Computational Science on Many-Core Architectures Exercise 5

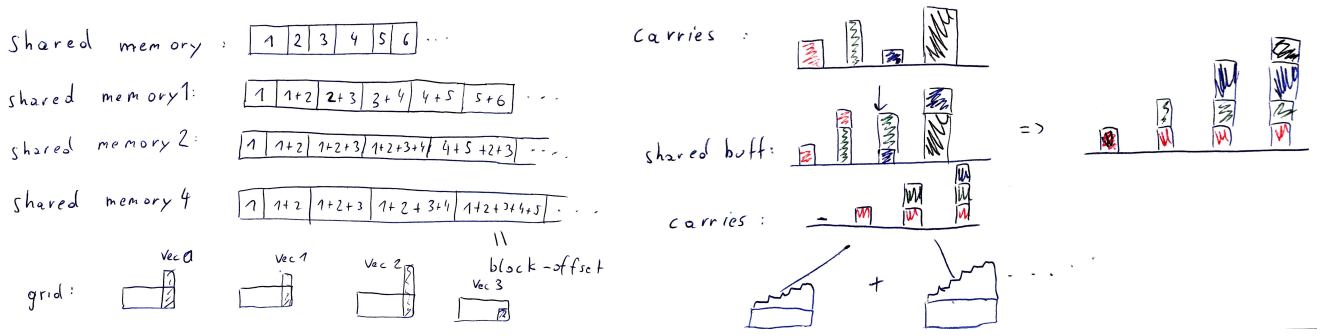
Example 1 Inclusive and Exclusive Scan (4 Points)

a)

```
10  __global__ void scan_kernel_1(double const *X,
11                                 double *Y,
12                                 int N,
13                                 double *carries)
14  {
15    __shared__ double shared_buffer[256];
16    double my_value;
17
18    unsigned int work_per_thread = (N - 1) / (gridDim.x * blockDim.x) + 1;
19    unsigned int block_start = work_per_thread * blockDim.x *  blockIdx.x;
20    unsigned int block_stop  = work_per_thread * blockDim.x * (blockIdx.x + 1);
21    unsigned int block_offset = 0;
22
23    // run scan on each section
24    for (unsigned int i = block_start + threadIdx.x; i < block_stop; i += blockDim.x)
25    {
26      // load data:
27      my_value = (i < N) ? X[i] : 0;
28
29      // inclusive scan in shared buffer:
30      for(unsigned int stride = 1; stride < blockDim.x; stride *= 2)
31      {
32        __syncthreads();
33        shared_buffer[threadIdx.x] = my_value;
34        __syncthreads();
35        if (threadIdx.x >= stride)
36          my_value += shared_buffer[threadIdx.x - stride];
37      }
38      __syncthreads();
39      shared_buffer[threadIdx.x] = my_value;
40      __syncthreads();
41
42      // exclusive scan requires us to write a zero value at the beginning of each block
43      my_value = (threadIdx.x > 0) ? shared_buffer[threadIdx.x - 1] : 0;
44
45      // write to output array
46      if (i < N)
47        Y[i] = block_offset + my_value;
48
49      block_offset += shared_buffer[blockDim.x-1];
50    }
51
52    // write carry:
53    if (threadIdx.x == 0)
54      carries[blockIdx.x] = block_offset;
55  }
```

For the first kernel "kernel_1" and simulate it with 4 blocks and 6 threads per block.
Begin with line 24: the for loop iterates over the values X which belong to the block.
At the end of the for loop → write the temperaryy result of the scan into a vector Ya nd the offset stored in block_offset. At the end of the kernel every block containes ist scannes value and the vector for the next step.

```
58  // exclusive-scan of carries
59  __global__ void scan_kernel_2(double *carries)
60  {
61    __shared__ double shared_buffer[256];
62
63    // load data:
64    double my_carry = carries[threadIdx.x];
65
66    // exclusive scan in shared buffer:
67
68    for(unsigned int stride = 1; stride < blockDim.x; stride *= 2)
69    {
70      __syncthreads();
71      shared_buffer[threadIdx.x] = my_carry;
72      __syncthreads();
73      if (threadIdx.x >= stride)
74        my_carry += shared_buffer[threadIdx.x - stride];
75    }
76    __syncthreads();
77    shared_buffer[threadIdx.x] = my_carry;
78    __syncthreads();
79
80    // write to output array
81    carries[threadIdx.x] = (threadIdx.x > 0) ? shared_buffer[threadIdx.x - 1] : 0;
82  }
83
84  __global__ void scan_kernel_3(double *Y, int N,
85                                double const *carries)
86  {
87    unsigned int work_per_thread = (N - 1) / (gridDim.x * blockDim.x) + 1;
88    unsigned int block_start = work_per_thread * blockDim.x *  blockIdx.x;
89    unsigned int block_stop  = work_per_thread * blockDim.x * (blockIdx.x + 1);
90
91    __shared__ double shared_offset;
92
93    if (threadIdx.x == 0)
94      shared_offset = carries[blockIdx.x];
95
96    __syncthreads();
97
98    // add offset to each element in the block:
99    for (unsigned int i = block_start + threadIdx.x; i < block_stop; i += blockDim.x)
100     if (i < N)
101       Y[i] += shared_offset;
102 }
```



