

Exercise: Conway's Game of Life with SYCL

Objective: Implement a parallel version of Conway's Game of Life using SYCL. You can choose to use either Unified Shared Memory (USM) or Buffers with Accessors, based on your preference and comfort level.

Tip:

- The grid is treated as a **toroidal (wrapping) grid** where edges connect to the opposite sides.
- Use modulo arithmetic to handle wrapping around the grid boundaries.
- For example, for cell $(0, 0)$ at the top-left corner, its neighboring cells are:

Neighboring Cells	Coordinates
Top-left	$(N-1, N-1)$
Top	$(N-1, 0)$
Top-right	$(N-1, 1)$
Left	$(0, N-1)$
Center	$(0, 0)$
Right	$(0, 1)$
Bottom-left	$(1, N-1)$
Bottom	$(1, 0)$
Bottom-right	$(1, 1)$

Task:

1. **Grid Initialization:**

- Write a function to initialize an $N \times N$ grid with random values (0 or 1).

2. **Printing the Grid:**

- Implement a function to display the grid for debugging or visualization.

3. **Neighbor Count:**

- Implement logic in a SYCL kernel to calculate the number of live neighbors for each cell. Use modulo arithmetic to handle wrapping.

4. **Game of Life Rules:**

- Apply the following rules to determine the state of each cell in the next generation:
 - A live cell (1) with fewer than 2 or more than 3 live neighbors dies.
 - A dead cell (0) with exactly 3 live neighbors becomes alive.
 - Otherwise, the cell remains in its current state.

5. **Debugging Mode:**

- Use a DEBUG flag to enable optional debugging. In this mode:
 - Store and print the neighbor count for each cell.

6. **Swapping Grids:**

- Use `std::swap` or equivalent buffer updates to alternate between the current and next-generation grids.

7. **Memory Management:**

- Depending on your choice:
 - If using **USM**, allocate memory with `malloc_shared` and free it at the end.
 - If using **Buffers**, create SYCL buffers and use accessors to manage data within kernels.

8. **Output:**

- Display the initial grid and all generations.

Optional:

- Add command-line arguments to customize the grid size and the number of generations.