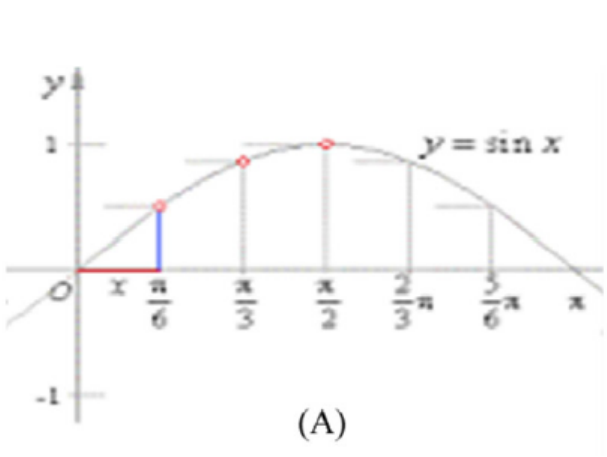
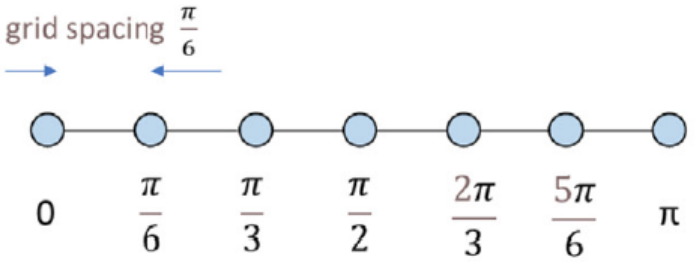
CHAPTER 8

Stencil



(A)



(B)

0.00	0.50	0.87	1.00	0.87	0.50	0.00

(C)

FIGURE 8.1

(A) Sine as a continuous, differentiable function for $0 \leq x \leq \pi$. (B) Design of a regular grid with constant spacing ($\frac{\pi}{6}$) between grid point for discretization. (C) Resulting discrete representation of the sine function for $0 \leq x \leq \pi$.

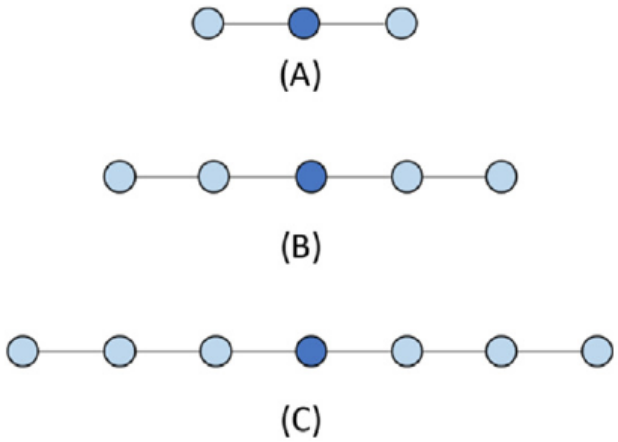


FIGURE 8.2

One-dimensional stencil examples. (A) Three-point (order 1) stencil. (B) Five-point (order 2) stencil. (C) Seven-point (order 3) stencil.

