CHAPTER 7

Convolution
An introduction to constant memory and caching

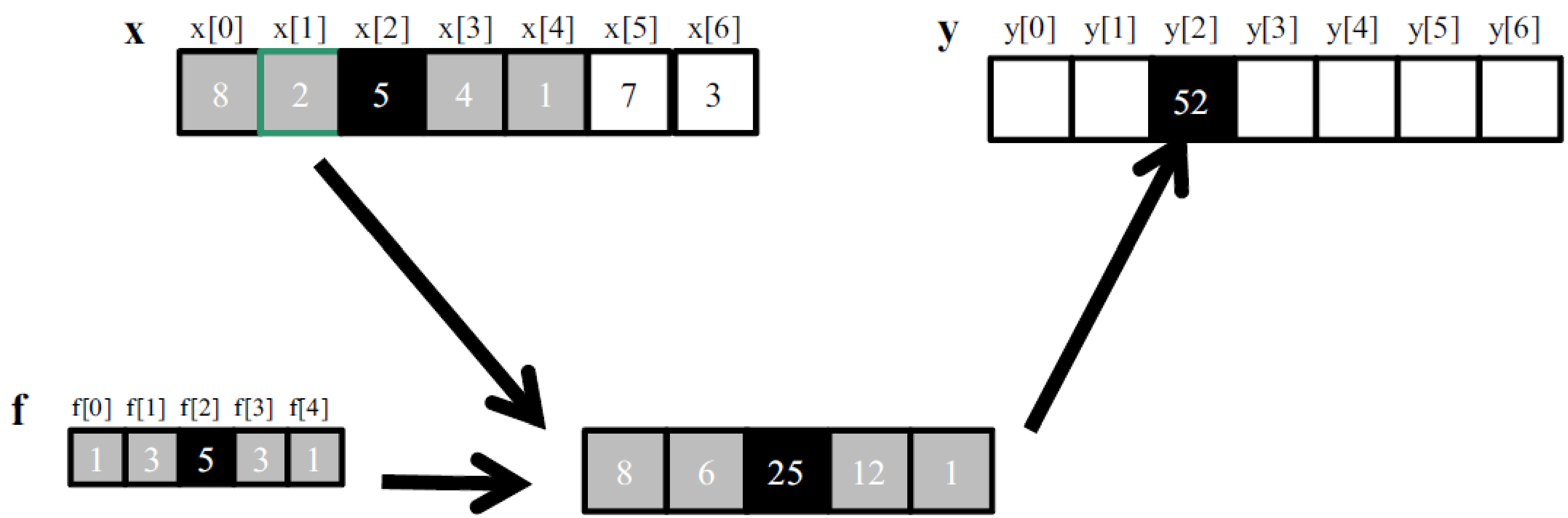

FIGURE 7.1

A 1D convolution example, inside elements.

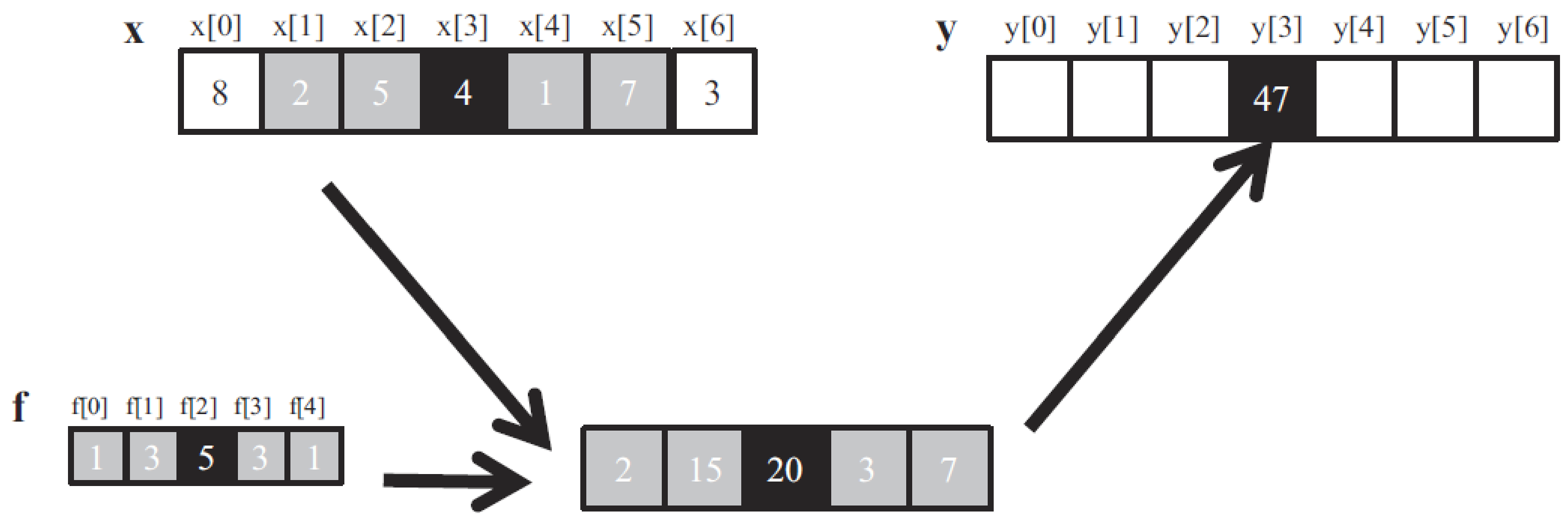

FIGURE 7.2

1D convolution, calculation of y[3].

