Supplementary information: GPU linear and non-linear Poisson-Boltzmann solver module for DelPhi

José Colmenares, Jesús Ortiz, and Walter Rocchia

Memory and thread organization

The grid partitioning in opposite parity logical structures is shown in Figure 1.

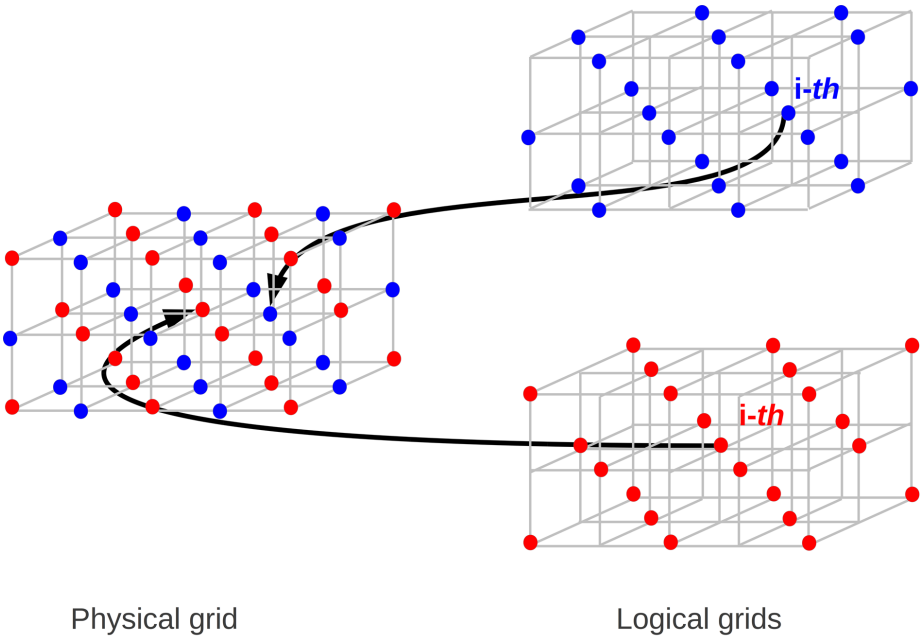

Figure 1: The grid points on the physical grid where the PBE is solved are separated in odd (red) and even (blue). This results in two logical sub-grids (on the right). A suitable indexing ensures that if each thread that is processing a red grid point in a slice loads the blue grid point having the same index, then all the needed nearest neighbors on that slice are loaded concurrently.

The linear indexing of the grid points follows: assuming (X, Y, Z) are the grid coordinates, and that the origin is in $(0, 0, 0)$, one can calculate the index in the linear array where the corresponding potential value is stored.

$$index = \left\lfloor \frac{X + Y \cdot N_x + Z \cdot N_x \cdot N_y}{2} \right\rfloor$$

where the lower brackets indicate the *floor* function and N_x and N_y are the dimensions of the grid along x and y directions, respectively.

The arrangement in threads, within the x-y plane, is sketched in Figure 2.

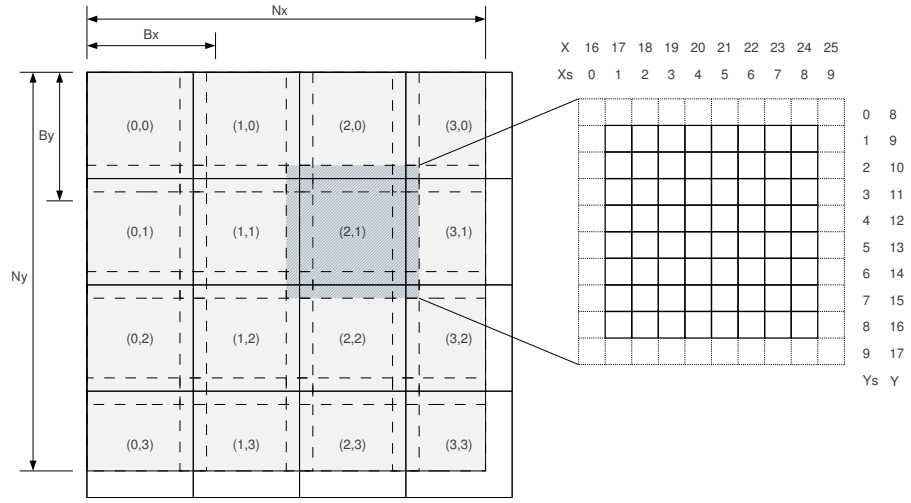

Figure 2: *Tartan* distribution of the blocks of threads in a generic x-y slice. Each thread has a fixed (X, Y) coordinate pair, and moves along the Z direction. B_x and B_y are the dimensions of the blocks. Dashed lines represent overlap between blocks, used to make available to the working threads all the needed neighboring points in the slice. X_s and Y_s are the relative coordinates of a thread within a block.

As one can see, the slice (shaded in gray) may not fit exactly in the blocks of threads, which must have all the same size. Due to this fact, there will likely be threads with an (X, Y) coordinate out of the grid (that is $X > N_x$ or $Y > N_y$). These threads will simply not perform any calculation. The overall performance is affected by the number of threads of this type, but its impact decreases in percentage with the size of the problem. Changing the block size would be of no help, since a suboptimal block size would in turn worsen the performance. As per performance, other relevant factors are the occupancy of the GPU, which requires bigger blocks, and the use of multiples of 32 threads, since the scheduler of the GPU allocates the threads in groups of 32, called warps. We found that a block size of 16x16 provides the best results.

A concise description of the adopted solving strategy follows:

Algorithm 1 Metalanguage for the Laplace stencil solution on a GPU

for each parity **do**
  Load the grid points of opposite parity into the texture memory
  for all $slice \in [1, nz-2]$ # *assuming x-y slices go from 0 to nz-1* **do**
    for each thread **do**
      - Based on the thread index, fetch the potential of the grid of opposite parity with the same index from the texture into shared memory
      if $slice = 1$ **then**
        - Based on the thread index, fetch the potential value of the "z-1" point, taken from the boundary conditions
      else
        - The potential that corresponds to the "z-1" is already available in the *temp* variable
      end if
      - Based on the thread index, fetch from the texture memory the potential that corresponds to the "z+1" grid point and store it in a *temp* variable
      - Update the current grid point value based on the Laplace stencil
    end for
  end for
end for