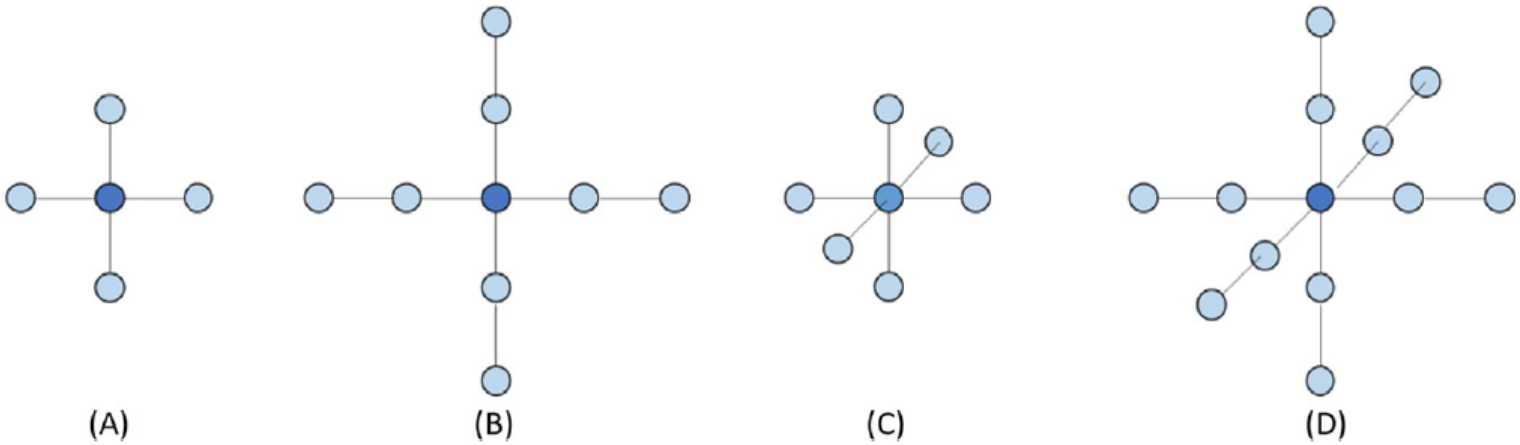


FIGURE 8.3

(A) Two-dimensional five-point stencil (order 1). (B) Two-dimensional nine-point stencil (order 2). (C) Three-dimensional seven-point stencil (order 1). (D) Three-dimensional 13-point stencil (order 2).