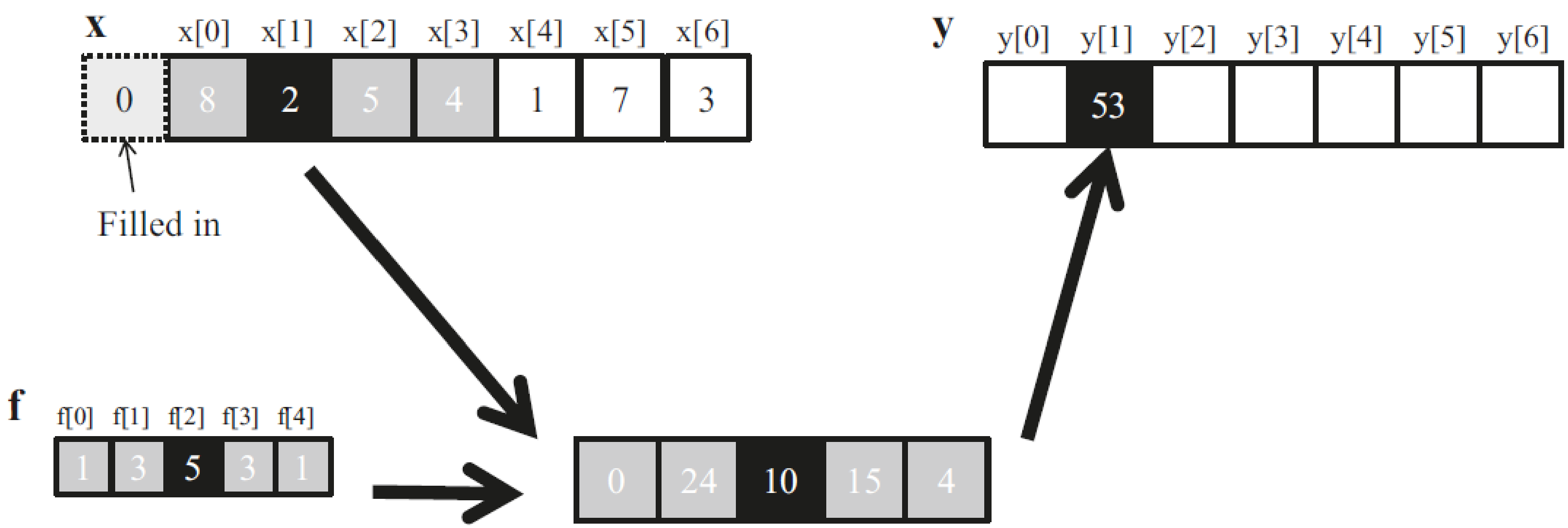

FIGURE 7.3

A 1D convolution boundary condition.

