz #3
  - Online test preferred (requires LockDown Browser), paper copies available too

Solving a System of Linear Equations

- Consider the problem of solving linear equations of the kind:

$$
\begin{array}{ccccccc}
a_{0,0}x_0 & + & a_{0,1}x_1 & + & \cdots & + & a_{0,n-1}x_{n-1} = b_0, \\
a_{1,0}x_0 & + & a_{1,1}x_1 & + & \cdots & + & a_{1,n-1}x_{n-1} = b_1, \\
\vdots & & \vdots & & & & \vdots \\
a_{n-1,0}x_0 & + & a_{n-1,1}x_1 & + & \cdots & + & a_{n-1,n-1}x_{n-1} = b_{n-1}.
\end{array}
$$

- This is written as $Ax = b$, where A is an n x n matrix with $A[i, j] = a_{i,j}$, b is an n x 1 vector $[\, b_0, b_1, \ldots, b_n \,]^T$, and x is the solution.

- Widely used in machine learning and data analysis libraries

Solving a System of Linear Equations

Two steps in solution are: reduction to triangular form, and back-substitution. The triangular form is as:

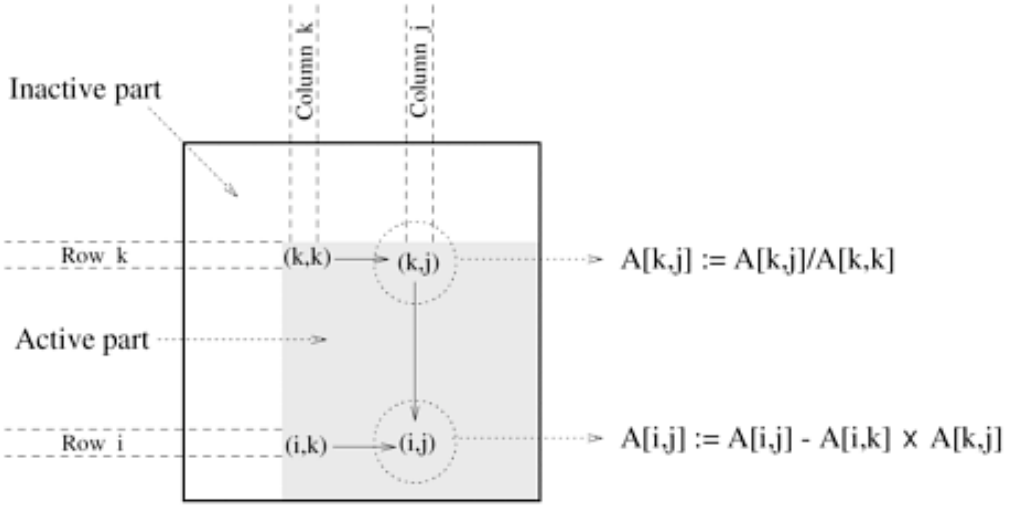
$$\begin{aligned} x_0 + u_{0,1}x_1 + u_{0,2}x_2 + \cdots + u_{0,n-1}x_{n-1} &= y_0, \\ x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} &= y_1, \\ &\vdots \qquad\qquad\qquad\quad \vdots \\ x_{n-1} &= y_{n-1}. \end{aligned}$$

We write this as: $Ux = y$.

A commonly used method for transforming a given matrix into an upper-triangular matrix is Gaussian Elimination.

Gaussian Elimination

- The computation has three nested loops - in the kth iteration of the outer loop, the algorithm performs $(n-k)^2$ computations. Summing from $k = 1..n$, we have roughly $(n^3/3)$ multiplications-subtractions.



A typical computation in Gaussian elimination.