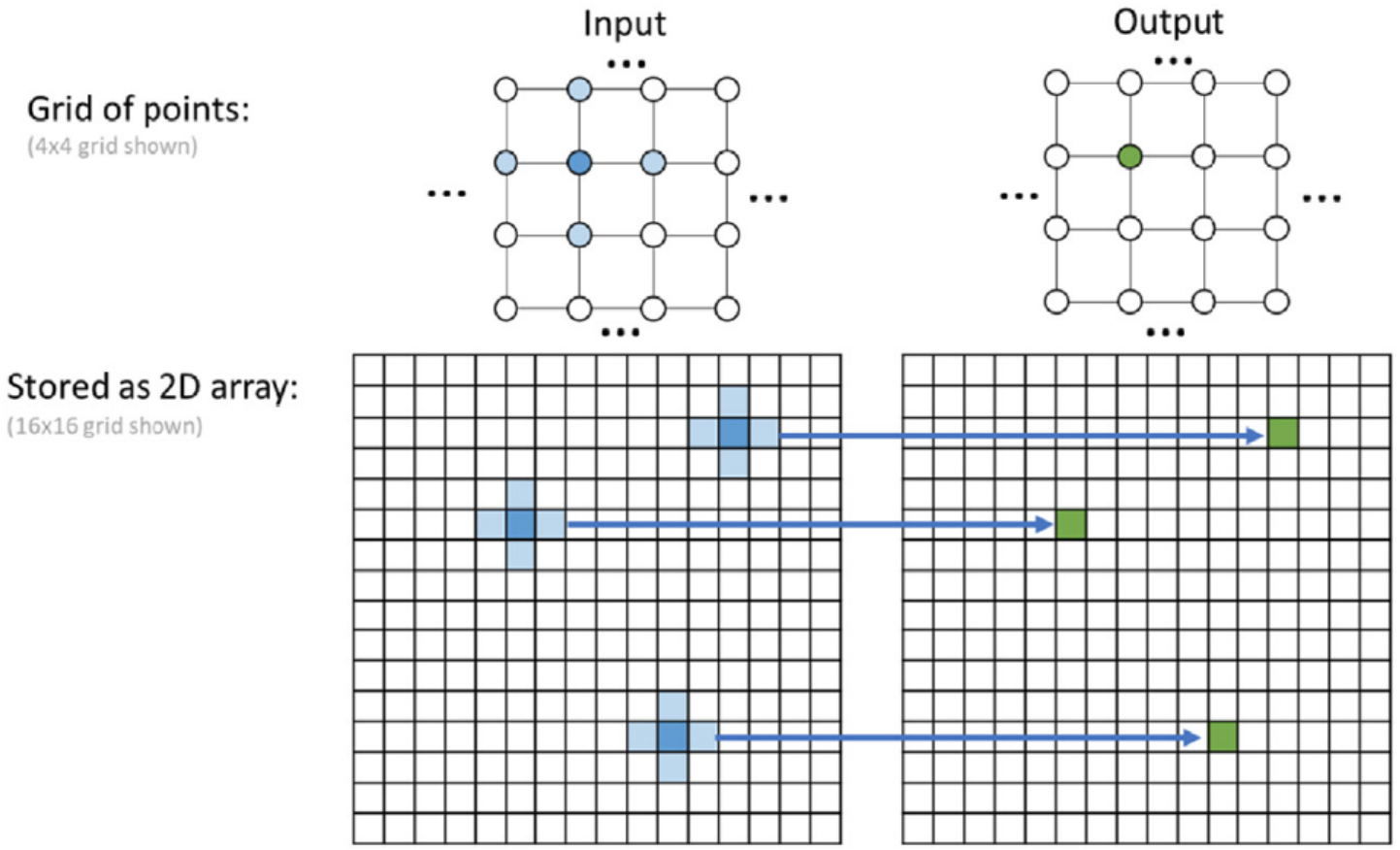


FIGURE 8.4

A 2D grid example and a five-point (order 1) stencil used to calculate the approximate derivative values at grid points.

