CHAPTER 15

Graph traversal

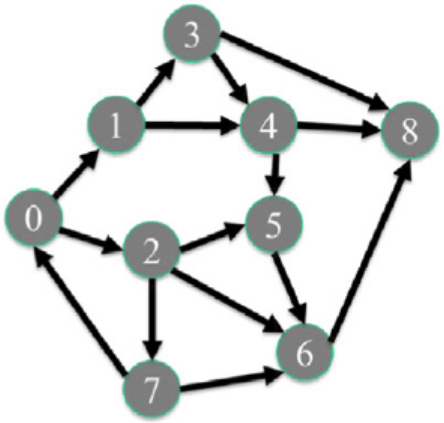
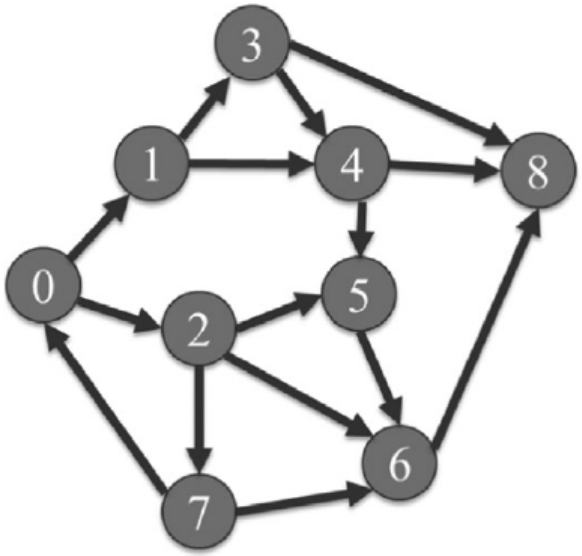


FIGURE 15.1

A simple graph example with 9 vertices and 15 directional edges.



destination vertex

source vertex	0	1	2	3	4	5	6	7	8
0		1	1						
1				1	1				
2						1	1	1	
3					1				1
4						1			1
5							1		
6									1
7	1						1		
8									

FIGURE 15.2

Adjacency matrix representation of the simple graph example.

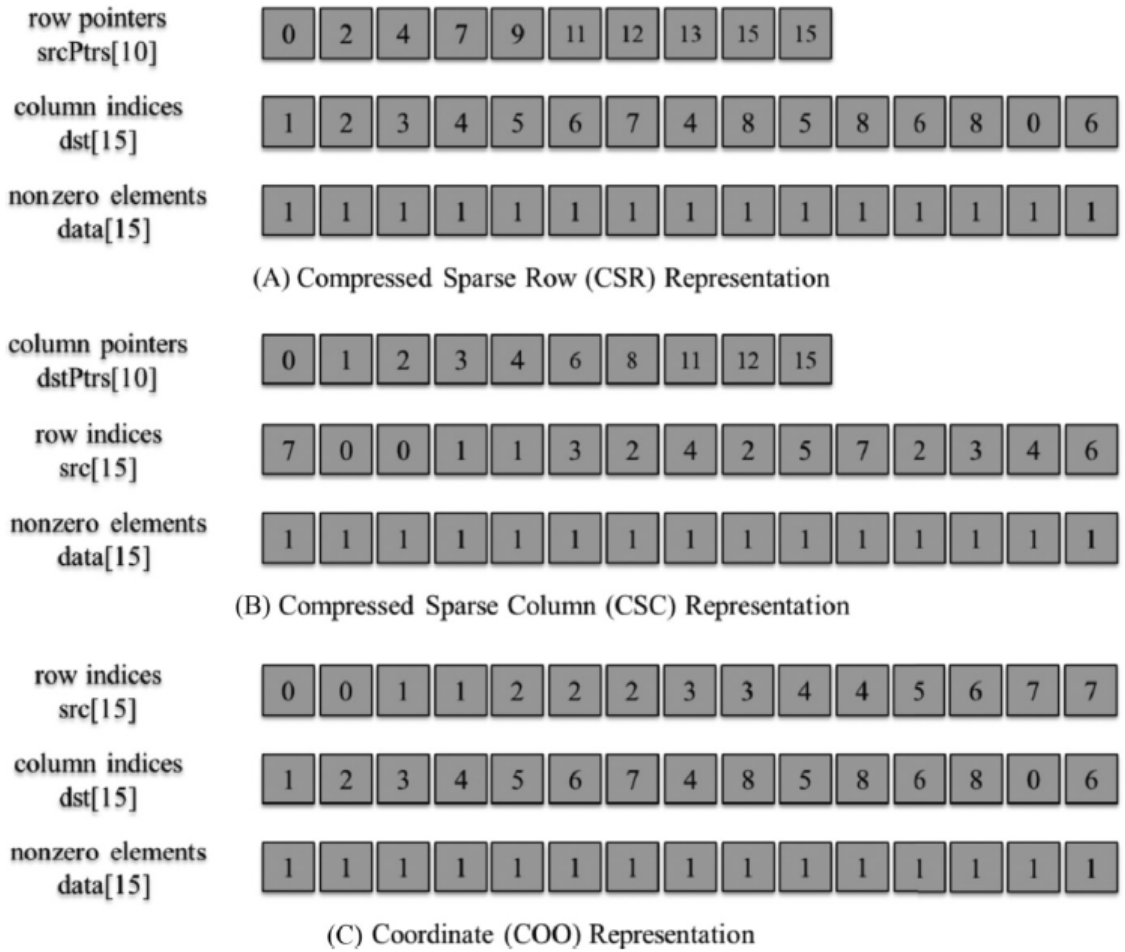


FIGURE 15.3

Three sparse matrix representations of the adjacency matrix: (A) CSR, (B) CSC, (C) COO. *COO*, coordinate; *CSC*, compressed sparse column; *CSR*, compressed sparse row.

