

Chapter-19

Parallel programming with OpenACC

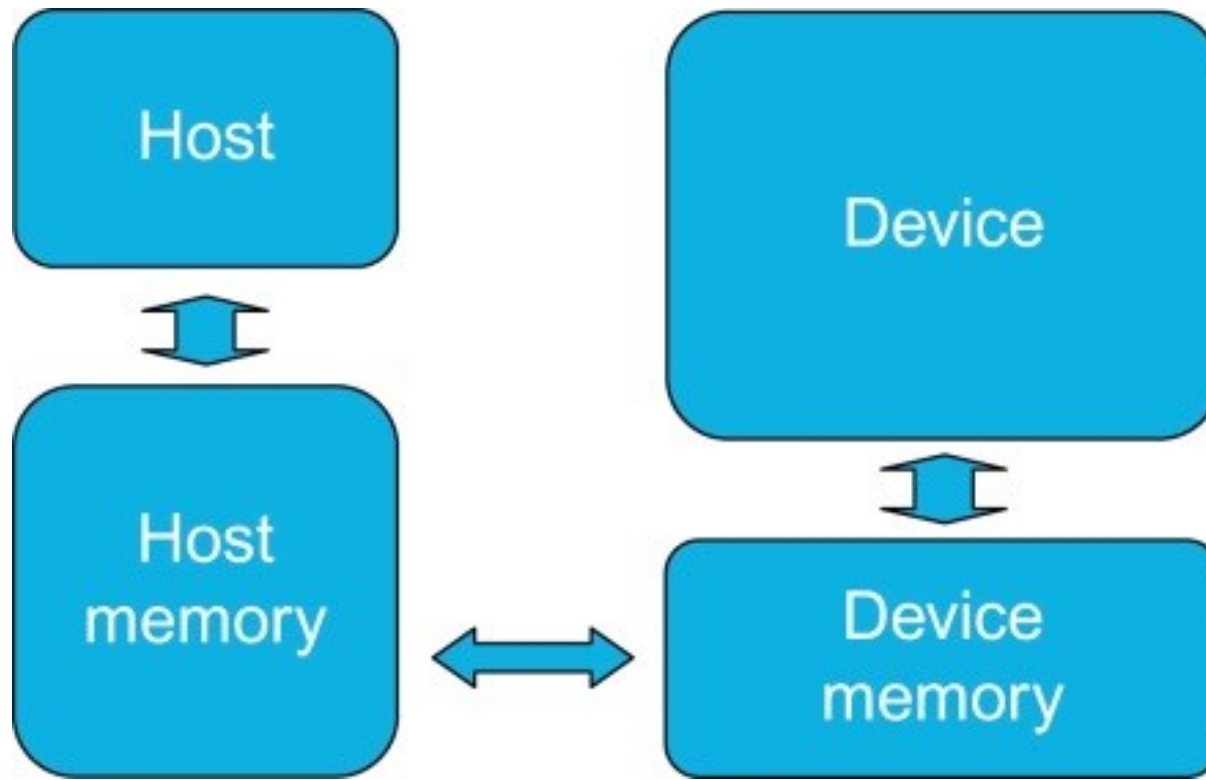


FIGURE 19.1: OpenACC abstract machine model.

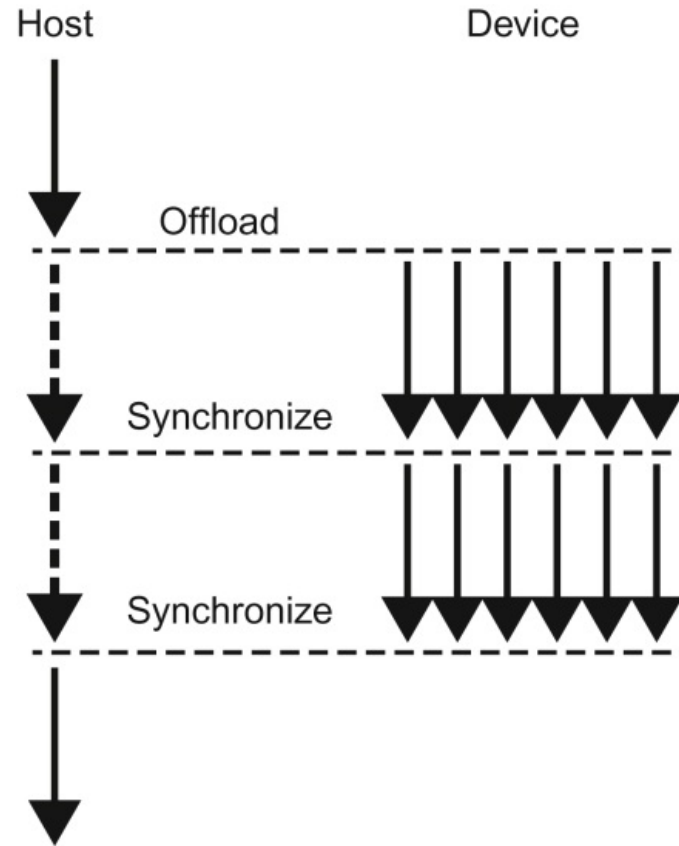
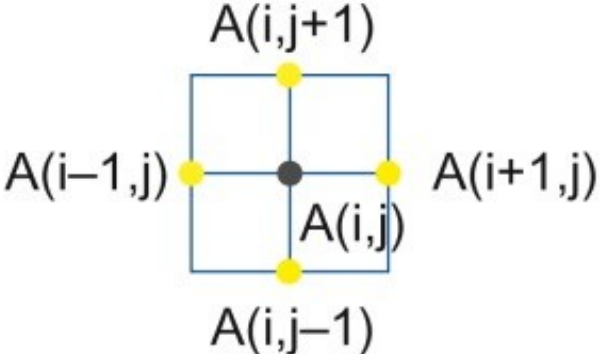


