Introduction to Parallel Computing

George Karypis

Some Serial Algorithms

Some Serial Algorithms

Working Examples

- Dense matrix algorithms
  - Matrix-vector and matrix-matrix multiplication
  - Gaussian elimination
- Sparse matrix algorithms
  - Matrix-vector multiplication
- Finding minimum and maximum elements in an array
- Graph algorithms
  - Depth-first and breadth-first traversals.
  - Maximal independent sets.
  - Minimum spanning tree (Dijkstra and Kruskal's algorithms)
  - Single source shortest path (Dijksta's & Bellman-Ford algorithms)
  - All-pairs shortest path algorithms (Floyds-Warshall)
- Sorting
  - Quicksort, radix sort, bucket sort, counting sort, sample sort.
- Search algorithms
  - Best-first and depth-first search
  - A* and IDA* heuristic search
- Discrete event simulation
  - Conservative and time-warp approaches
- Longest common subsequence
- Optimal matrix parenthesization
- 0/1 Knapsack problem

Dense Matrix-Vector Multiplication

```
procedure MAT_VECT (A, x, y)
begin
    for i := 0 to n − 1 do
    begin
        y[i] := 0;
        for j := 0 to n − 1 do
            y[i] := y[i] + A[i, j] × x[j];
    endfor;
end MAT_VECT
```

Algorithm 8.1 A serial algorithm for multiplying an $n \times n$ matrix A with an $n \times 1$ vector x to yield an $n \times 1$ product vector y.

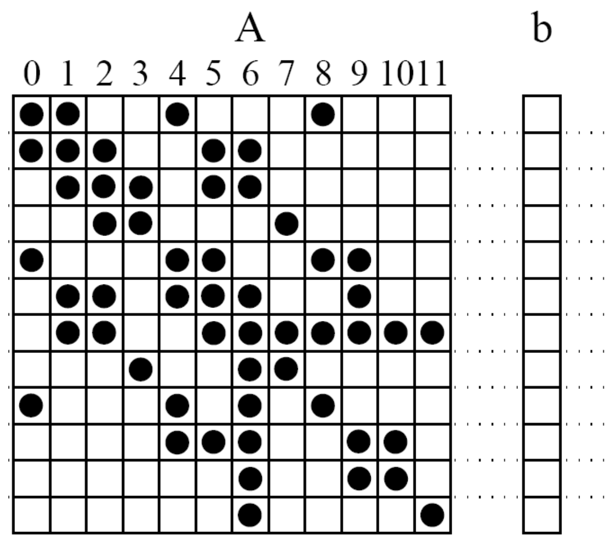
Dense Matrix-Matrix Multiplication

```
procedure MAT_MULT (A, B, C)
begin
    for i := 0 to n - 1 do
        for j := 0 to n - 1 do
            begin
                C[i, j] := 0;
                for k := 0 to n - 1 do
                    C[i, j] := C[i, j] + A[i, k] × B[k, j];
            endfor;
end MAT_MULT
```

Algorithm 8.2 The conventional serial algorithm for multiplication of two $n \times n$ matrices.

Sparse Matrix-Vector Multiplication

$$y = Ab$$



$$y[i] = \sum_{j=1}^{n} (A[i, j] \times b[j])$$

Gaussian Elimination

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

```
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
```

Algorithm 8.4 A serial Gaussian elimination algorithm that converts the system of linear equations $Ax = b$ to a unit upper-triangular system $Ux = y$. The matrix U occupies the upper-triangular locations of A. This algorithm assumes that $A[k, k] \neq 0$ when it is used as a divisor on lines 6 and 7.

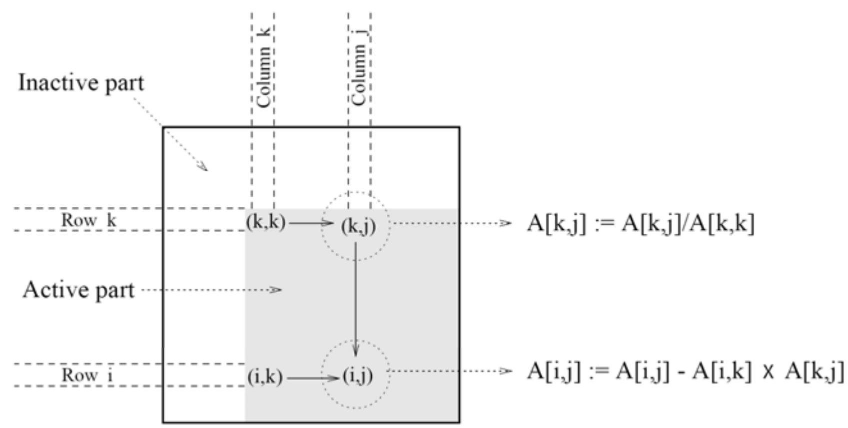


Figure 3.28 A typical computation in Gaussian elimination and the active part of the coefficient matrix during the kth iteration of the outer loop.

