Universidad Zaragoza

Workshop 2 – RESCUER MSCA DOCTORAL NETWORK 2024-2028
Universidad de Zaragoza

GPU programming (I)

Task 1: Basic CUDA conversion, use of cuBLAS and loop flattening

Mario Morales Hernández mmorales@unizar.es

Module 4 – GPU programming (I)

Task 1 - Basic CUDA conversion, use of cuBLAS and loop flattening

The first task focuses on understanding how to convert a CPU-based shallow water equations solver to CUDA. The main challenge lies in transforming nested loops that process 2D grids into parallel computations suitable for GPU execution. We'll start with simpler functions like h_update_cells_2D and boundary conditions, which have straightforward parallel patterns. Then we will explore the library cuBLAS to convert h_compute_water_mass. Then, we'll learn loop flattening techniques essential for more complex functions like flux computations. This transformation is crucial as it forms the foundation for efficient GPU parallelization.

Key tasks:

1. Start with h_update_cells_2D:

    - Add CUDA compilation directives (#ifdef __CUDACC__)

    - Add __global__ keyword for CUDA kernels

    - Identify the loop structure

    - Implement thread indexing

    Help: use the macro

    #ifdef __CUDACC__

    ... GPU things

    #else

    ... CPU things

    #endif

    to keep the code clean and usable for the CPU and the GPU.

2. Create CUDA kernels for:

    - h_set_west_boundary

    - h_set_east_boundary

    - h_set_north_boundary

    - h_set_south_boundary

3. Modify swe2d.c to change the kernel calls for the five subroutines. You can use the same macro pattern for the call to keep the CPU and the GPU version. Please, pay attention to the kernel call and the appropriate grid/block configuration. Define a macro called nThreads for the blockSize.

4. Convert h_compute_water_mass. Please note that this subroutine implies a reduction collective operation, so it should be handled with an external library. We are going to use Cublas for that. Please look for the documentation and examples on the internet about the function cuBlasDaSum.

5. Flattening 2D loops involves transforming nested loops that iterate over a grid into a single loop using a linear index. This is essential for GPU parallelization, as CUDA threads are indexed linearly. The key is to map the 2D indices (i, j) (row and column) to a single linear index idx and vice versa.

Please, flatten 2D loops in CPU for:

- h_compute_x_fluxes
- h_compute_y_fluxes
- h_wet_dry_x
- h_wet_dry_y

Example:

Original nested loop:

```
for (int j = 0; j < nY; j++) {     // Rows
   for (int i = 0; i < nX; i++) { // Columns
      // Perform computations using i, j
   }
}
for (int idx = 0; idx < [....]; idx++) { // Single loop
   int j = [...] ;  // Row index
   int i = [...];  // Column index
   // Perform computations using i, j
}
```

Please, fill in the brackets. Practice index transformations on paper first. Verify then that index calculations preserve correct cell access. Please, leave the flattened version commented in the code. **Adapt the indexes accordingly for every loop (the don't iterate up to nX or nY).**