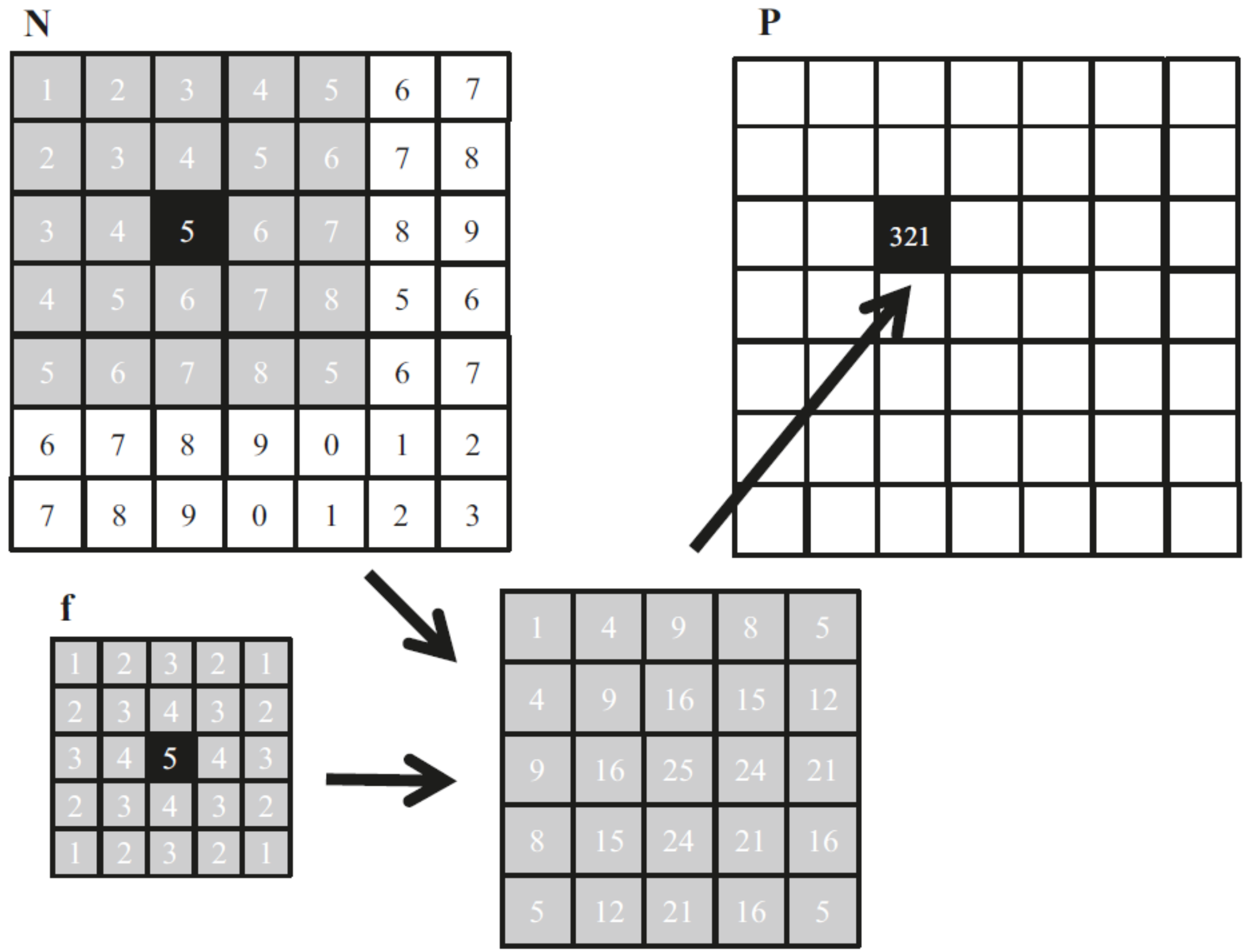

FIGURE 7.4

A 2D convolution example.

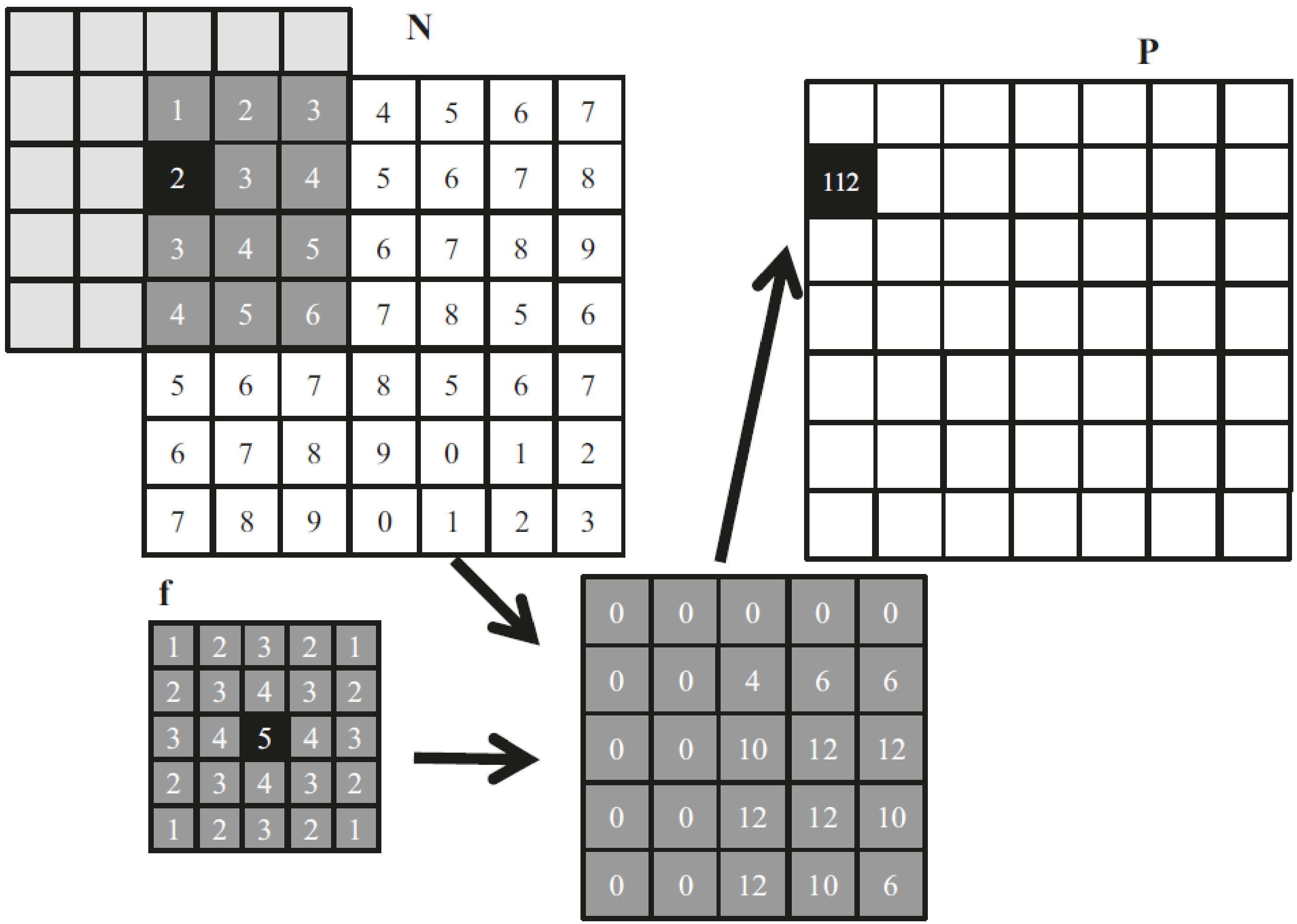

FIGURE 7.5

A 2D convolution boundary condition.