FIGURE 19.2: The OpenACC offloading execution model.

```
// C or C++  
#pragma acc <directive> <clauses>  
{ ... }  
  
! Fortran  
!$acc <directive> <clauses>  
...  
!$acc end <directive>
```

FIGURE 19.3: Basic format for OpenACC directives.



The diagram shows a central node labeled $A(i,j)$ represented by a black dot. It is surrounded by four nodes, each represented by a yellow dot: $A(i,j+1)$ is directly above, $A(i,j-1)$ is directly below, $A(i-1,j)$ is to the left, and $A(i+1,j)$ is to the right. These four nodes are connected to the central node by blue lines forming a cross shape.

$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

FIGURE 19.4: A Laplace equation example.

```

53.     while ( err > tol && iter < iter_max ) {
54.         err=0.0;
55.         for( int j = 1; j < n-1; j++) {
56.             for(int i = 1; i < m-1; i++) {
57.
58.                 Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
59.                                     A[j-1][i] + A[j+1][i]);
60.
61.                 err = max(err, abs(Anew[j][i] - A[j][i]));
62.             }
63.         }
64.         for( int j = 1; j < n-1; j++) {
65.             for( int i = 1; i < m-1; i++ ) {
66.                 A[j][i] = Anew[j][i];
67.             }
68.         }
69.         iter++;
70.     }

```

FIGURE 19.5: Jacobi Iterative Method example code.

```

53.     while ( err > tol && iter < iter_max ) {
54.         err=0.0;
55.         #pragma acc kernels
56.         {
57.             for( int j = 1; j < n-1; j++) {
58.                 for(int i = 1; i < m-1; i++) {
59.
60.                     Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
61.                                           A[j-1][i] + A[j+1][i]);
62.
63.                     err = max(err, abs(Anew[j][i] - A[j][i]));
64.                 }
65.             }
66.
67.             for( int j = 1; j < n-1; j++) {
68.                 for( int i = 1; i < m-1; i++ ) {
69.                     A[j][i] = Anew[j][i];
70.                 }
71.             }
72.         }
73.         iter++;
74.     }

```

FIGURE 19.6: Example code with OpenACC kernels directive.

```

$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
    40, Loop not fused: function call before adjacent loop
        Generated vector sse code for the loop
    51, Loop not vectorized/parallelized: potential early exits
    55, Generating copyout(Anew[1:4094][1:4094])
        Generating copyin(A[:, :])
        Generating copyout(A[1:4094][1:4094])
        Generating Tesla code
    57, Loop is parallelizable
    59, Loop is parallelizable
        Accelerator kernel generated
        57, #pragma acc loop gang /* blockIdx.y */
        59, #pragma acc loop gang, vector(128) /* blockIdx.x
threadIdx.x */
        63, Max reduction generated for error
    67, Loop is parallelizable
    69, Loop is parallelizable
        Accelerator kernel generated
        67, #pragma acc loop gang /* blockIdx.y */
        69, #pragma acc loop gang, vector(128) /* blockIdx.x
threadIdx.x */

```

FIGURE 19.7: Compiler output from example kernels code.