Floyd's All-Pairs Shortest Path

$$d_{i,j}^{(k)} = \begin{cases} w(v_i, v_j) & \text{if} \quad k = 0 \\ \min\left\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right\} & \text{if} \quad k \geq 1 \end{cases}$$

1. **procedure** FLOYD_ALL_PAIRS_SP(A)
2. **begin**
3. $\quad D^{(0)} = A;$
4. $\quad$ **for** $k := 1$ **to** n **do**
5. $\qquad$ **for** $i := 1$ **to** n **do**
6. $\qquad\quad$ **for** $j := 1$ **to** n **do**
7. $\qquad\qquad d_{i,j}^{(k)} := \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right);$
8. **end** FLOYD_ALL_PAIRS_SP

Algorithm 10.3 Floyd's all-pairs shortest paths algorithm. This program computes the all-pairs shortest paths of the graph $G = (V, E)$ with adjacency matrix A.

Quicksort

```
procedure QUICKSORT (A, q, r)
begin
    if q < r then
    begin
        x := A[q];
        s := q;
        for i := q + 1 to r do
            if A[i] ≤ x then
            begin
                s := s + 1;
                swap(A[s], A[i]);
            end if
        swap(A[q], A[s]);
        QUICKSORT (A, q, s);
        QUICKSORT (A, s + 1, r);
    end if
end QUICKSORT
```

Algorithm 9.5 The sequential quicksort algorithm.

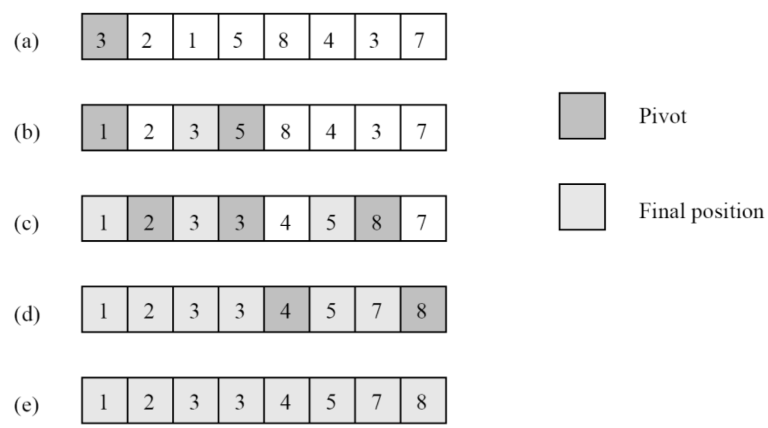


Figure 9.15 Example of the quicksort algorithm sorting a sequence of size $n = 8$.

Minimum Finding

```
procedure SERIAL_MIN (A, n)
begin
min = A[0];
for i := 1 to n − 1 do
    if (A[i] < min) min := A[i];
endfor;
return min;
end SERIAL_MIN
```

Algorithm 3.1 A serial program for finding the minimum in an array of numbers A of length n.

15—Puzzle Problem

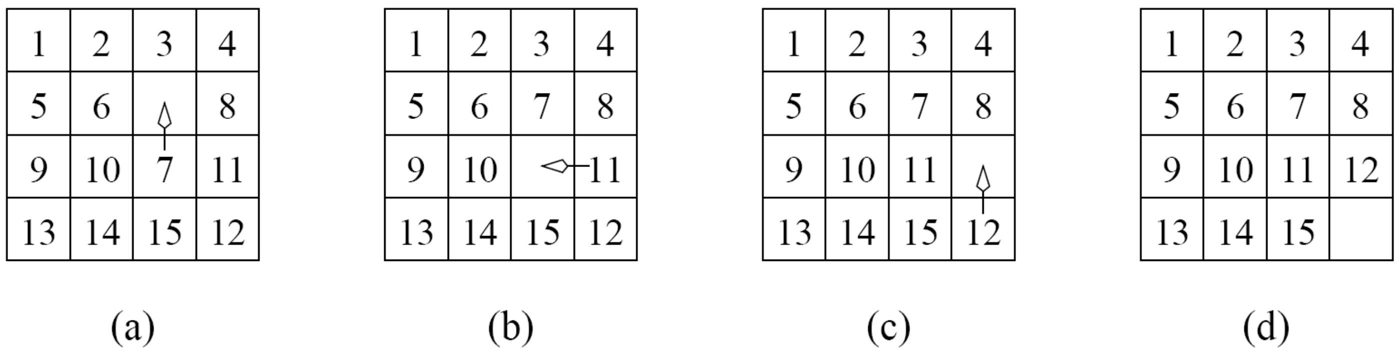


Figure 3.17 A 15-puzzle problem instance showing the initial configuration (a), the final configuration (d), and a sequence of moves leading from the initial to the final configuration.