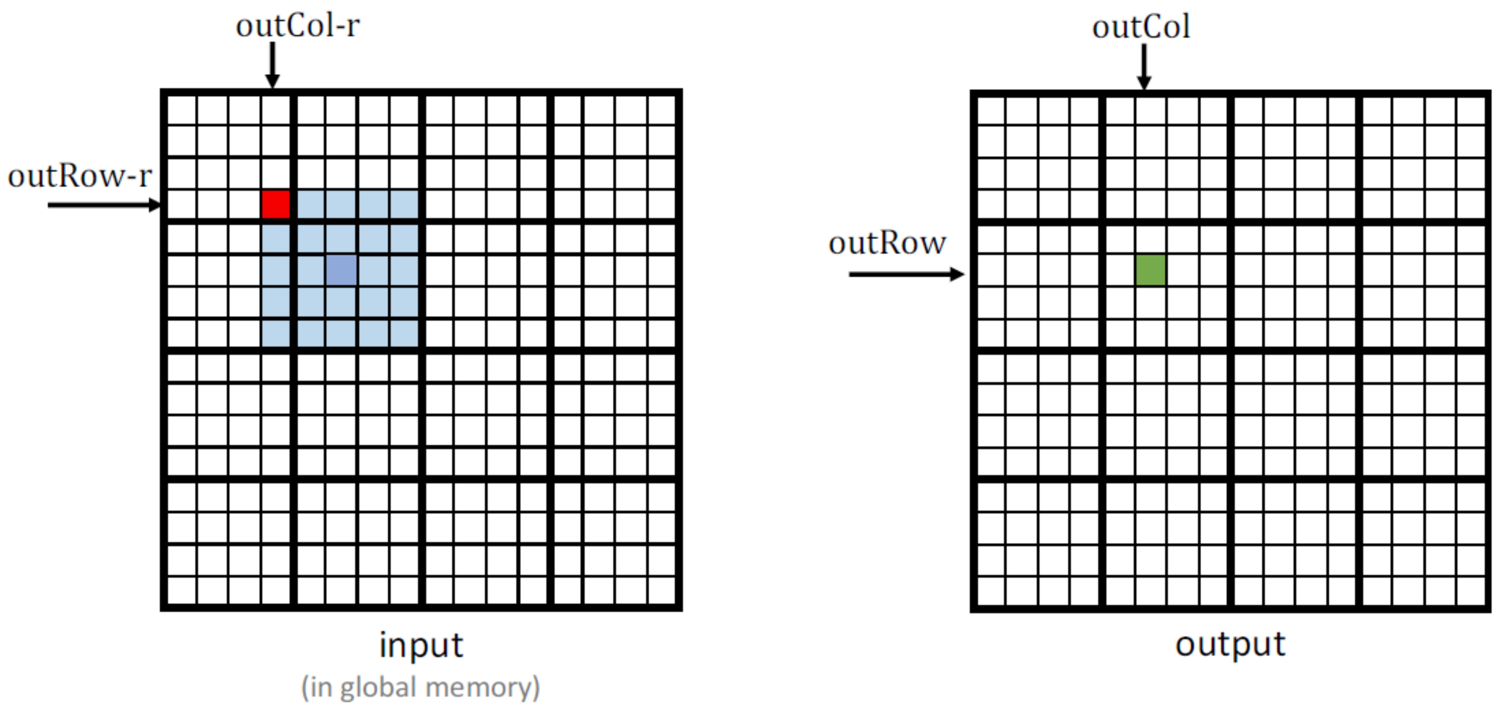

FIGURE 7.6

Parallelization and thread organization for 2D convolution.

```
__global__ void convolution_2D_basic_kernel(float *N, float *F, float *P,
    int r, int width, int height) {
    int outCol = blockIdx.x*blockDim.x + threadIdx.x;
    int outRow = blockIdx.y*blockDim.y + threadIdx.y;
    float Pvalue = 0.0f;
    for (int fRow = 0; fRow < 2*r+1; fRow++) {
      for (int fCol = 0; fCol < 2*r+1; fCol++) {
          inRow = outRow - r + fRow;
          inCol = outCol - r + fCol;
          if (inRow >= 0 && inRow < height && inCol >= 0 && inCol < width) {
             Pvalue += F[fRow][fCol]*N[inRow*width + inCol];
          }
        }
    }
    P[outRow][outCol] = Pvalue;
}
```

FIGURE 7.7

A 2D convolution kernel with boundary condition handling.