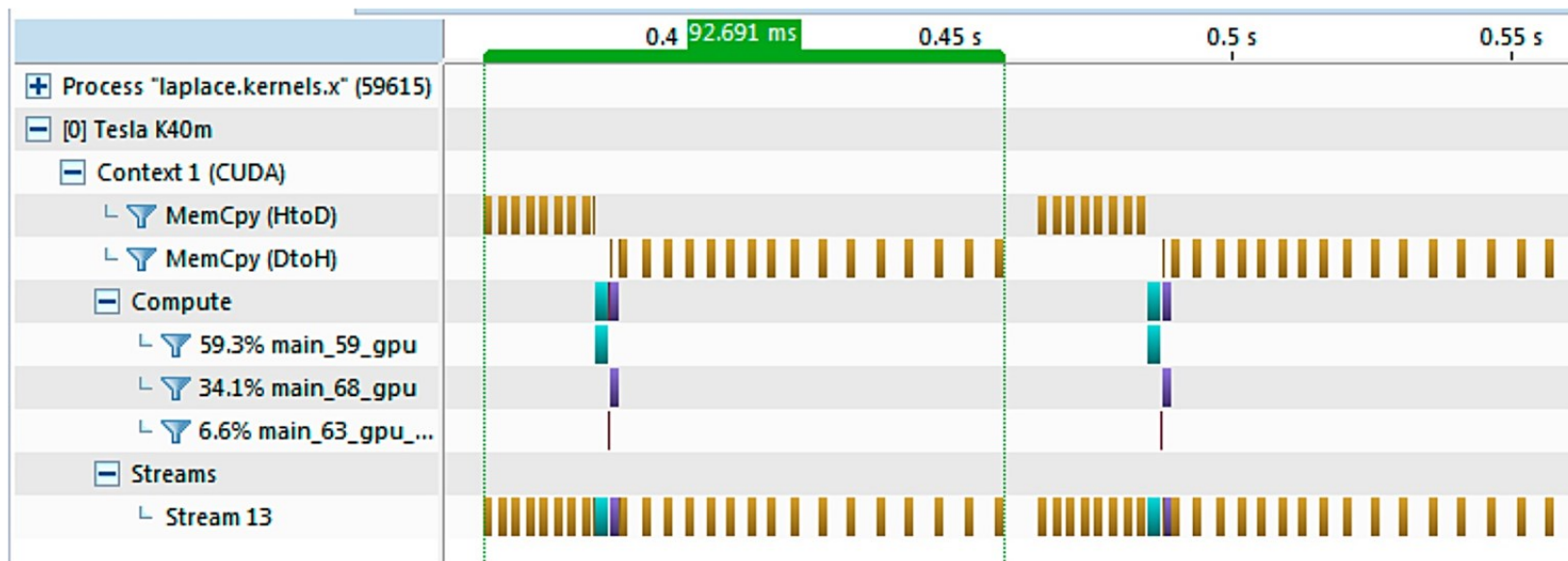


FIGURE 19.8: GPU timeline of kernels directive example code.

```

53.     while ( err > tol &&  iter < iter_max ) {
54.         err=0.0;
55.         #pragma acc parallel loop reduction(max:err) collapse(2)
56.         for( int j = 1; j < n -1; j++) {
57.             for(int i = 1; i < m -1; i++) {
58.
59.                 Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
60.                                     A[j-1][i] + A[j+1][i]);
61.
62.                 err = max(err, abs(Anew[j][i] - A[j][i]));
63.             }
64.         }
65.         #pragma acc parallel loop collapse(2)
66.         for( int j = 1; j < n-1; j++) {
67.             for( int i = 1; i < m-1; i++ ) {
68.                 A[j][i] = Anew[j][i];
69.             }
70.         }
71.     }
72.
73.     iter++;
74. }

```

FIGURE 19.9: Jacobi Iterative Method code using parallel directive.

```

$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
    41, Loop not fused: function call before adjacent loop
        Loop not vectorized: may not be beneficial
        Unrolled inner loop 4 times
        Generated 3 prefetches in scalar loop
    52, Loop not vectorized/parallelized: potential early exits
    56, Accelerator kernel generated
        Generating Tesla code
        56, Generating reduction(max:error)
        57, #pragma acc loop gang, vector(128) collapse(2) /*
blockIdx.x threadIdx.x */
        59, /* blockIdx.x threadIdx.x collapsed */
    56, Generating copyout(Anew[1:4094][1:4094])
        Generating copyin(A[:, :])
    67, Accelerator kernel generated
        Generating Tesla code
        68, #pragma acc loop gang, vector(128) collapse(2) /*
blockIdx.x threadIdx.x */
        70, /* blockIdx.x threadIdx.x collapsed */
    67, Generating copyin(Anew[1:4094][1:4094])
        Generating copyout(A[1:4094][1:4094])

```

FIGURE 19.10: Compiler feedback for Jacobi Iterative Method using parallel directive.