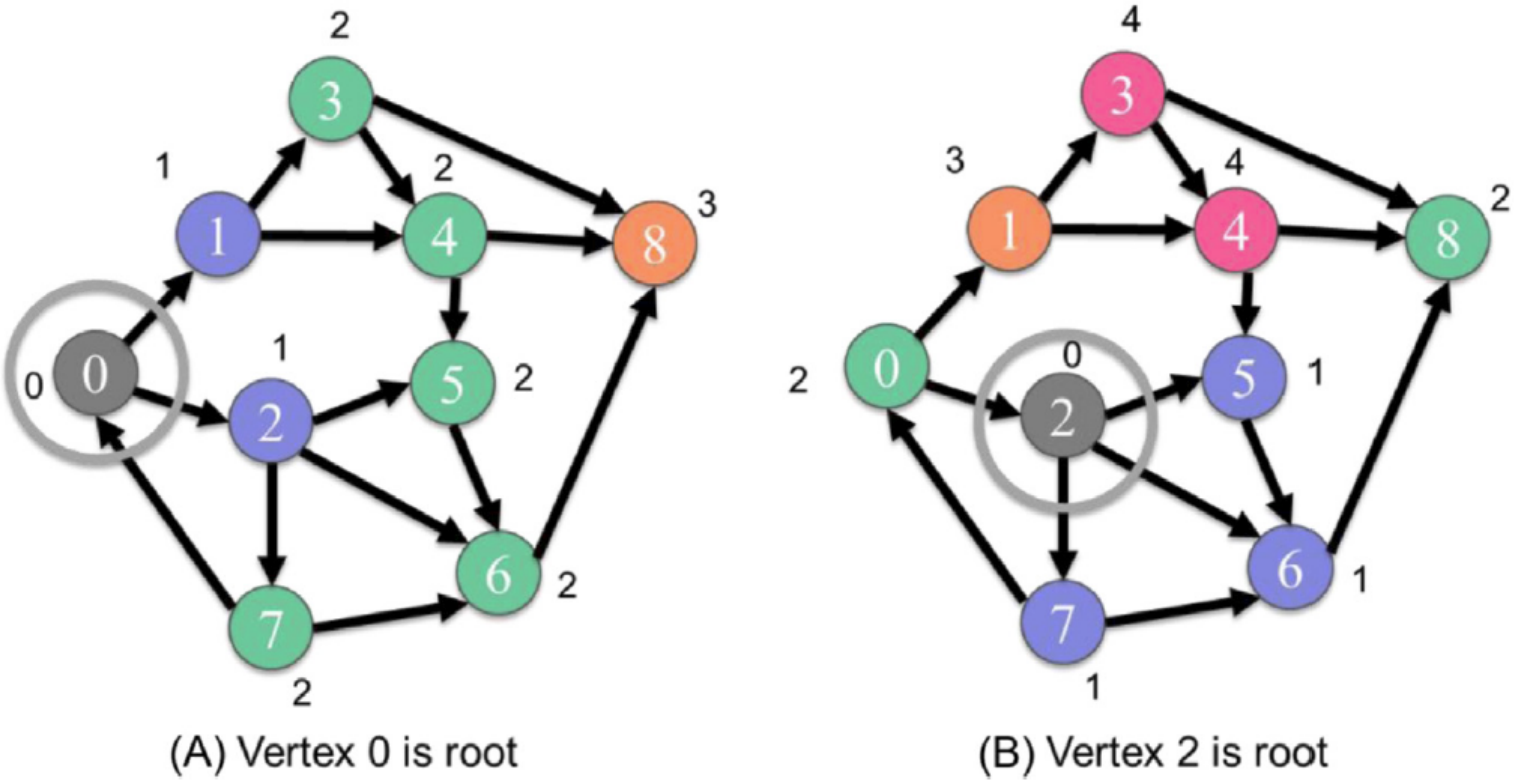
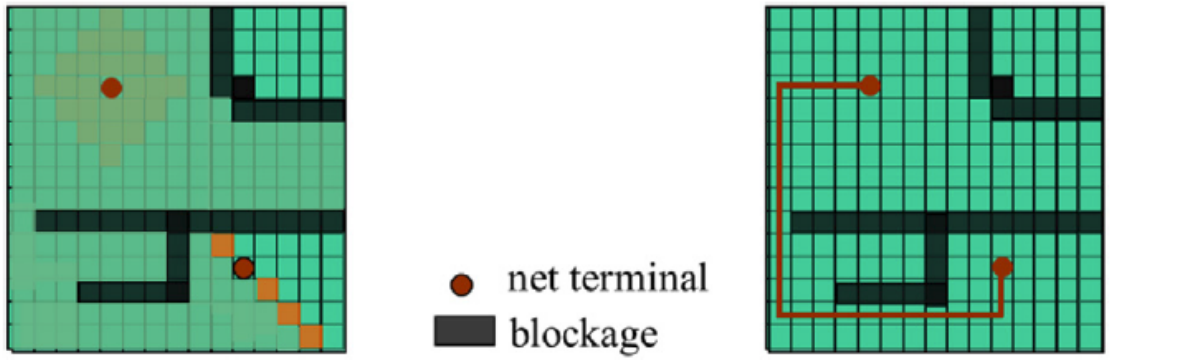


FIGURE 15.4

(A and B) Two examples of breadth-first search results for two different root vertices. The labels adjacent to each vertex indicate the number of hops (depth) from the root vertex.