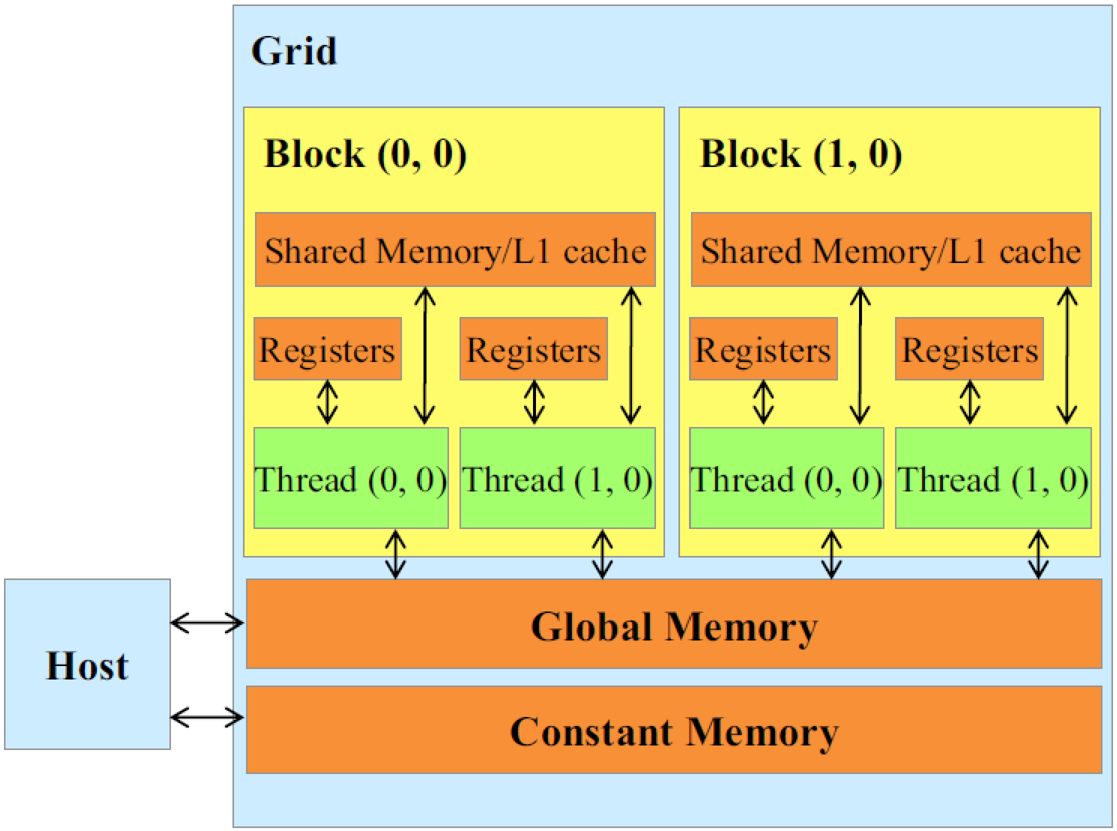

FIGURE 7.8

A review of the CUDA memory model.

```
__global__ void convolution_2D_const_mem_kernel(float *N, float *P, int r,
      int width, int height) {
    int outCol = blockIdx.x*blockDim.x + threadIdx.x;
    int outRow = blockIdx.y*blockDim.y + threadIdx.y;
    float Pvalue = 0.0f;
    for (int fRow = 0; fRow < 2*r+1; fRow++) {
      for (int fCol = 0; fCol < 2*r+1; fCol++) {
          inRow = outRow - r + fRow;
          inCol = outCol - r + fCol;
          if (inRow >= 0 && inRow < height && inCol >= 0 && inCol < width) {
             Pvalue += F[fRow][fCol]*N[inRow*width + inCol];
          }
        }
    }
    P[outRow*width+outCol] = Pvalue;
}
```

FIGURE 7.9

A 2D convolution kernel using constant memory for F.