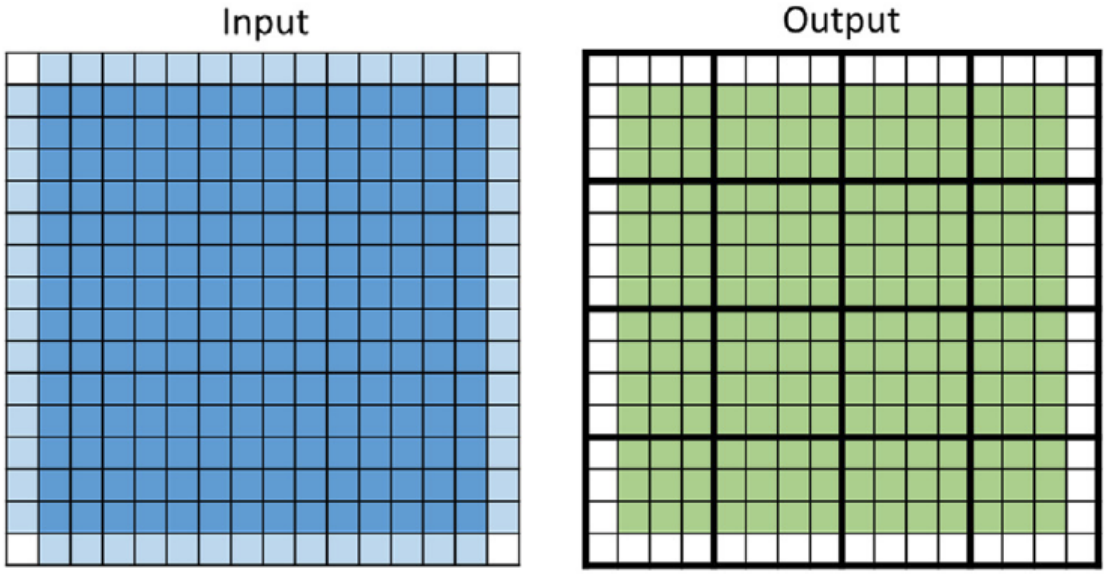


FIGURE 8.5

Simplifying boundary condition. The boundary cells contain boundary conditions that will not be updated from one iteration to the next. Thus only the inner output grid points need to be calculated during each stencil sweep.

```
01   __global__ void stencil_kernel(float* in, float* out, unsigned int N) {
02     unsigned int i = blockIdx.z*blockDim.z + threadIdx.z;
03     unsigned int j = blockIdx.y*blockDim.y + threadIdx.y;
04     unsigned int k = blockIdx.x*blockDim.x + threadIdx.x;
05     if(i >= 1 && i < N - 1 && j >= 1 && j < N - 1&& k >= 1 && k < N - 1) {
06         out[i*N*N + j*N + k] = c0*in[i*N*N + j*N + k]
07                              + c1*in[i*N*N + j*N + (k - 1)]
08                              + c2*in[i*N*N + j*N + (k + 1)]
09                              + c3*in[i*N*N + (j - 1)*N + k]
10                              + c4*in[i*N*N + (j + 1)*N + k]
11                              + c5*in[(i - 1)*N*N + j*N + k]
12                              + c6*in[(i + 1)*N*N + j*N + k];
13     }
14   }
```

FIGURE 8.6

A basic stencil sweep kernel.