(A) Breadth-first search (B) Identifying a routing path

FIGURE 15.5

Maze routing in integrated circuits—an application for breadth-first search: (A) breadth-first search, (B) identifying a routing path.

```
01  __global__ void bfs_kernel(CSRGraph csrGraph, unsigned int* level,
02                   unsigned int* newVertexVisited, unsigned int currLevel) {
03      unsigned int vertex = blockIdx.x*blockDim.x + threadIdx.x;
04      if(vertex < csrGraph.numVertices) {
05          if(level[vertex] == currLevel - 1) {
06              for(unsigned int edge = csrGraph.srcPtrs[vertex];
07                      edge < csrGraph.srcPtrs[vertex + 1]; ++edge) {
08                  unsigned int neighbor = csrGraph.dst[edge];
09                  if(level[neighbor] == UINT_MAX) { // Neighbor not visited
10                      level[neighbor] = currLevel;
11                      *newVertexVisited = 1;
12                  }
13              }
14          }
15      }
16  }
```

FIGURE 15.6

A vertex-centric push (top-down) BFS kernel. *BFS*, breadth-first search.

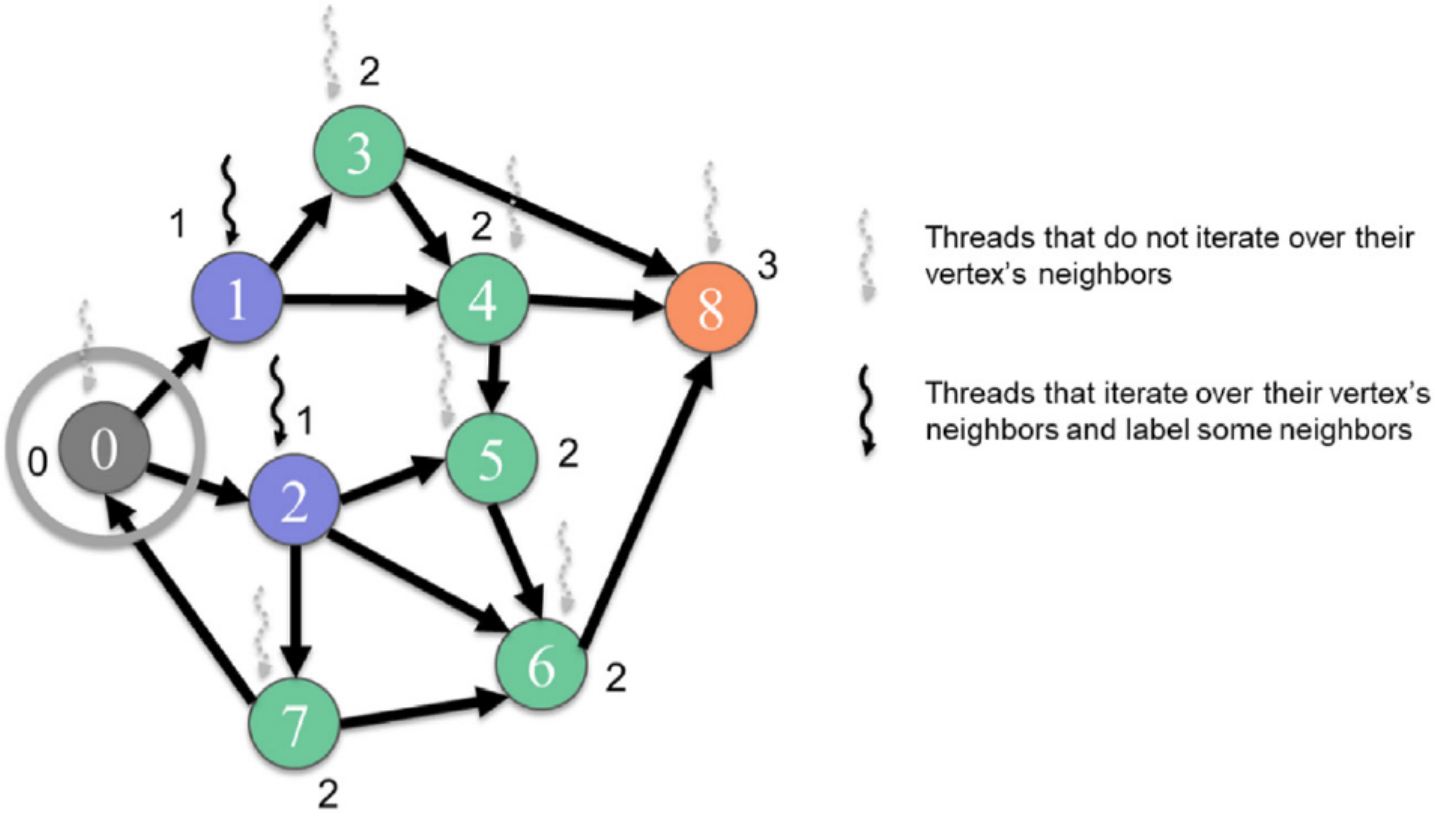


FIGURE 15.7

Example of a vertex-centric push BFS traversal from level 1 to level 2. *BFS*, breadth-first search.

```
01  __global__ void bfs_kernel(CSCGraph cscGraph, unsigned int* level,
02                  unsigned int* newVertexVisited, unsigned int currLevel) {
03      unsigned int vertex = blockIdx.x*blockDim.x + threadIdx.x;