Gaussian Elimination

```
1.    procedure GAUSSIAN_ELIMINATION (A, b, y)
2.    begin
3.       for k := 0 to n − 1 do                    /* Outer loop */
4.       begin
5.          for j := k + 1 to n − 1 do
6.             A[k, j] := A[k, j]/A[k, k];        /* Division step */
7.          y[k] := b[k]/A[k, k];
8.          A[k, k] := 1;
9.          for i := k + 1 to n − 1 do
10.         begin
11.            for j := k + 1 to n − 1 do
12.               A[i, j] := A[i, j] − A[i, k] × A[k, j]; /* Elimination step */
13.            b[i] := b[i] − A[i, k] × y[k];
14.            A[i, k] := 0;
15.         endfor;                               /* Line 9 */
16.      endfor;                                  /* Line 3 */
17.   end GAUSSIAN_ELIMINATION
```

Serial Gaussian Elimination

Parallel Gaussian Elimination

- Assuming distributed-address-space

- Assume $p = n$ with each row assigned to a processor.
- Step 1: The first step of the algorithm normalizes the row. This is a serial operation
- Step 2: In the second step, the normalized row is sent to all the processors
- Step 3: Each processor can independently eliminate this row from its own
- Step 4: Repeat the above steps n times; The (k+1)st iteration starts only after all the computation and communication for the kth iteration is complete (cannot parallelize the outer loop directly)

Parallel Gaussian Elimination

- When p < n, can have
  - Cyclic distribution
  - Block distribution

- Not optimal, can be improved with pipeline
- In the pipelined version, there are three steps
  - Normalization of a row
  - Communication
  - Elimination
- These steps are performed in an asynchronous fashion.

Outline

- Questions?
- Gaussian Elimination algorithm of solving linear equations
- Sorting, quicksort, and parallelizing quicksort
- Search, depth-first search (DFS), and parallelizing DFS
- Quiz #3

Sorting

- One of the most commonly used and well-studied kernels.
- Sorting can be comparison-based or noncomparison-based.
- The fundamental operation of comparison-based sorting is compare-exchange.
- The lower bound on any comparison-based sort of *n* numbers is Θ(nlog n).

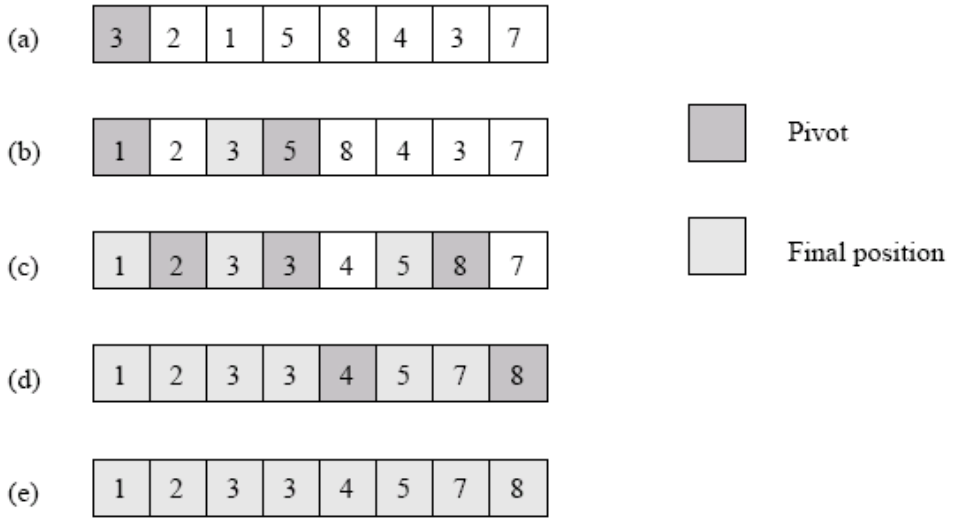
Quicksort

- Quicksort is one of the most common sorting algorithms for sequential computers because of its simplicity, low overhead, and optimal average complexity.
- Quicksort selects one of the entries in the sequence to be the pivot and divides the sequence into two - one with all elements less than the pivot and other greater.
- The process is recursively applied to each of the sublists.

Quicksort

The sequential quicksort algorithm.

```
1.   procedure QUICKSORT (A, q, r)
2.   begin
3.       if q < r then
4.       begin
5.           x := A[q];
6.           s := q;
7.           for i := q + 1 to r do
8.               if A[i] ≤ x then
9.               begin
10.                  s := s + 1;
11.                  swap(A[s], A[i]);
12.              end if
13.          swap(A[q], A[s]);
14.          QUICKSORT (A, q, s);
15.          QUICKSORT (A, s + 1, r);
16.      end if
17.  end QUICKSORT
```



Example of the quicksort algorithm sorting a sequence of size *n* = 8.