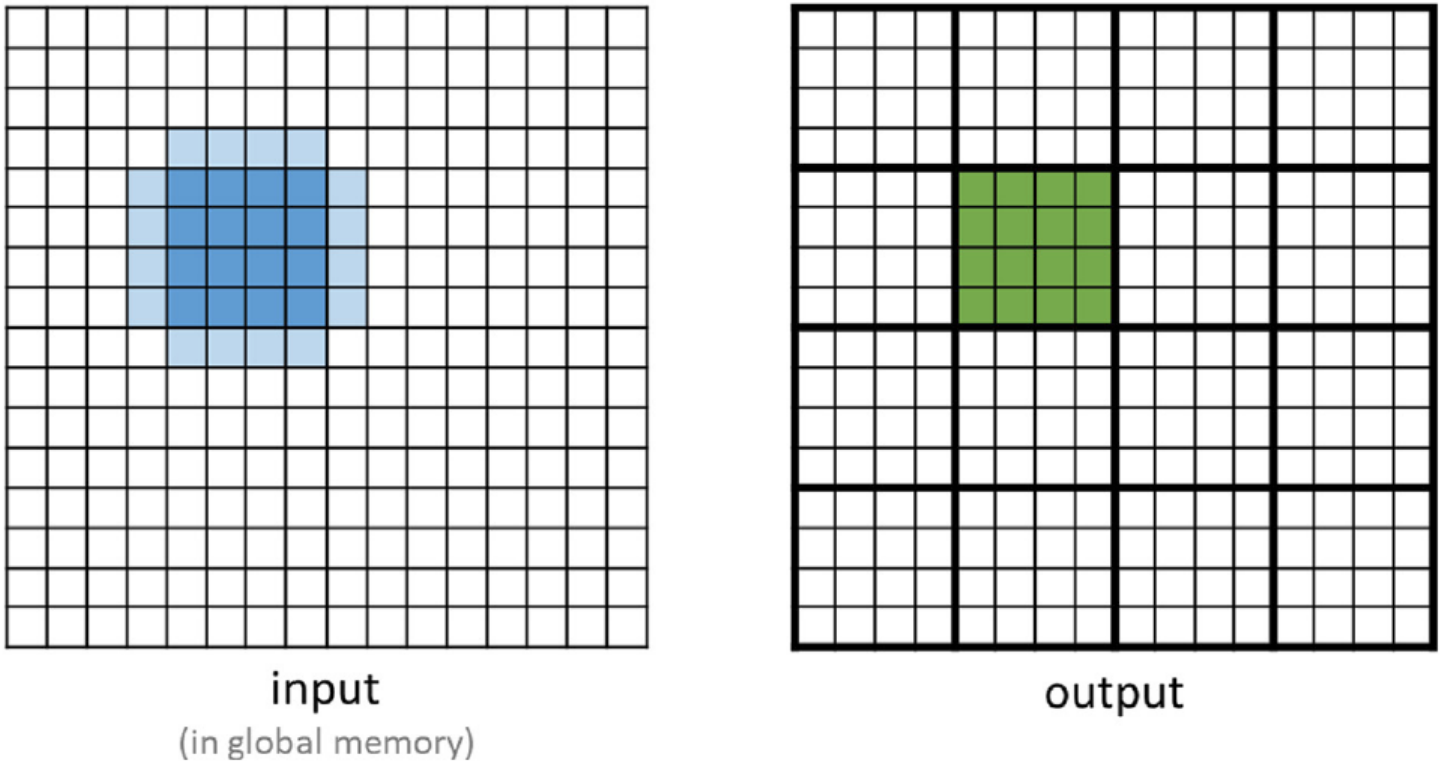


FIGURE 8.7

Input and output tiles for a 2D five-point stencil.

```
01 __global__ void stencil_kernel(float* in, float* out, unsigned int N) {
02    int i = blockIdx.z*OUT_TILE_DIM + threadIdx.z - 1;
03    int j = blockIdx.y*OUT_TILE_DIM + threadIdx.y - 1;
04    int k = blockIdx.x*OUT_TILE_DIM + threadIdx.x - 1;
05      __shared__ float in_s[IN_TILE_DIM][IN_TILE_DIM][IN_TILE_DIM];
06    if(i >= 0 && i < N && j >= 0 && j < N && k >= 0 && k < N) {
07        in_s[threadIdx.z][threadIdx.y][threadIdx.x] = in[i*N*N + j*N + k];
08    }
09     __syncthreads();
10    if(i >= 1 && i < N-1 && j >= 1 && j < N-1 && k >= 1 && k < N-1) {
11      if(threadIdx.z >= 1 && threadIdx.z < IN_TILE_DIM-1 && threadIdx.y >= 1
12        && threadIdx.y<IN_TILE_DIM-1 && threadIdx.x>=1 && threadIdx.x<IN_TILE_DIM-1){
13           out[i*N*N + j*N + k] = c0*in_s[threadIdx.z][threadIdx.y][threadIdx.x]
14                                + c1*in_s[threadIdx.z][threadIdx.y][threadIdx.x-1]
15                                + c2*in_s[threadIdx.z][threadIdx.y][threadIdx.x+1]
16                                + c3*in_s[threadIdx.z][threadIdx.y-1][threadIdx.x]
17                                + c4*in_s[threadIdx.z][threadIdx.y+1][threadIdx.x]
18                                + c5*in_s[threadIdx.z-1][threadIdx.y][threadIdx.x]
19                                + c6*in_s[threadIdx.z+1][threadIdx.y][threadIdx.x];
20     }
21    }
22 }
```

FIGURE 8.8

A 3D seven-point stencil sweep kernel with shared memory tiling.