04      if(vertex < cscGraph.numVertices) {
05          if(level[vertex] == UINT_MAX) { // Vertex not yet visited
06              for(unsigned int edge = cscGraph.dstPtrs[vertex];
07                      edge < cscGraph.dstPtrs[vertex + 1]; ++edge) {
08                  unsigned int neighbor = cscGraph.src[edge];
09                  if(level[neighbor] == currLevel - 1) {
10                      level[vertex] = currLevel;
11                      *newVertexVisited = 1;
12                      break;
13                  }
14              }
15          }
16      }
17  }
```

FIGURE 15.8

A vertex-centric pull (bottom-up) BFS kernel. *BFS*, breadth-first search.

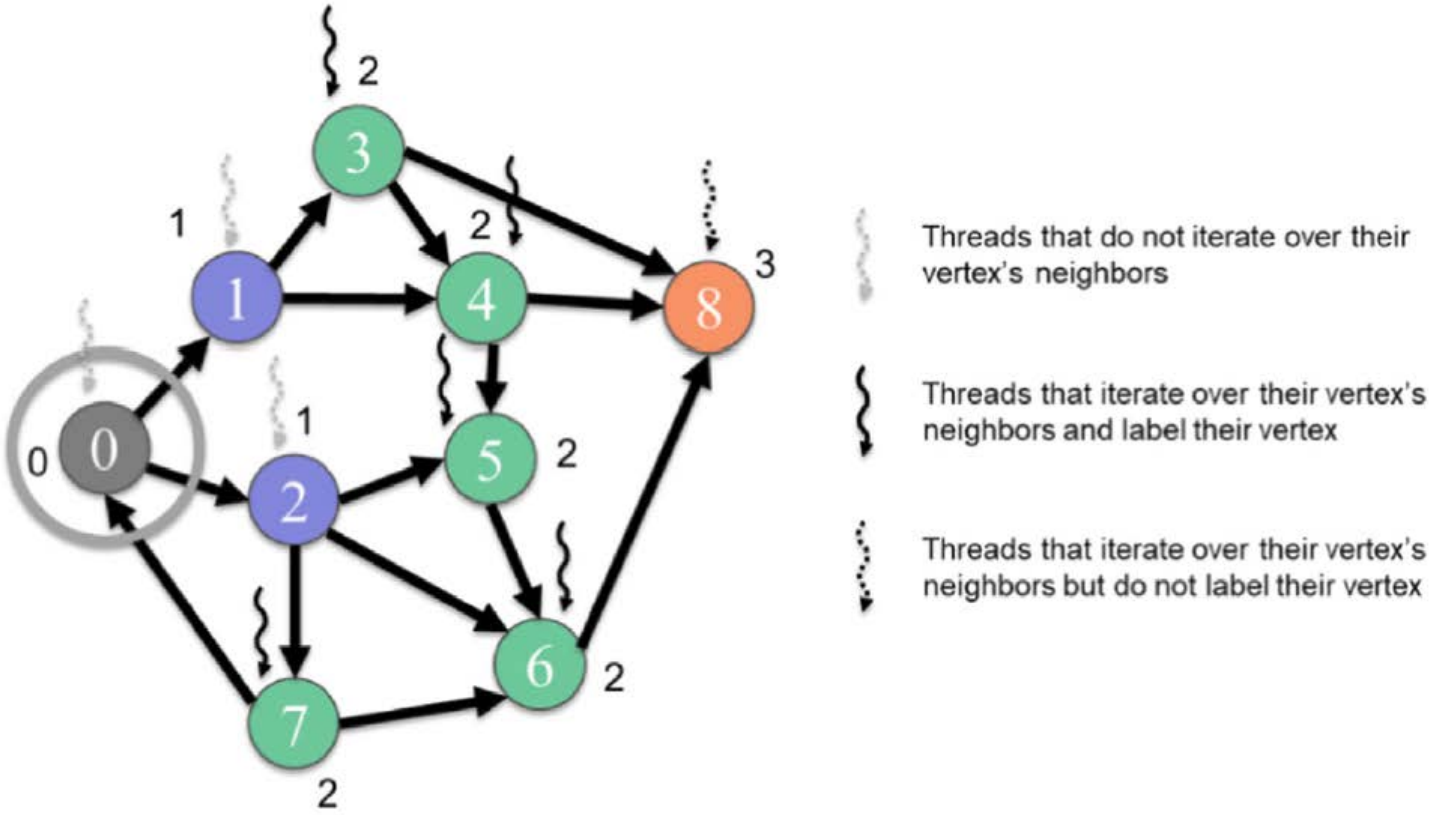


FIGURE 15.9

Example of a vertex-centric pull (bottom-up) traversal from level 1 to level 2.

```
01  __global__ void bfs_kernel(COOGraph cooGraph, unsigned int* level,
02                              unsigned int* newVertexVisited, unsigned int currLevel) {
03      unsigned int edge = blockIdx.x*blockDim.x + threadIdx.x;
04      if(edge < cooGraph.numEdges) {
05          unsigned int vertex = cooGraph.src[edge];
06          if(level[vertex] == currLevel - 1) {
07              unsigned int neighbor = cooGraph.dst[edge];
08              if(level[neighbor] == UINT_MAX) { // Neighbor not visited
09                  level[neighbor] = currLevel;
10                  *newVertexVisited = 1;
11              }
12          }
13      }
14  }
```