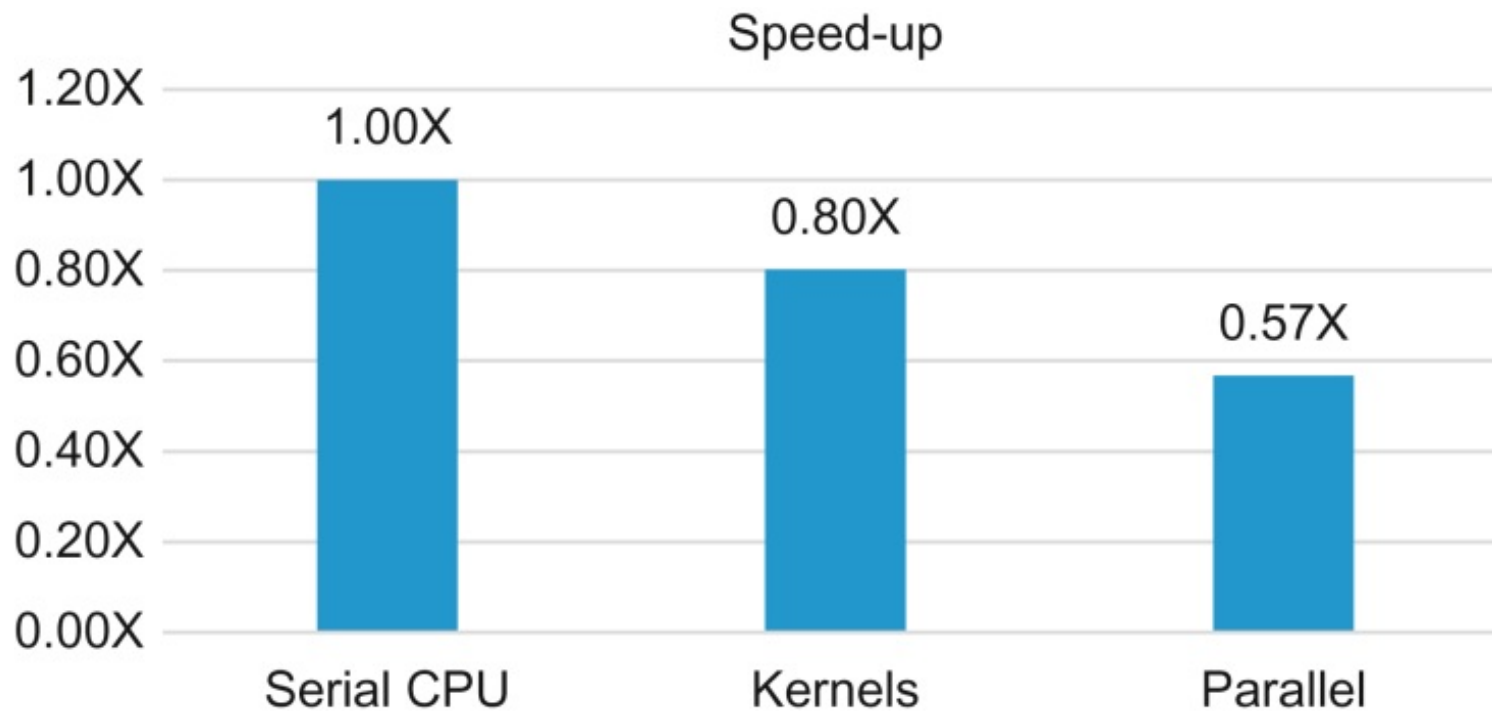


FIGURE 19.11: Performance speed-up from OpenACC kernels and parallel (higher is better).