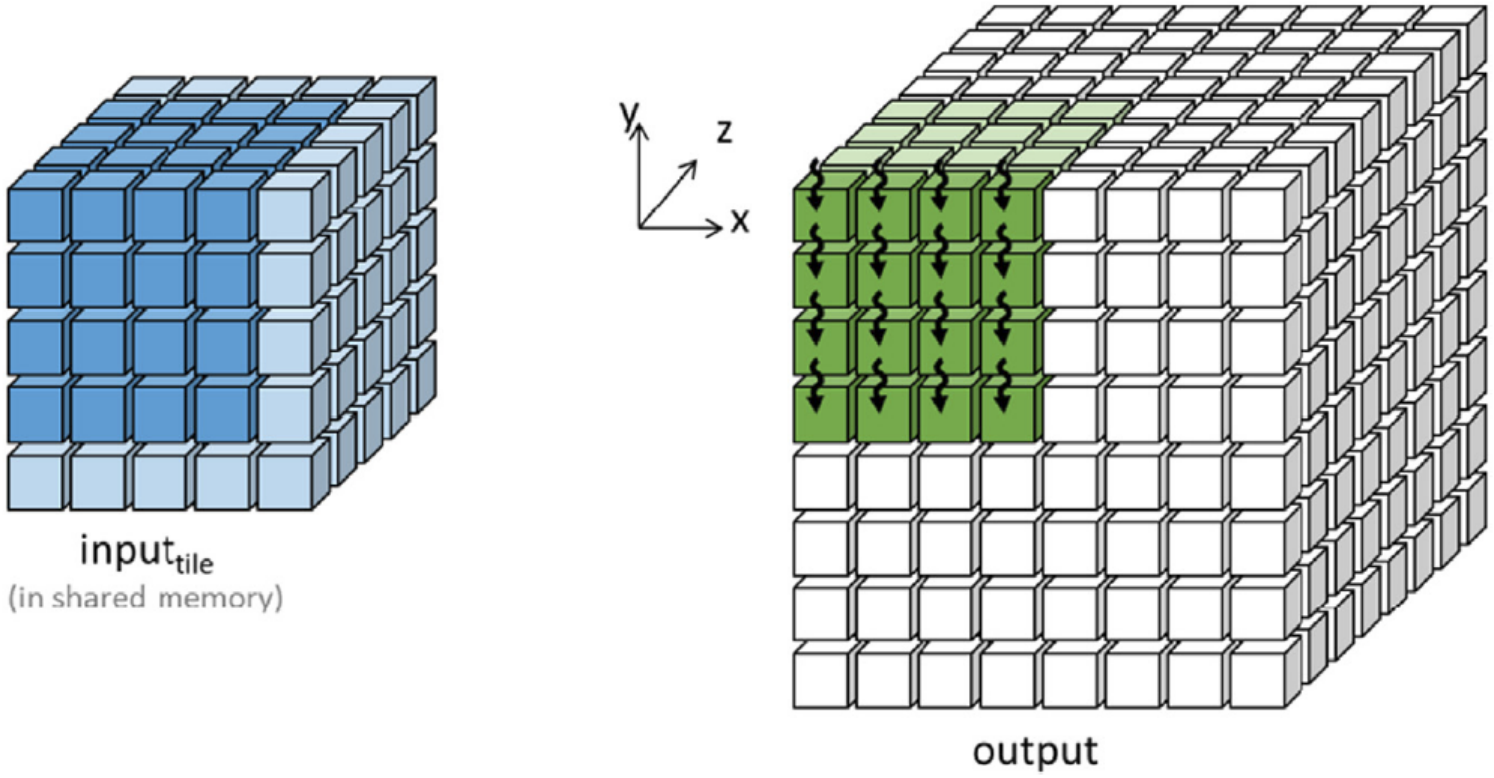


FIGURE 8.9

Thread coarsening in the z direction for a 3D seven-point stencil sweep.

```
01 __global__ void stencil_kernel(float* in, float* out, unsigned int N) {
02    int iStart = blockIdx.z*OUT_TILE_DIM;
03    int j = blockIdx.y*OUT_TILE_DIM + threadIdx.y - 1;
04    int k = blockIdx.x*OUT_TILE_DIM + threadIdx.x - 1;
05    __shared__ float inPrev_s[IN_TILE_DIM][IN_TILE_DIM];
06    __shared__ float inCurr_s[IN_TILE_DIM][IN_TILE_DIM];
07    __shared__ float inNext_s[IN_TILE_DIM][IN_TILE_DIM];
08    if(iStart-1 >= 0 && iStart-1 < N && j >= 0 && j < N && k >= 0 && k < N) {
09        inPrev_s[threadIdx.y][threadIdx.x] = in[(iStart - 1)*N*N + j*N + k];
10    }
11    if(iStart >= 0 && iStart < N && j >= 0 && j < N && k >= 0 && k < N) {
12        inCurr_s[threadIdx.y][threadIdx.x] = in[iStart*N*N + j*N + k];
13    }
14    for(int i = iStart; i < iStart + OUT_TILE_DIM; ++i) {
15      if(i + 1 >= 0 && i + 1 < N && j >= 0 && j < N && k >= 0 && k < N) {
16           inNext_s[threadIdx.y][threadIdx.x] = in[(i + 1)*N*N + j*N + k];
17      }