FIGURE 15.10

An edge-centric BFS kernel. *BFS*, breadth-first search.

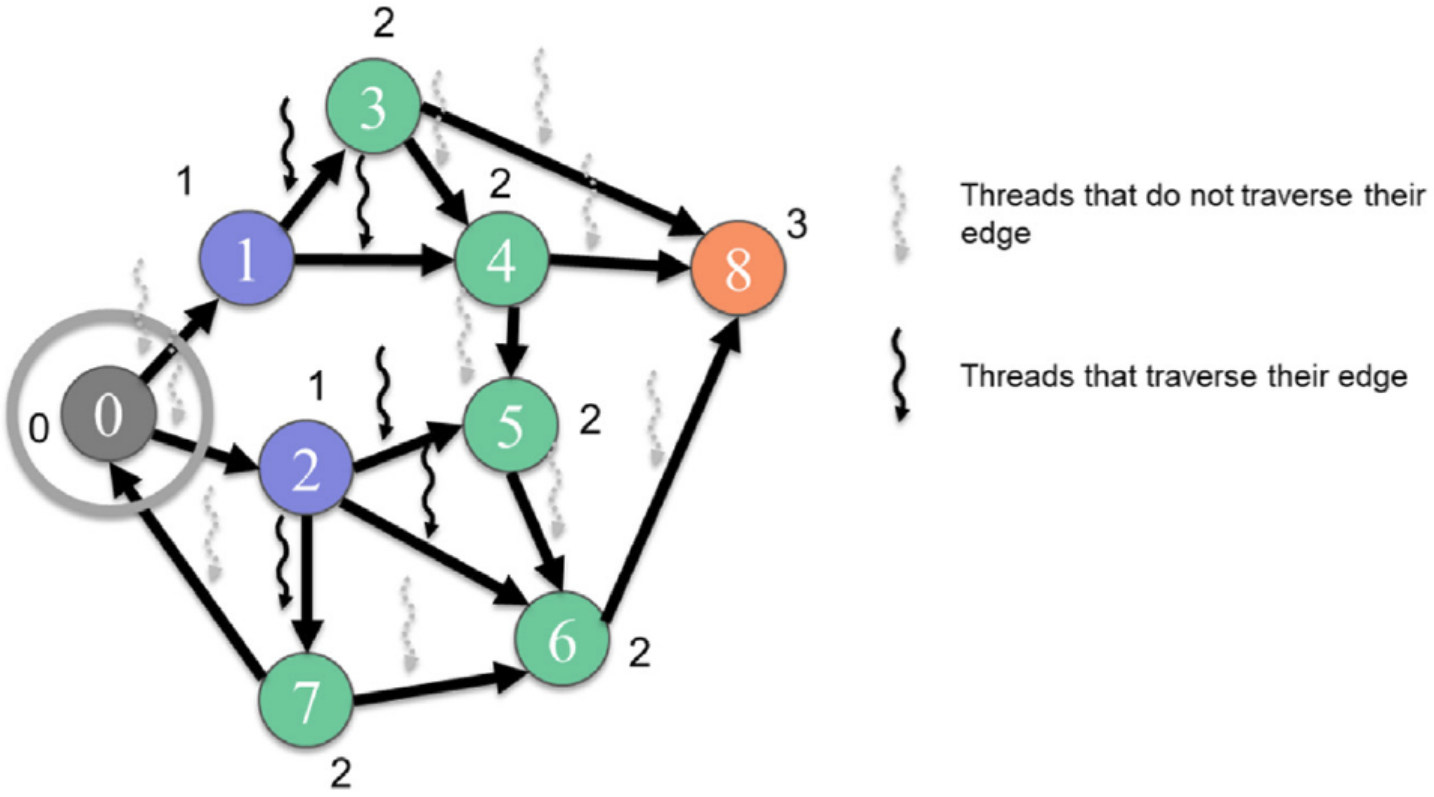


FIGURE 15.11

Example of an edge-centric traversal from level 1 to level 2.

```
01  __global__ void bfs_kernel(CSRGraph csrGraph, unsigned int* level,
02              unsigned int* prevFrontier, unsigned int* currFrontier,
03              unsigned int numPrevFrontier, unsigned int* numCurrFrontier,
04              unsigned int currLevel) {
05      unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
06      if(i < numPrevFrontier) {
07          unsigned int vertex = prevFrontier[i];
08          for(unsigned int edge = csrGraph.srcPtrs[vertex];
09                  edge < csrGraph.srcPtrs[vertex + 1]; ++edge) {
10              unsigned int neighbor = csrGraph.dst[edge];
11              if(atomicCAS(&level[neighbor],UINT_MAX,currLevel) == UINT_MAX) {
12                  unsigned int currFrontierIdx = atomicAdd(numCurrFrontier, 1);
13                  currFrontier[currFrontierIdx] = neighbor;
14              }
15          }
16      }
17  }
```

FIGURE 15.12

A vertex-centric push (top-down) BFS kernel with frontiers. *BFS*, breadth-first search.