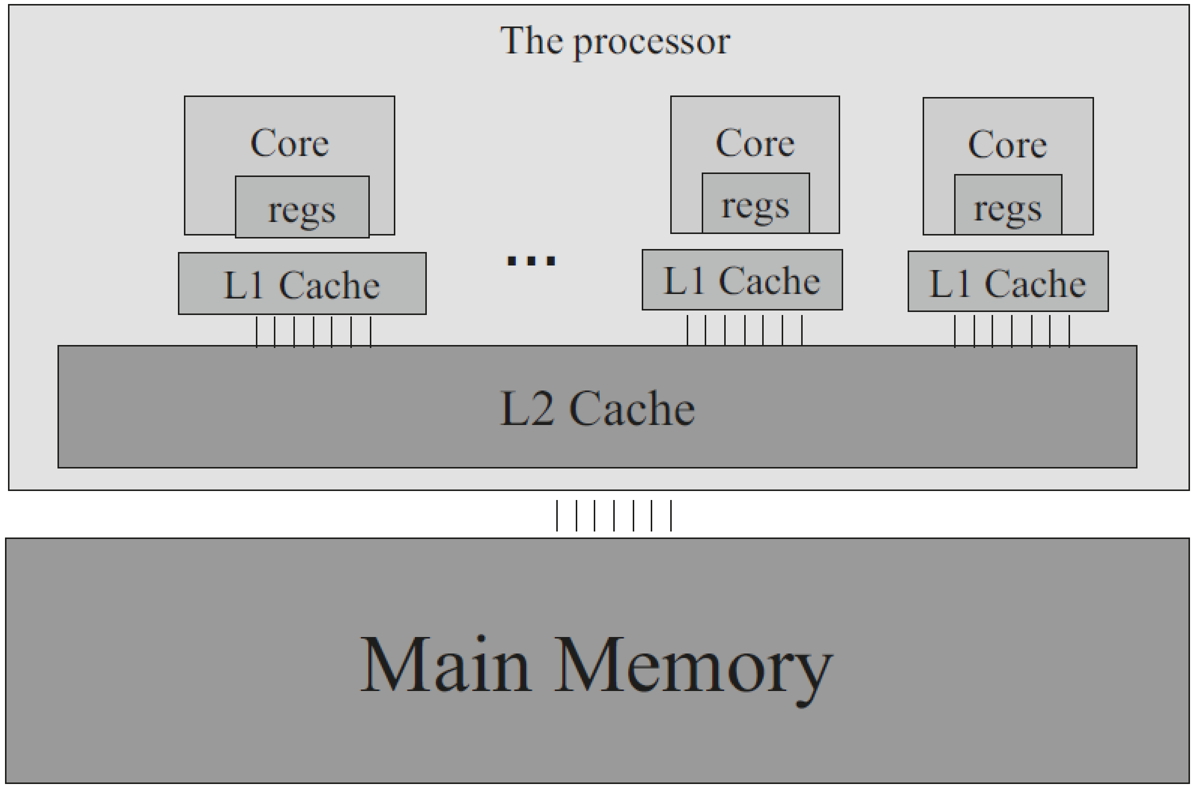

FIGURE 7.10

A simplified view of the cache hierarchy of modern processors.