b)

Listing 1: kernel for inclusive_scan)

```
1  __global__ void makeInclusive(double *Y, int N, const double *X)
2  {
3      for (int i = blockDim.x * blockIdx.x + threadIdx.x; i < N-1; i += gridDim.x
4      {
5          Y[i] = Y[i+1];
6      }
7      if (blockDim.x * blockIdx.x + threadIdx.x == 0)
8          // First step: Scan within each thread group and write carries
9      scan_kernel_1<<<num_blocks, threads_per_block>>>(input, output, N, carries);
10
11     // Second step: Compute offset for each thread group (exclusive scan for eac
12     scan_kernel_2<<<1, num_blocks>>>(carries);
13
14     // Third step: Offset each thread group accordingly
15     scan_kernel_3<<<num_blocks, threads_per_block>>>(output, N, carries);
16
17     // Make inclusive
18     makeInclusive<<<num_blocks, threads_per_block>>>(output, N, input);
19
20     cudaFree(carries);
21     }{
22         Y[N-1] += X[N-1];
23     }
24 }
25
26 void exclusive_scan(double const * input, double* output, int N)
27 {
28     int num_blocks = 256;
29     int threads_per_block = 256;
30
31     double *carries;
32     cudaMalloc(&carries, sizeof(double) * num_blocks);
33
34     // First step: Scan within each thread group and write carries
35     scan_kernel_1<<<num_blocks, threads_per_block>>>(input, output, N, carries);
36
```

```
37      // Second step: Compute offset for each thread group (exclusive scan for eac
38      scan_kernel_2<<<1, num_blocks>>>(carries);
39
40      // Third step: Offset each thread group accordingly
41      scan_kernel_3<<<num_blocks, threads_per_block>>>(output, N, carries);
42
43      // Make inclusive
44      makeInclusive<<<num_blocks, threads_per_block>>>(output, N, input);
45
46      cudaFree(carries);
47  }
```

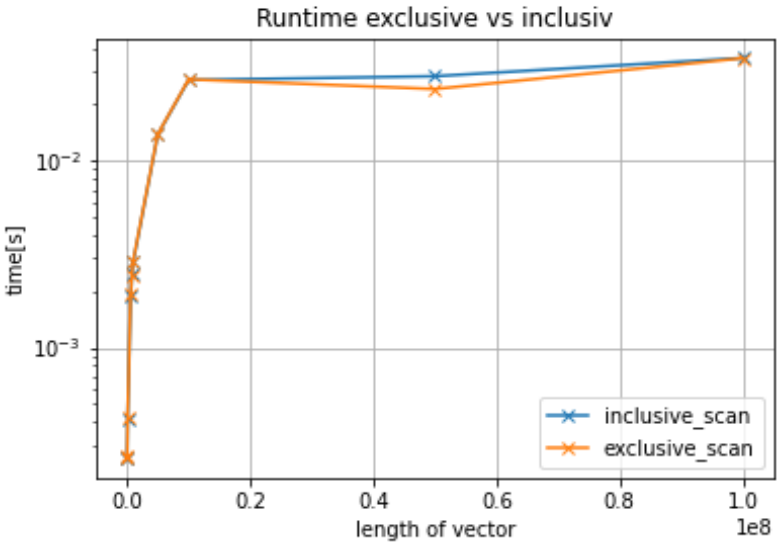
c)

Only nessesary to remove the folloing code snipped:

Listing 2: kernel for inclusive_scan)

```
1  // exclusive scan requires us to write a zero value at the beginning of each blo
2  my_value = (threadIdx.x > 0) ? shared_buffer[threadIdx.x - 1] : 0;
```

d)



There are bassicaly no differences.

2 Poisson equation (5 Points)