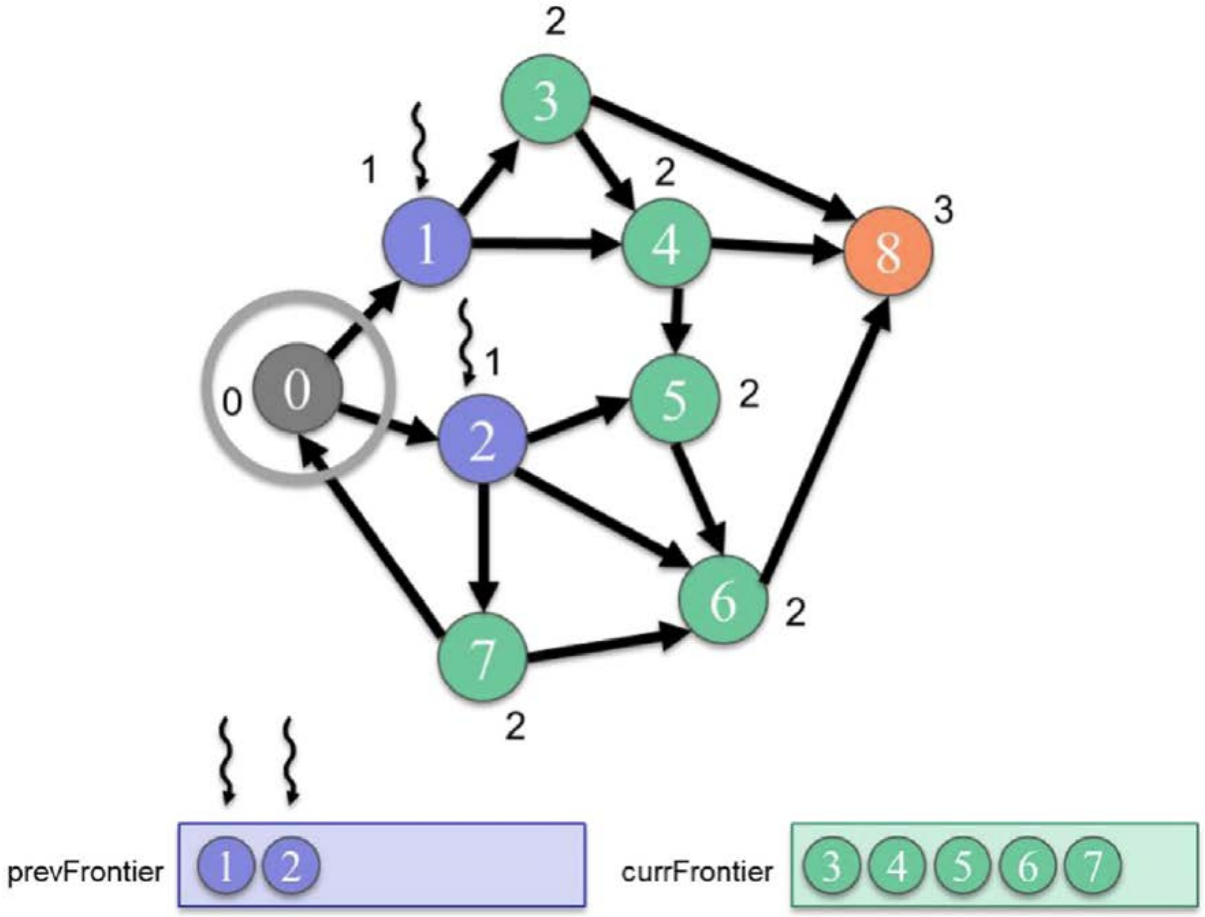


FIGURE 15.13

Example of a vertex-centric push (top-down) BFS traversal from level 1 to level 2 with frontiers. *BFS*, breadth-first search.

```
01  __global__ void bfs_kernel(CSRGraph csrGraph, unsigned int* level,
02              unsigned int* prevFrontier, unsigned int* currFrontier,
03              unsigned int numPrevFrontier, unsigned int* numCurrFrontier,
04              unsigned int currLevel) {
05
06      // Initialize privatized frontier
07        shared   unsigned int currFrontier_s[LOCAL_FRONTIER_CAPACITY];
08        shared   unsigned int numCurrFrontier_s;
09      if(threadIdx.x == 0) {
10          numCurrFrontier_s = 0;
11      }
12        syncthreads();
13
14      // Perform BFS
15      unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
16      if(i < numPrevFrontier) {
17          unsigned int vertex = prevFrontier[i];
18          for(unsigned int edge = csrGraph.srcPtrs[vertex];
19                  edge < csrGraph.srcPtrs[vertex + 1]; ++edge) {
20              unsigned int neighbor = csrGraph.dst[edge];
21              if(atomicCAS(&level[neighbor], UINT_MAX, currLevel) == UINT_MAX) {
22                  unsigned int currFrontierIdx_s = atomicAdd(&numCurrFrontier_s, 1);