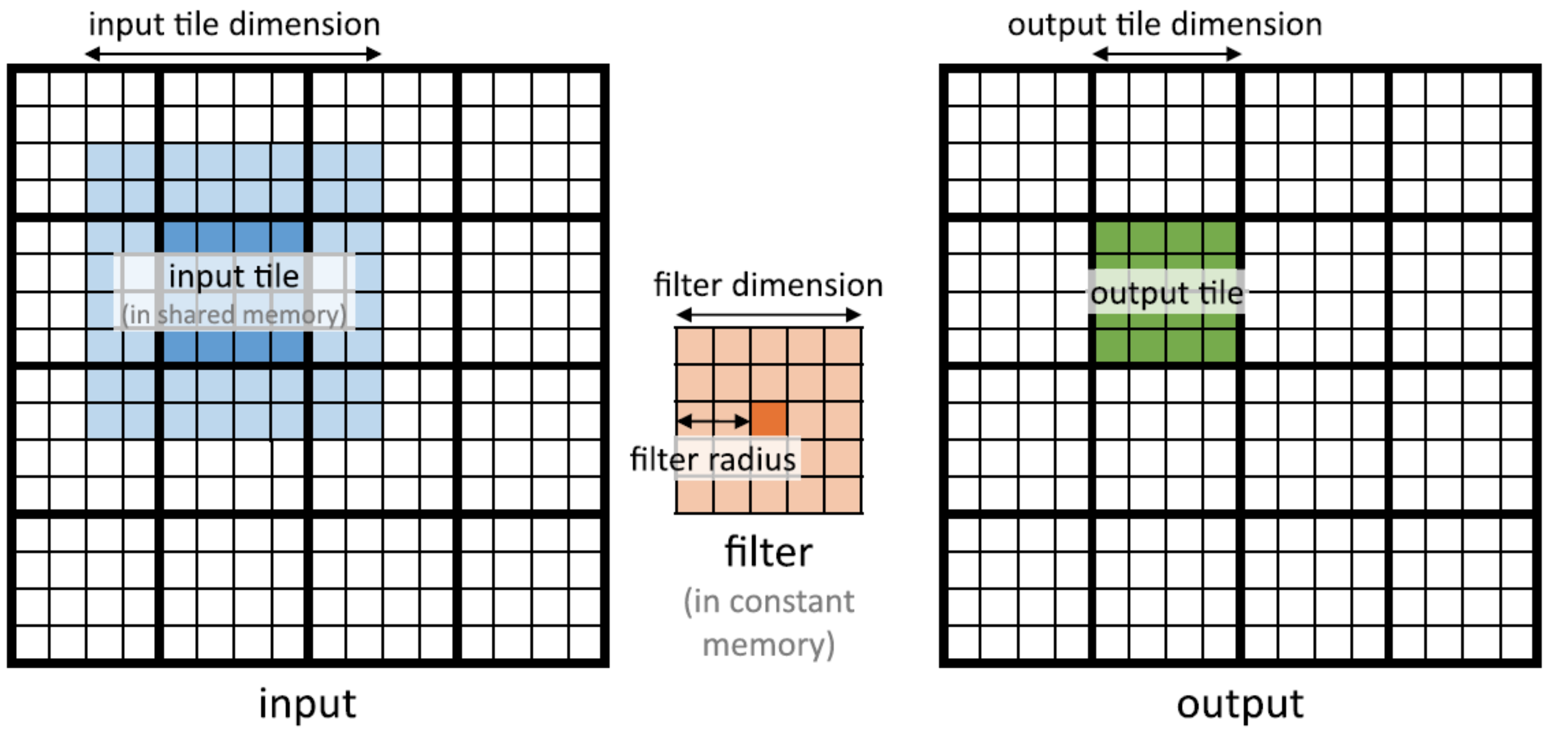

FIGURE 7.11

Input tile versus output tile in a 2D convolution.

```
#define IN_TILE_DIM 32
#define OUT_TILE_DIM ((IN_TILE_DIM) - 2*(FILTER_RADIUS))
__constant__ float F_c[2*FILTER_RADIUS+1][2*FILTER_RADIUS+1];
__global__ void convolution_tiled_2D_const_mem_kernel(float *N, float *P,
                                                     int width, int height) {
  int col = blockIdx.x*OUT_TILE_DIM + threadIdx.x - FILTER_RADIUS;
  int row = blockIdx.y*OUT_TILE_DIM + threadIdx.y - FILTER_RADIUS;
  //loading input tile
  __shared__ N_s[IN_TILE_DIM][IN_TILE_DIM];
  if(row>=0 && row<height && col>=0 && col<width) {
    N_s[threadIdx.y][threadIdx.x] = N[row*width + col];
  } else {
    N_s[threadIdx.y][threadIdx.x] = 0.0;
  }
  __syncthreads();
  // Calculating output elements
  int tileCol = threadIdx.x - FILTER_RADIUS;
  int tileRow = threadIdx.y - FILTER_RADIUS;
  // turning off the threads at the edges of the block
  if (col >= 0 && col < width && row >=0 && row < height) {
    if (tileCol>=0 && tileCol<OUT_TILE_DIM && tileRow>=0
                 && tileRow<OUT_TILE_DIM){
      float Pvalue = 0.0f;
      for (int fRow = 0; fRow < 2*FILTER_RADIUS+1; fRow++) {
        for (int fCol = 0; fCol < 2*FILTER_RADIUS+1; fCol++) {
          Pvalue += F[fRow][fCol]*N_s[tileRow+fRow][tileCol+fCol];
        }
      }
      P[row*width+col] = Pvalue;
    }
  }
}
```

FIGURE 7.12

A tiled 2D convolution kernel using constant memory for F.

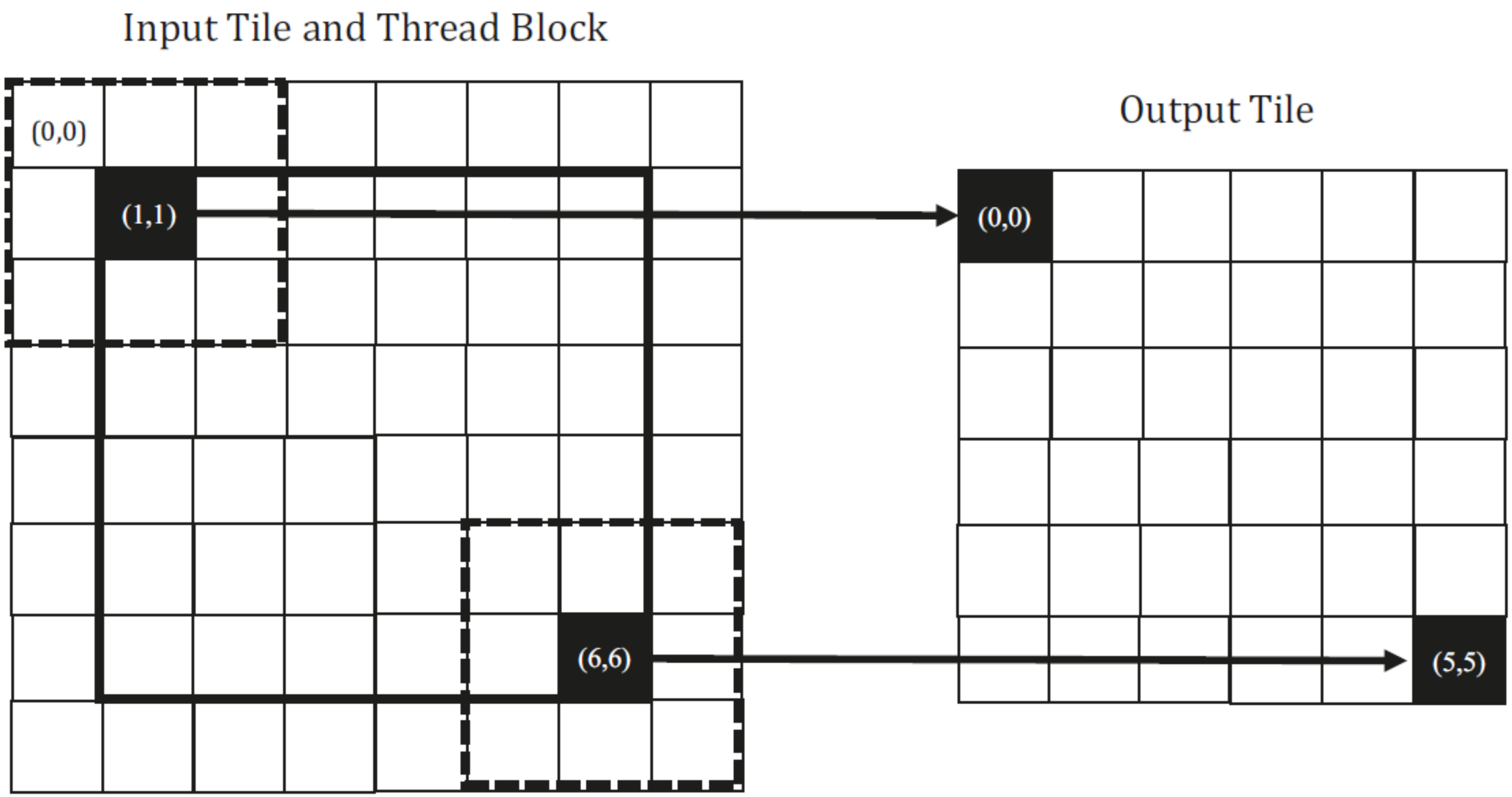

FIGURE 7.13

A small example that illustrates the thread organization for using the input tile elements in the shared memory to calculate the output tile elements.