Quicksort

- The performance of quicksort depends critically on the quality of the pivot.
- In the best case, the pivot divides the list in such a way that the larger of the two lists does not have more than αn elements (for some constant α).
- In this case, the complexity of quicksort is $O(n \log n)$.

Parallelizing Quicksort

- Lets start with recursive decomposition - the list is partitioned serially and each of the subproblems is handled by a different processor.

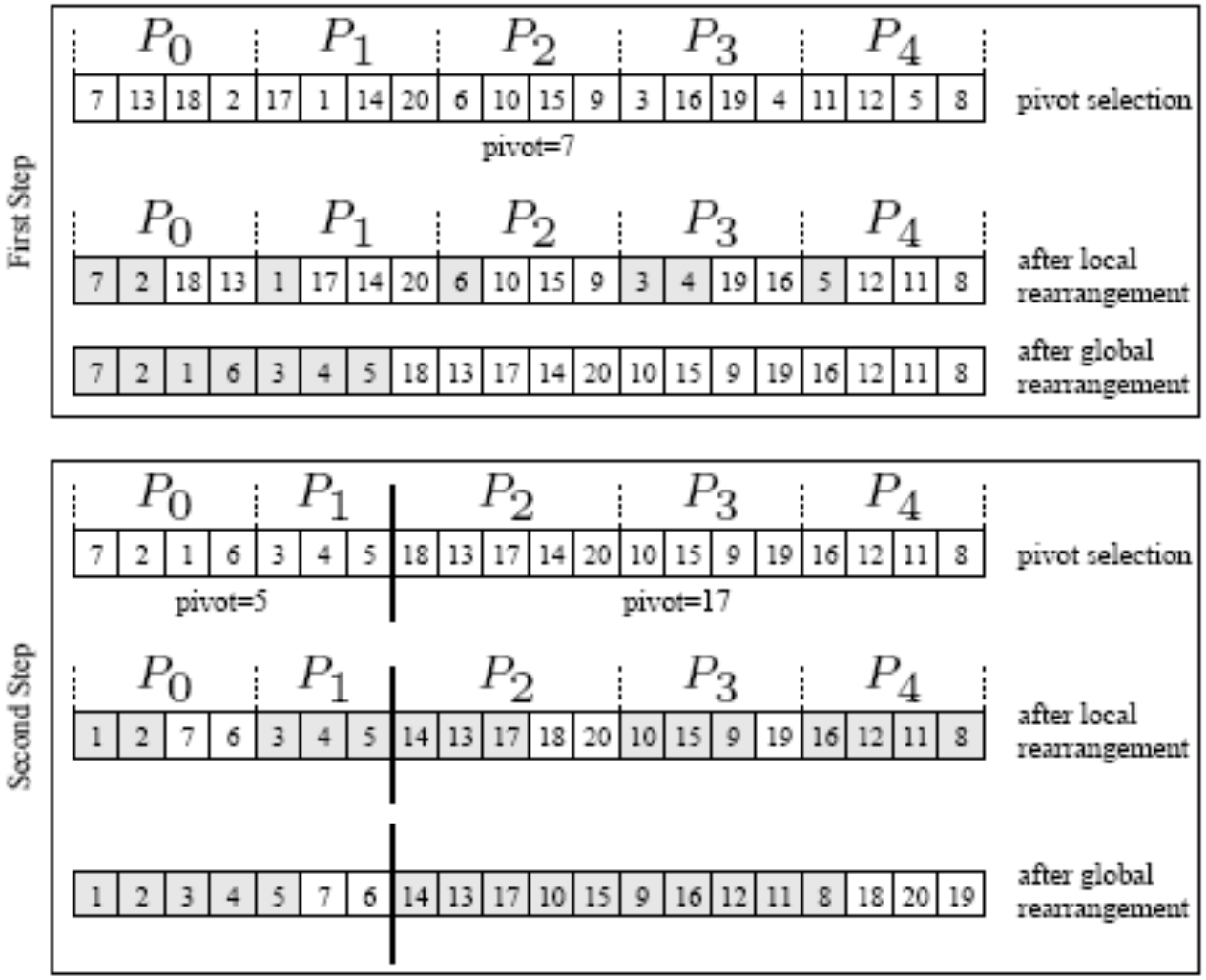
- The time for this algorithm is lower-bounded by $\Omega(n)$ as the list is partitioned serially regardless how many processors used!