23                  if(currFrontierIdx_s < LOCAL_FRONTIER_CAPACITY) {
24                      currFrontier_s[currFrontierIdx_s] = neighbor;
25                  } else {
26                      numCurrFrontier_s = LOCAL_FRONTIER_CAPACITY;
27                      unsigned int currFrontierIdx = atomicAdd(numCurrFrontier, 1);
28                      currFrontier[currFrontierIdx] = neighbor;
29                  }
30              }
31          }
32      }
33      __syncthreads();
34
35      // Allocate in global frontier
36      __shared__ unsigned int currFrontierStartIdx;
37      if(threadIdx.x == 0) {
38          currFrontierStartIdx = atomicAdd(numCurrFrontier, numCurrFrontier_s);
39      }
40        syncthreads();
41
42      // Commit to global frontier
43      for(unsigned int currFrontierIdx_s = threadIdx.x;
44              currFrontierIdx_s < numCurrFrontier_s; currFrontierIdx_s += blockDim.x) {
45          unsigned int currFrontierIdx = currFrontierStartIdx + currFrontierIdx_s;
46          currFrontier[currFrontierIdx] = currFrontier_s[currFrontierIdx_s];
47      }
48
49  }
```

FIGURE 15.14

A vertex-centric push (top-down) BFS kernel with privatization of frontiers. *BFS*, breadth-first search.

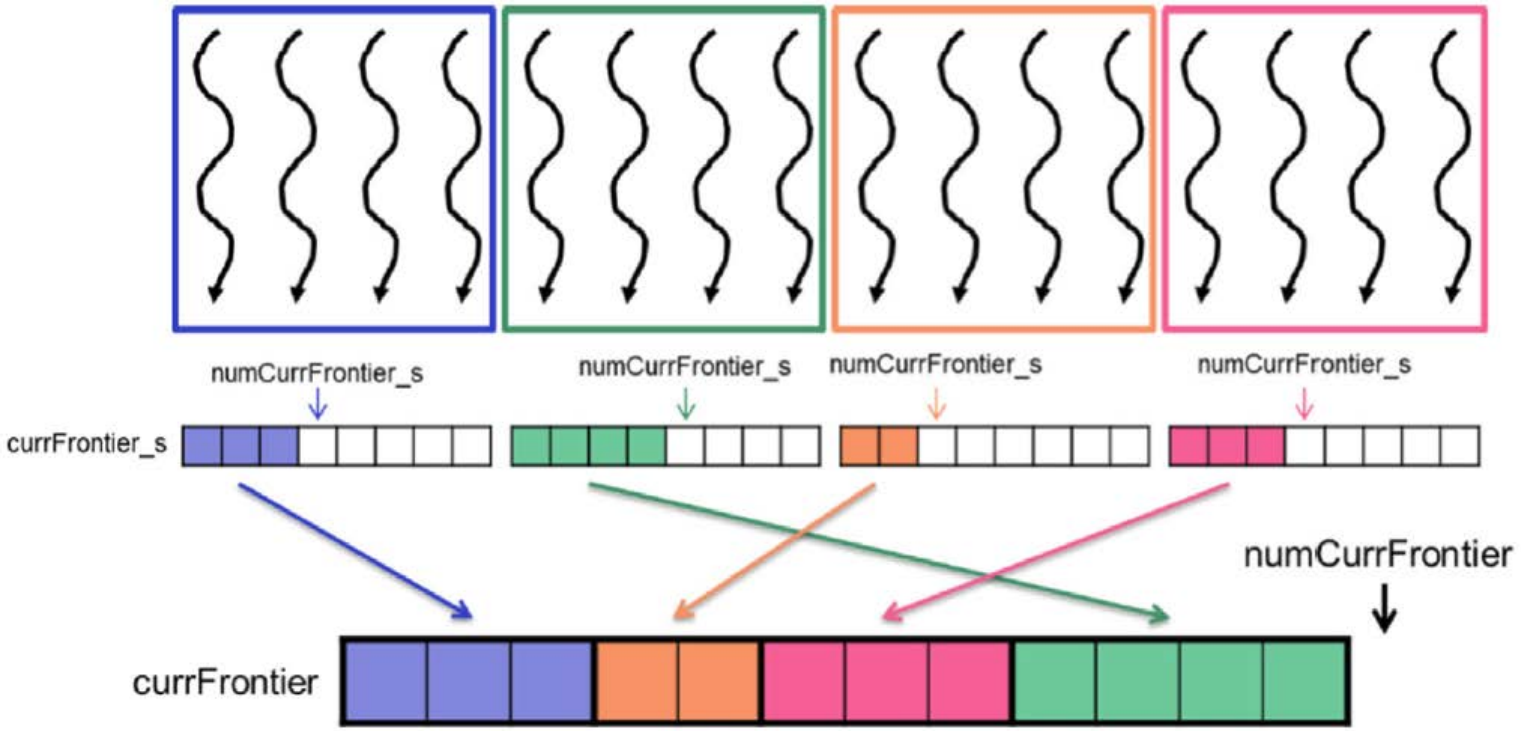


FIGURE 15.15

Privatization of frontiers example.