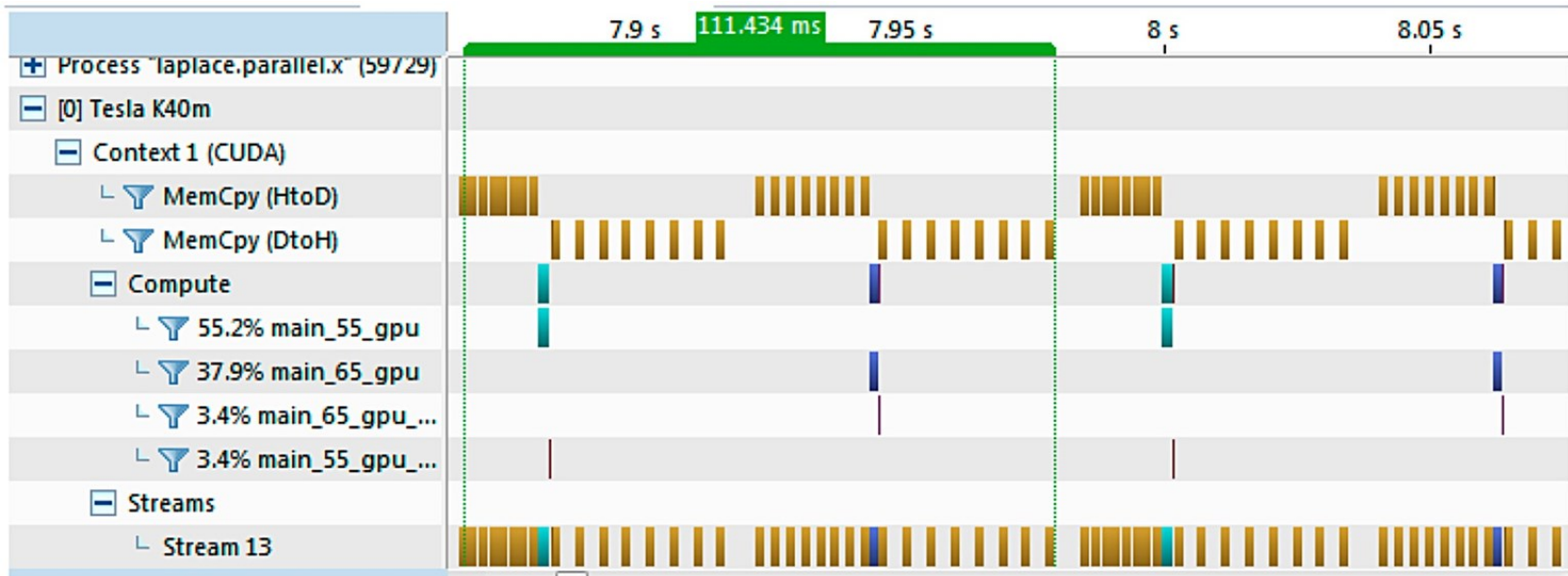


FIGURE 19.12: GPU timeline of parallel loop code.

Create	Allocate space for the listed variable on the accelerator device at the beginning of the region and delete the space at the end.
Copyin	Create the listed variables on the device, then copy the values of that variable into the device variable at the beginning of the region. The space will be deleted at the end of the region.
Copyout	Create the listed variables on the device, then copy the values of that variable from the device variable at the end of the region. The space will be deleted at the end of the region.
Copy	Behaves like a combined copyin and copyout.
Present	Declares that the variables can be assumed to already exist on the device, so no allocation, deletion, or data movement is necessary.

FIGURE 19.13: Five common data clauses and their meanings.

C/C++	<code>clause(start:count)</code> , start may be excluded if starting at 0
Fortran	<code>clause(start:end)</code> , start or end maybe excluded if they are the beginning or end of the array

FIGURE 19.14: Data clause array size notation.

```

53.  #pragma acc data create(Anew[:n][:m]) copy(A[:n][:m])
54.  while ( err > tol && iter < iter_max ) {
55.      err=0.0;
56.      #pragma acc parallel loop reduction(max:error) collapse(2)
57.          for( int j = 1; j < n-1; j++) {
58.              for(int i = 1; i < m-1; i++) {
59.
60.                  Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1]+
61.                                     A[j-1][i] + A[j+1][i]);
62.
63.                  err = max(err, abs(Anew[j][i] - A[j][i]));
64.              }
65.          }
66.      #pragma acc parallel loop collapse(2)
67.          for( int j = 1; j < n-1; j++) {
68.              for( int i = 1; i < m-1; i++ ) {
69.                  A[j][i] = Anew[j][i];
70.              }
71.          }
72.
73.      iter++;
74.  }

```

FIGURE 19.15: Jacobi Iterative Method with data region.