18      __syncthreads();
19      if(i >= 1 && i < N - 1 && j >= 1 && j < N - 1 && k >= 1 && k < N - 1) {
20        if(threadIdx.y >= 1 && threadIdx.y < IN_TILE_DIM - 1
21           && threadIdx.x >= 1 && threadIdx.x < IN_TILE_DIM - 1) {
22           out[i*N*N + j*N + k] = c0*inCurr_s[threadIdx.y][threadIdx.x]
23                                + c1*inCurr_s[threadIdx.y][threadIdx.x-1]
24                                + c2*inCurr_s[threadIdx.y][threadIdx.x+1]
25                                + c3*inCurr_s[threadIdx.y+1][threadIdx.x]
26                                + c4*inCurr_s[threadIdx.y-1][threadIdx.x]
27                                + c5*inPrev_s[threadIdx.y][threadIdx.x]
28                                + c6*inNext_s[threadIdx.y][threadIdx.x];
29        }
30      }
31      __syncthreads();
32      inPrev_s[threadIdx.y][threadIdx.x] = inCurr_s[threadIdx.y][threadIdx.x];
33      inCurr_s[threadIdx.y][threadIdx.x] = inNext_s[threadIdx.y][threadIdx.x];
34    }
35 }
```

FIGURE 8.10

Kernel with thread coarsening in the z direction for a 3D seven-point stencil sweep.

