9 BONUS: GPU Programming and Performance Analysis [75 points]

The following two program segments are executed on a GPU with C compute units. In each compute unit, one or more thread-blocks can run. Each thread-block is composed of threads that are grouped into warps of W threads.

In both programs, 2D thread-blocks are used. Each thread-block is identified by its block indices (bx, by), and each thread is identified by its thread indices (tx, ty). The size of a thread-block is bdx * bdy. Consider that a thread-block is decomposed into warps in a way that threads with consecutive tx and equal ty belong to the same warp. More specifically, the warp number for a thread (tx, ty) is $\frac{\texttt{ty*bdx+tx}}{\texttt{warp-size}}$.

The entire input size is rows * cols integers. The size of an integer element is 4 bytes. The input is divided into tiles that are assigned to the thread-blocks.

local_data is an array in *local memory*, a fast on-chip memory that is used as a software-programmable cache. The amount of local memory per compute unit is S bytes. The threads of a thread-block can load data from *global memory* (i.e., the GPU off-chip memory) into local memory. The size of a global memory transaction is equal to the warp size times 4 bytes.

Program A:

```
__gpu_kernel_a(int* data, int rows, int cols){

  int* local_data[bdx * bdy];

  const int g_row = by * bdy + ty;
  const int g_col = bx * bdx + tx;
  const int l_row = ty;
  const int l_col = tx;
  const int pos   = g_row * cols + g_col;

  local_data[l_row * bdx + l_col] = data[pos];

  // Compute using local_data
}
```

Program B:

```
__gpu_kernel_b(int* data, int rows, int cols){

  int* local_data[bdx * bdy];

  const int g_row = bx * bdx + tx;
  const int g_col = by * bdy + ty;
  const int l_row = tx;
  const int l_col = ty;
  const int pos   = g_row * cols + g_col;

  local_data[l_row * bdy + l_col] = data[pos];

  // Compute using local_data
}
```

Please answer the questions on the next page.

(a) [15 points] What is the maximum number of thread-blocks that run in *each* compute unit for programs A and B?

$\lfloor \frac{S}{bdx \times bdy \times 4} \rfloor$.

Explanation. Given that each thread loads *one* integer value into local memory, the amount of local memory needed per thread-block is $bdx \times bdy \times 4$ (i.e., the number of integer elements of the array in local memory $bdx \times bdy$ times the size of an integer). Thus, the number of thread-blocks per compute unit is: $\lfloor \frac{S}{bdx \times bdy \times 4} \rfloor$.

(b) [15 points] Assuming that the GPU does *not* have caches, which program will execute faster? Why?

Program A.

Explanation. Program A will be faster, because it performs coalesced memory accesses (i.e., consecutive threads in the same warp access consecutive elements in memory), which ensure the minimum possible number of memory transactions.

(c) [15 points] Assume that the GPU has a single level of cache shared by all compute units. What will be the effect of this cache on the execution time of programs A and B?

Program B will be much faster than before (i.e., part (b)), while program A will not experience any improvement.

Explanation. There will be no significant change in the performance of program A, because the coalesced memory accesses already ensured the minimum possible number of memory transactions. However, program B will be much faster, because many accesses will hit the cache. For instance, if the threads with `ty` = 0 cause cache misses, the corresponding cache blocks will be loaded into the cache. Later accesses by the threads with `ty` = 1 will likely hit in the cache.

(d) [15 points] Assume that the access latency to the shared cache in part (c) is negligible. What should be the minimum size of the shared cache to guarantee that programs A and B have the same (or very similar) performance? (NOTE: The solution is independent of the warp scheduling policy).

$C \times \lfloor \frac{S}{bdx \times bdy \times 4} \rfloor \times (bdx \times bdy \times 4)$ bytes.

Explanation. Each thread-block loads one tile in local memory. Thus, the size of the shared cache per thread-block should be the size of the tile (`bdx * bdy`). Taking into account that there are $\lfloor \frac{S}{bdx \times bdy \times 4} \rfloor$ thread-blocks in each of the `C` compute units, the amount shared cache needed to keep all tiles in the cache is $C \times \lfloor \frac{S}{bdx \times bdy \times 4} \rfloor \times (bdx \times bdy \times 4)$ bytes.

(e) [15 points] Now assume that *only one* thread-block is executed in each compute unit. Each thread-block in program A needs always `T` ms to complete its work, because the computation is very regular. What will be the total execution time of program A?

$NumBatches \times T$ ms.

Explanation. The total number of thread-blocks is $NumBlocks = \lfloor \frac{rows \times cols}{bdx \times bdy} \rfloor$. The number of concurrent thread-blocks is $NumConcBlocks = C$. Thus, the total number of thread-blocks will be executed in a number of batches ($NumBatches$) that is equal to:

$NumBatches = \lceil \frac{NumBlocks}{NumConcBlocks} \rceil$

The total execution time is the $NumBatches \times T$ ms.