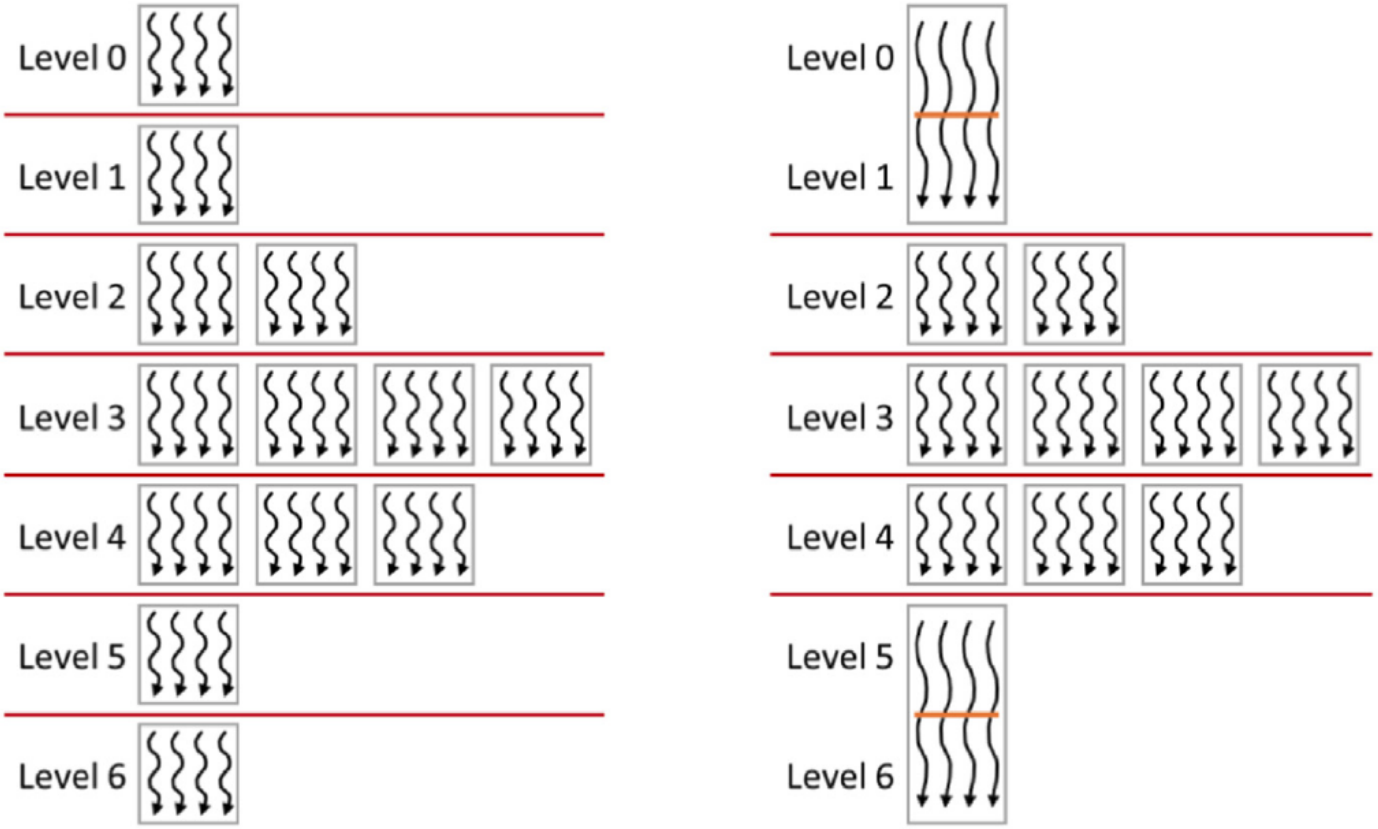
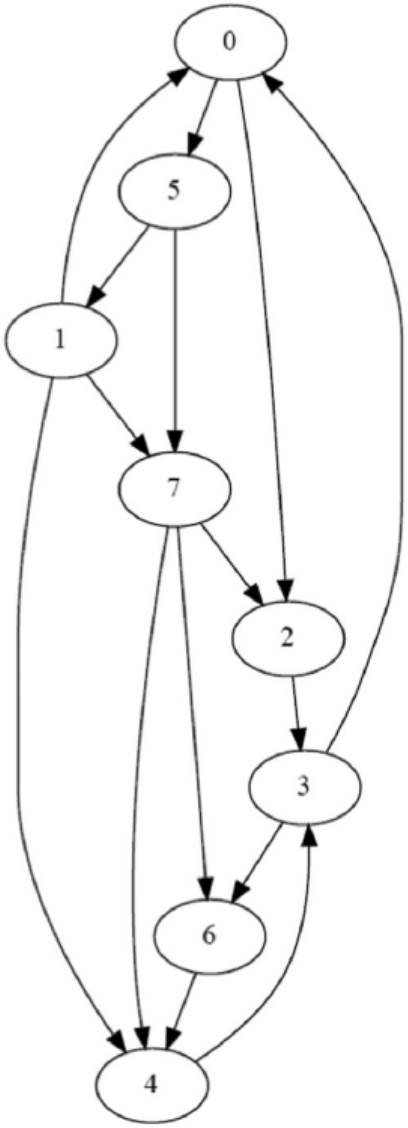


FIGURE 15.16

Executing multiple levels in one grid for levels with small frontiers: (A) launching a new grid for each level, (B) consecutive small levels in one grid.



In-text figure 1