- We need to parallelize the partitioning step too – i.e. use p processors to partition the list concurrently – not easy though

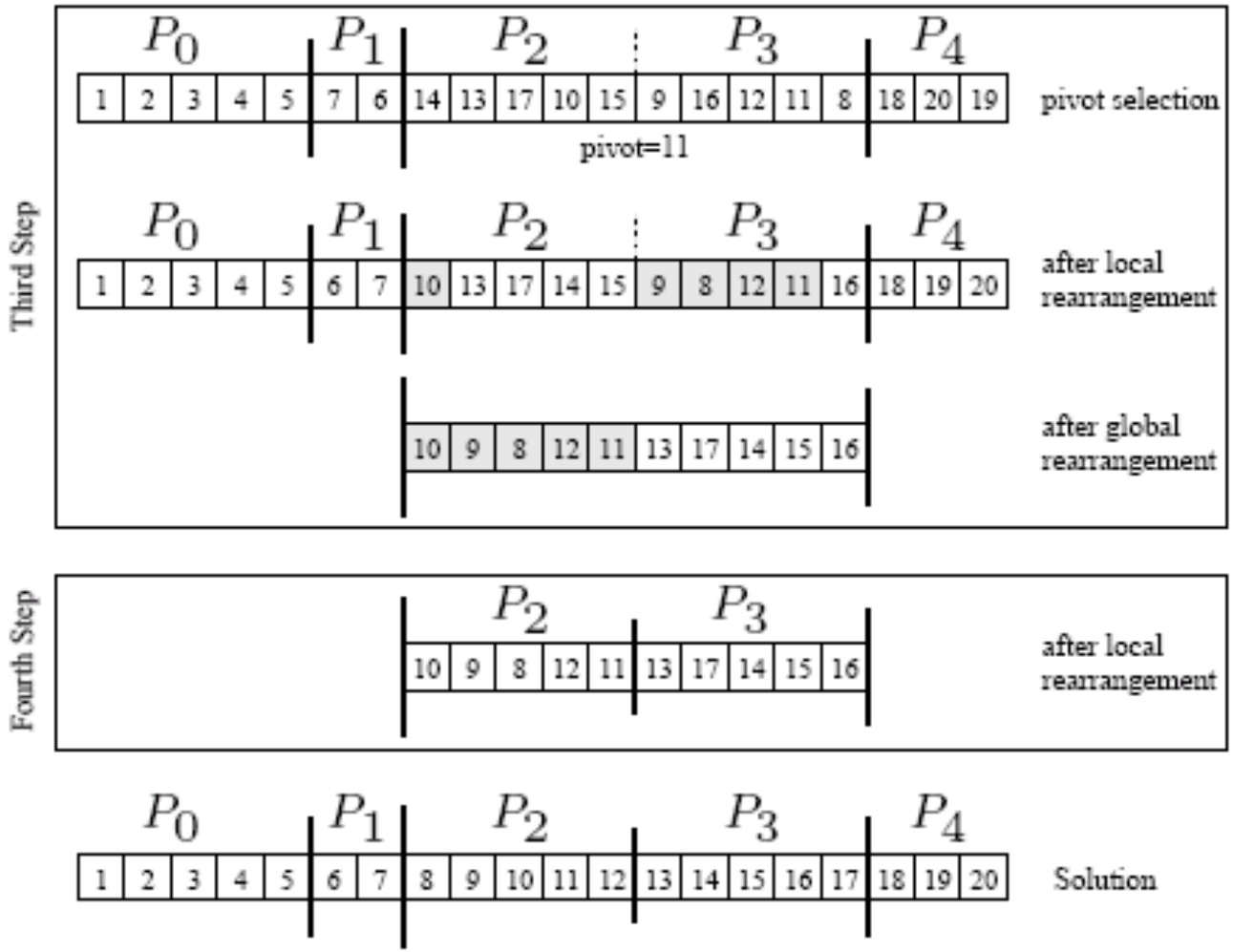
Parallelizing Quicksort: Shared Address Space Formulation