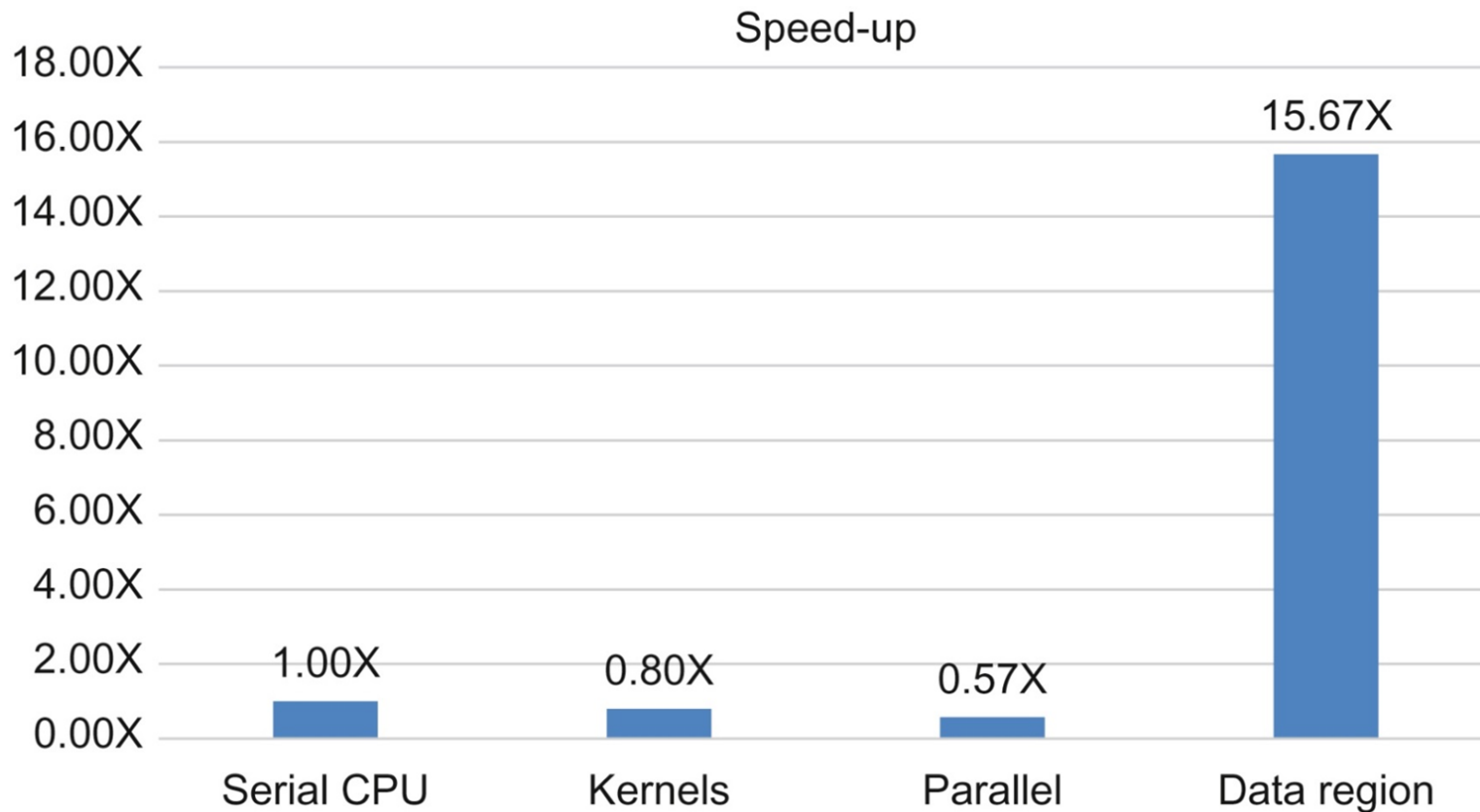


FIGURE 19.16: Speed-up with addition of data directive (higher is better).