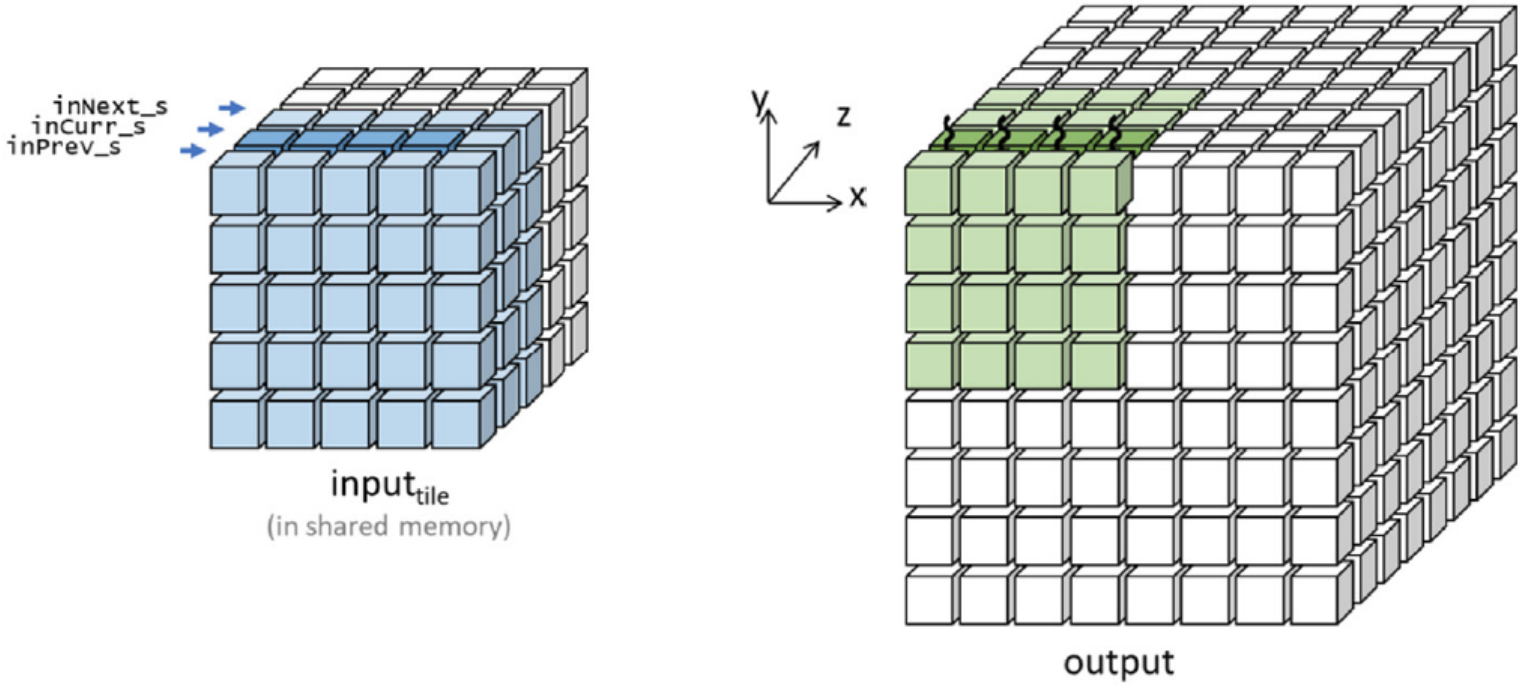


FIGURE 8.11

The mapping of the shared memory arrays to the input tile after the first iteration.

```
01 __global__ void stencil_kernel(float* in, float* out, unsigned int N) {
02    int iStart = blockIdx.z*OUT_TILE_DIM;
03    int j = blockIdx.y*OUT_TILE_DIM + threadIdx.y - 1;
04    int k = blockIdx.x*OUT_TILE_DIM + threadIdx.x - 1;
05    float inPrev;
06    __shared__ float inCurr_s[IN_TILE_DIM][IN_TILE_DIM];
07    float inCurr;
08    float inNext;
09    if(iStart-1 >= 0 && iStart-1 < N && j >= 0 && j < N && k >= 0 && k < N) {
10        inPrev = in[(iStart - 1)*N*N + j*N + k];
11    }
12    if(iStart >= 0 && iStart < N && j >= 0 && j < N && k >= 0 && k < N) {
13        inCurr = in[iStart*N*N + j*N + k];
14        inCurr_s[threadIdx.y][threadIdx.x] = inCurr;
15    }
16    for(int i = iStart; i < iStart + OUT_TILE_DIM; ++i) {
17      if(i + 1 >= 0 && i + 1 < N && j >= 0 && j < N && k >= 0 && k < N) {
18            inNext = in[(i + 1)*N*N + j*N + k];
19      }
20      __syncthreads();
21      if(i >= 1 && i < N - 1 && j >= 1 && j < N - 1 && k >= 1 && k < N - 1) {
22        if(threadIdx.y >= 1 && threadIdx.y < IN_TILE_DIM - 1
23           && threadIdx.x >= 1 && threadIdx.x < IN_TILE_DIM - 1) {
24            out[i*N*N + j*N + k] = c0*inCurr
25                                 + c1*inCurr_s[threadIdx.y][threadIdx.x-1]
26                                 + c2*inCurr_s[threadIdx.y][threadIdx.x+1]
27                                 + c3*inCurr_s[threadIdx.y+1][threadIdx.x]
28                                 + c4*inCurr_s[threadIdx.y-1][threadIdx.x]
29                                 + c5*inPrev
30                                 + c6*inNext;
31        }
32      }
33      __syncthreads();
34      inPrev = inCurr;
35      inCurr = inNext;
34      inCurr_s[threadIdx.y][threadIdx.x] = inNext_s;
35    }
35 }
```

FIGURE 8.12

Kernel with thread coarsening and register tiling in the z direction for a 3D seven-point stencil sweep.