- Consider a list of size n equally divided across p processors.
- A pivot is selected by one of the processors and made known to all processors.
- Each processor partitions its list into two, say L_i and U_i, based on the selected pivot.
- All of the L_i lists are merged and all of the U_i lists are merged separately.
- The set of processors is partitioned into two (in proportion of the size of lists L and U). The process is recursively applied to each of the lists.

Shared Address Space Formulation



Shared Address Space Formulation

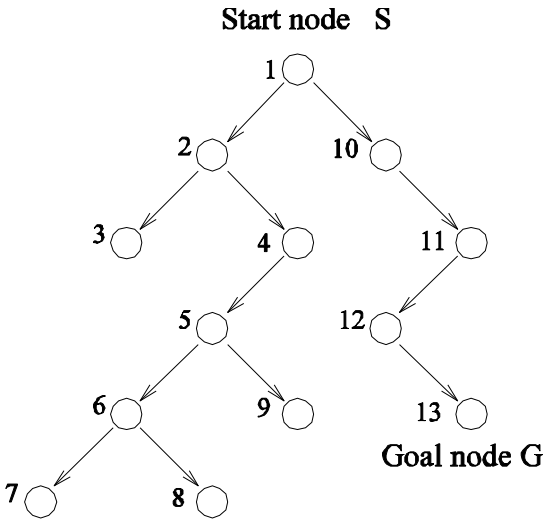


Parallelizing Quicksort: Distributed Address Space

- Similarly, but arrays are explicitly distributed among processes
  - Not stored in shared memory and cannot be accessed by all processes
- A designated processor selects and broadcasts the pivot.
- Each processor splits its local list into two lists, one less (L_i), and other greater (U_i) than the pivot.
- A processor in the low half of the machine sends its list U_i to the paired processor in the other half. The paired processor sends its list L_i.
- It is easy to see that after this step, all elements less than the pivot are in the low half of the machine and all elements greater than the pivot are in the high half.
- The above process is recursively conducted until each processor has its own local list, which is sorted locally.

Depth-First Search Algorithm

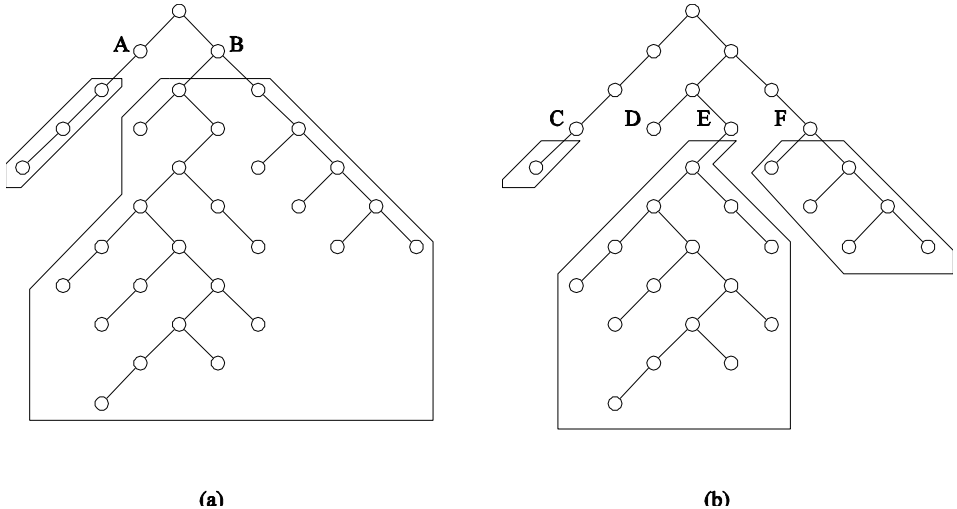
- Applies to search spaces that are trees.
- DFS begins by expanding the initial node and generating its successors. In each subsequent step, DFS expands one of the most recently generated nodes.
- If there exists no success, DFS backtracks to the parent and explores an alternate child.
- The main advantage of DFS is that its storage requirement is linear in the depth of the state space being searched.