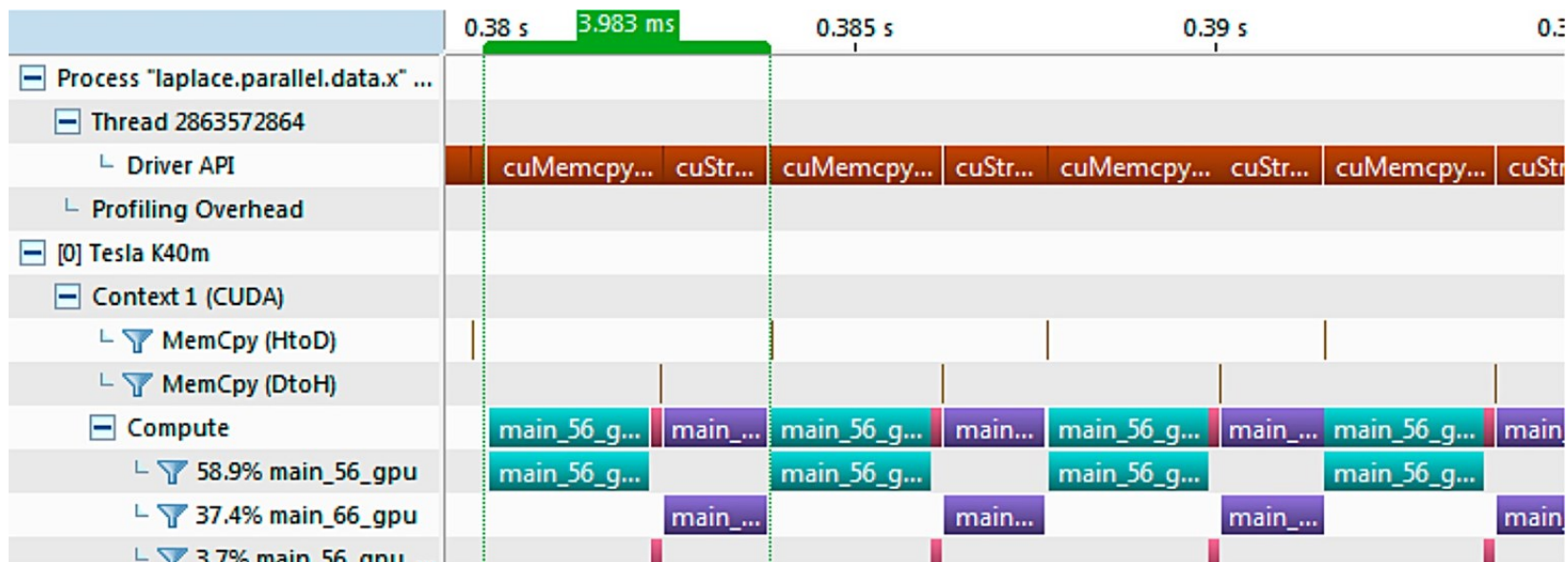


FIGURE 19.17: GPU timeline after optimizing data motion.

```

1.  template <class ctype> class Data
2.  {
3.      private:
4.      /// Length of the data array
5.          int len ;
6.      /// Data array
7.          ctype *arr;
8.
9.      public:
10.         /// Class constructor
11.         Data(int length)
12.         {
13.             len = length;
14.             arr = new ctype[len];
15.             #pragma acc enter data create(arr[0:len])
16.         }
17.         /// Class destructor
18.         ~Data()
19.         {
20.             #pragma acc exit data delete(arr)
21.             delete arr;
22.             len = 0;
23.         }
24.     }

```

FIGURE 19.18: Example of unstructured data directives in C++ class.

Enter data	Create	Allocate space for the listed variable on the accelerator, but do not initiate any data transfer. Increments reference count.
	Copyin	Create the listed variables on the device, then copy the values of that variable into the device variable. Increments reference count.
Exit data	Copyout	Copy the values of that variable from the device variable and delete the device copy. Decrements reference count.
	Delete	Immediately set the reference count to 0 and remove the device copy of the variable without any data transfer.
	Release	Decrement the reference count for the variable and behave as a delete if the reference count is decremented to zero.

FIGURE 19.19: Data clauses for unstructured data directives.

```

1. #pragma acc update host(u_new[offset_first_row:m-2],u_new[offset_last_row:m-2])
2. MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE,
3.             t_nb, 0, u_new+offset_bottom_boundary, m-2,
4.             MPI_DOUBLE, b_nb, 0,
5.             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
6. MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE,
7.             b_nb, 1, u_new+offset_top_boundary, m-2,
8.             MPI_DOUBLE, t_nb, 1,
9.             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
10. #pragma acc update device(u_new[offset_top_boundary:m-2],u_new[offset_bottom_boundary:m-2])

```

FIGURE 19.20: Example of update directive with MPI halo exchange.

```
1.  #pragma acc parallel loop gang
2.  for (int l=0; l < N; l++)
3.  #pragma acc loop worker
4.      for (int k=0; k < N; k++ )
5.  #pragma acc loop seq
6.      for (int j=0; j < N; j++ )
7.  #pragma acc loop vector
8.          for (int i=0; i < N; i++)
9.              { ... }
```

FIGURE 19.21: Example of loop directive specifying levels of parallelism.

```
1. #pragma acc parallel loop gang num_gangs(1024) num_workers(32)
   vector_length(32)
2. for (int l=0; l < N; l++)
3. #pragma acc loop worker
4.     for (int k=0; k < N; k++ )
5. #pragma acc loop seq
6.         for (int j=0; j < N; j++ )
7. #pragma acc loop vector
8.             for (int i=0; i < N; i++)
9.                 { ... }
```

FIGURE 19.22: Adjusting loop parameters within a parallel region.

```
1. #pragma acc kernels loop gang(1024)
2.   for (int l=0; l < N; l++)
3.   #pragma acc loop worker(32)
4.     for (int k=0; k < N; k++ )
5.   #pragma acc loop seq
6.     for (int j=0; j < N; j++ )
7.   #pragma acc loop vector(32)
8.     for (int i=0; i < N; i++)
9.       { ... }
```