IN_TILE_DIM		8	16	32	Bound
5x5 filter (FILTER_RADIUS = 2)	**OUT_TILE_DIM**	4	12	28	-
	Ratio	3.13	7.03	9.57	12.5
9x9 filter (FILTER_RADIUS = 4)	**OUT_TILE_DIM**	-	8	24	-
	Ratio	-	10.13	22.78	40.5

FIGURE 7.14

Arithmetic-to-global memory access ratio as a function of tile size and filter size for a 2D tiled convolution.

```
#define TILE_DIM 32
__constant__ float F_c[2*FILTER_RADIUS+1][2*FILTER_RADIUS+1];
__global__ void convolution_cached_tiled_2D_const_mem_kernel(float *N,
                                  float *P, int width, int height) {
  int col = blockIdx.x*TILE_DIM + threadIdx.x;
  int row = blockIdx.y*TILE_DIM + threadIdx.y;
  //loading input tile
  __shared__ N_s[TILE_DIM][TILE_DIM];
  if(row<height && col<width) {
    N_s[threadIdx.y][threadIdx.x] = N[row*width + col];
  } else {
    N_s[threadIdx.y][threadIdx.x] = 0.0;
  }
  __syncthreads();
  // Calculating output elements
  // turning off the threads at the edges of the block
  if (col < width && row < height) {
    float Pvalue = 0.0f;
    for (int fRow = 0; fRow < 2*FILTER_RADIUS+1; fRow++) {
      for (int fCol = 0; fCol < 2*FILTER_RADIUS+1; fCol++) {
        if (threadIdx.x-FILTER_RADIUS+fCol >= 0 &&
            threadIdx.x-FILTER_RADIUS+fCol < TILE_DIM &&
            threadIdx.y-FILTER_RADIUS+fRow >= 0 &&
            threadIdx.y-FILTER_RADIUS+fRow < TILE_DIM){
          Pvalue += F[fRow][fCol]*N_s[threadIdx.y+fRow][threadIdx.x+fCol];
        }
        else {
          if (row-FILTER_RADIUS+fRow >= 0 &&
              row-FILTER_RADIUS+fRow < height &&
              col-FILTER_RADIUS+fCol >=0 &&
              col-FILTER_RADIUS+fCol < width) {
            Pvalue += F[fRow][fCol]*
                                    N[(row-FILTER_RADIUS+fRow)*width+col-
FILTER_RADIUS+fCol];
          }
        }
      }
      P[row*width+col] = Pvalue;
    }
  }
}
```

FIGURE 7.15

A tiled 2D convolution kernel using caching for halos and constant memory for F.

$$
\begin{aligned}
P_{2,2} = {} & N_{0,0}*M_{0,0} + N_{0,1}*M_{0,1} + N_{0,2}*M_{0,2} + N_{0,3}*M_{0,3} + N_{0,4}*M_{0,4} \\
& + N_{1,0}*M_{1,0} + N_{1,1}*M_{1,1} + N_{1,2}*M_{1,2} + N_{1,3}*M_{1,3} + N_{1,4}*M_{1,4} \\
& + N_{2,0}*M_{2,0} + N_{2,1}*M_{1,1} + N_{2,2}*M_{2,2} + N_{2,3}*M_{2,3} + N_{2,4}*M_{2,4} \\
& + N_{3,0}*M_{3,0} + N_{3,1}*M_{3,1} + N_{3,2}*M_{3,2} + N_{3,3}*M_{3,3} + N_{3,4}*M_{3,4} \\
& + N_{4,0}*M_{4,0} + N_{4,1}*M_{4,1} + N_{4,2}*M_{4,2} + N_{4,3}*M_{4,3} + N_{4,4}*M_{4,4} \\
= {} & 1*1 + 2*2 + 3*3 + 4*2 + 5*1 \\
& + 2*2 + 3*3 + 4*4 + 5*3 + 6*2 \\
& + 3*3 + 4*4 + 5*5 + 6*4 + 7*3 \\
& + 4*2 + 5*3 + 6*4 + 7*3 + 8*2 \\
& + 5*1 + 6*2 + 7*3 + 8*2 + 5*1 \\
= {} & 1 + 4 + 9 + 8 + 5 \\
& + 4 + 9 + 16 + 15 + 12 \\
& + 9 + 16 + 25 + 24 + 21 \\
& + 8 + 15 + 24 + 21 + 16 \\
& + 5 + 12 + 21 + 16 + 5 \\
= {} & 321
\end{aligned}
$$

In-text figure 1

```
__global__ void
convolution_2D_basic_kernel(float *N, float *F, float *P, int r,
                            int width, int height) {
  // kernel body
}
```

In-text figure 2

```
#define FILTER_RADIUS 2
__constant__ float F[2*FILTER_RADIUS+1][2*FILTER_RADIUS+1];
```

In-text figure 3

```
cudaMemcpyToSymbol(F,F_h,(2*FILTER_RADIUS+1)*(2*FILTER_RADIUS+1)*sizeof(float));
```

In-text figure 4

```
cudaMemcpyToSymbol(dest, src, size)
```

In-text figure 5