Parallel Depth-First Search

- How is the search space partitioned across processors?
- Different subtrees can be searched concurrently.
- However, subtrees can be very different in size.
- It is difficult to estimate the size of a subtree rooted at a node.
- Dynamic load balancing is required.

Parallel Depth-First Search

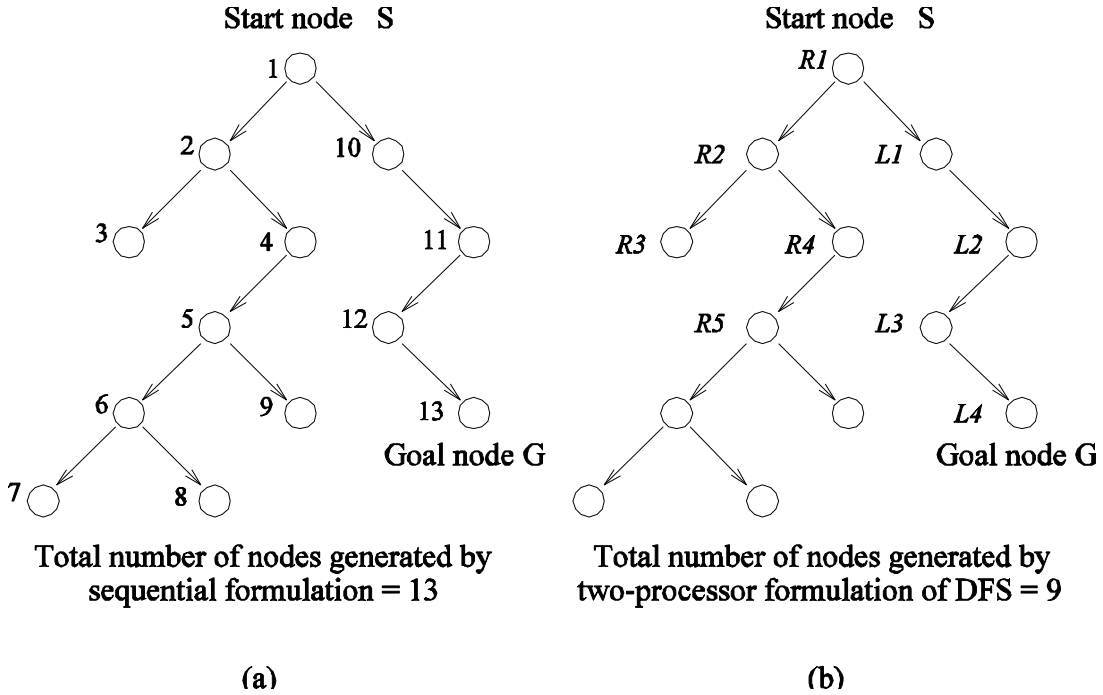


The unstructured nature of tree search and the imbalance resulting from static partitioning.

Parallel Depth-First Search: Dynamic Load Balancing

- When a processor runs out of work, it gets more work from another processor.
- This is done using work requests and responses in distributed-address-space machines and locking and extracting work in shared-address space machines.
- On reaching final state at a processor, all processors terminate.

Super-linear Speedup in Parallel Search



The difference in number of nodes searched by sequential and parallel formulations of DFS. For this example, parallel DFS reaches a goal node after searching fewer nodes than sequential DFS.

Quiz #3

- Password for online test (must use Lockdown Browser): ttudiscl20
- Note that only one attempt is allowed

Supplemental Readings

- Reference book ITPC, Chapters 8.3, 9.4, 11.4-11.6
- Foster, DBPP, Chapter 2
  http://www.mcs.anl.gov/~itf/dbpp

Questions?

Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].