FIGURE 19.23: Adjusting loop parameters within a kernels region.


```

75.         while ( err > tol && iter < iter_max ) {
76.             err=0.0;
77.             #pragma acc parallel loop reduction(max:err)
device_type(nvidia) tile(32,4)
78.
79.             for( int j = 1; j < n-1; j++) {
80.                 for(int i = 1; i < m-1; i++) {
81.
82.                     Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
83.                                         A[j-1][i] + A[j+1][i]);
84.
85.                     err = max(err, abs(Anew[j][i] - A[j][i]));
86.                 }
87.             }
88.             #pragma acc parallel loop device_type(nvidia) tile(32,4)
89.             for( int j = 1; j < n-1; j++) {
90.                 for( int i = 1; i < m-1; i++ ) {
91.                     A[j][i] = Anew[j][i];
92.                 }
93.             }
94.         }
95.
96.         iter++;
97.     }

```

FIGURE 19.24: Jacobi Iterative Method code with loop tile clause.

```
1. #pragma acc routine seq  
2. unsigned char mandelbrot(int Px, int Py);}
```

FIGURE 19.25: Example of the routine directive.

```
1. #pragma acc data create(A[N])
2. {
3. #pragma acc parallel loop async
4. for (int i=0; i<N; i++) A[i] = 1;
5. #pragma acc update host(A[:N]) async
6. for (int j=0; j<N; j++) B[j] = 2;
7. #pragma acc wait
8. for (int k=0; k<N; k++) C[k] = A[k] + B[k];
9. }
```

FIGURE 19.26: Example of async and wait.

```

1. #pragma acc data create(A[WIDTH*HEIGHT])
2. for(int block = 0; block < num_blocks; block++ ) {
3.   int start = block * (HEIGHT/num_blocks),
a.   end      = start + (HEIGHT/num_blocks);
4. #pragma acc update
   device(A[block*block_size:block_size]) async(block%3)
5. #pragma acc parallel loop async(block%3)
6.   for(int y=start;y<end;y++) {
a.     for(int x=0;x<WIDTH;x++) {
b.       A[y*WIDTH+x]=x*y;
7.     }
8.   }
9. #pragma acc update
   self(A[block*block_size:block_size]) async(block%3)
10.  }
11. #pragma acc wait

```

FIGURE 19.27: Example of pipelining with async and wait.

CUDA	OpenACC
Grid	Gangs
Threadblock	Gang
Thread	Worker or vector lane
Warp	Vector
Threadblock size	Number of workers * vector length
Shared memory	Cache
Stream	Asynchronous work queue
CUDA memcpy	Update

FIGURE 19.28: Table of CUDA and OpenACC terminology.

```

1.  #pragma acc data create(x[0:n]) copyout(y[0:n])
2.  {
3.      #pragma acc kernels
4.      {
5.          for( i = 0; i < n; i++)
6.          {
7.              x[i] = 1.0f;
8.              y[i] = 0.0f;
9.          }
10.     }
11.
12.     #pragma acc host_data use_device(x,y)
13.     {
14.         cublasSaxpy(n, 2.0, x, 1, y, 1);
15.     }
16. }

```

FIGURE 19.29: Example using `host_data` with NVIDIA cuBLAS.

```
1. void saxpy(int n, float a, float * restrict x, float * restrict y)
2. {
3.     #pragma acc kernels deviceptr(x,y)
4.     {
5.         for(int i=0; i<n; i++)
6.         {
7.             y[i] += a*x[i];
8.         }
9.     }
10. }
```

FIGURE 19.30: Example of device ptr clause.

```
1.    // Declaration from header file
2.    #pragma acc routine seq
3.    extern "C" float saxpy_dev(float, float, float);
4.
5.    // Implementation from source file.
6.    extern "C"
7.    __device__
8.    float saxpy_dev(float a, float x, float y)
9.    {
10.        return a * x + y;
11.    }
```

FIGURE 19.31: Example using OpenACC routine directive with CUDA device kernel.


```

1.      #pragma acc data create(x[0:n]) copyout(y[0:n])
2.      {
3.          #pragma acc kernels
4.          {
5.              for( i = 0; i < n; i++)
6.              {
7.                  x[i] = 1.0f;
8.                  y[i] = 0.0f;
9.              }
10.         }
11.
12.     #pragma acc parallel loop
13.         for( i = 0; i < n; i++ )
14.         {
15.             y[i] = saxpy_dev(2.0, x[i], y[i]);
16.         }
17.     }

```

FIGURE 19.32: Example calling CUDA device kernel from OpenACC.