# BFS Queueing

PUMPS 2015

## OBJECTIVE

The purpose of this lab is to understand hierarchical queuing in the context of the breadth first search algorithm as an example. You will implement a single iteration of breadth first search that takes a set of nodes in the current level (also called wave-front) as input and outputs the set of nodes belonging to the next level.

## INSTRUCTIONS

In the provided source code you will find a function named `s2g_cpu_scatter`. This function implements a simple scatter pattern on CPU. It loops over an input array, then for each input element it performs some computation (`outInvariant(...)`), loops over the output array, does some more computation (`outDependent(...)`), and accumulates to the output element.

- Edit the function `s2g_cpu_gather` to implement a gather version of `s2g_cpu_scatter` on CPU. Compile and test the code.
- Edit the kernel `s2g_gpu_scatter_kernel` to implement a scatter version of `s2g_cpu_scatter` on the GPU, and edit the function `s2g_gpu_scatter` to launch the kernel you implemented.
- Edit the kernel `s2g_gpu_gather_kernel` to implement a gather version of `s2g_cpu_gather` on the GPU, and edit the function `s2g_gpu_gather` to launch the kernel you implemented.

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

## QUESTIONS

- Comment on the relative behavior of each implementation.
- Explain why you think certain implementations perform better than others for various input combinations.

## SUGGESTIONS

- The system's autosave feature is not an excuse to not backup your code and answers to your questions regularly.
- If you have not done so already, read the tutorial
- Do not modify the template code provided – only insert code where the `//@@` demarcation is placed
- Develop your solution incrementally and test each version thoroughly before moving on to the next version
- Do not wait until the last minute to attempt the lab.
- If you get stuck with boundary conditions, grab a pen and paper. It is much easier to figure out the boundary conditions there.
- Implement the serial CPU version first, this will give you an understanding of the loops
- Get the first dataset working first. The datasets are ordered so the first one is the easiest to handle
- Make sure that your algorithm handles non-regular dimensional inputs (not square or multiples of 2). The slides may present the algorithm with nice inputs, since it minimizes the conditions. The datasets reflect different sizes of input that you are expected to handle
- Make sure that you test your program using all the datasets provided (the datasets can be selected using the dropdown next to the submission button)
- Check for errors: for example, when developing CUDA code, one can check for if the function call succeeded and print an error if not via the following macro:

```
#define wbCheck(stmt) do {                                                \
        cudaError_t err = stmt;                                           \
        if (err != cudaSuccess) {                                         \
            wbLog(ERROR, "Failed to run stmt ", #stmt);                   \
            wbLog(ERROR, "CUDA error ...  ", cudaGetErrorString(err));\
            return -1;                                                    \
        }                                                                 \
    } while(0)
```

An example usage is `wbCheck(cudaMalloc(...))`. A similar macro can be developed while programming OpenCL code.

## LOCAL DEVELOPMENT

While not required, the library used throughout the course can be downloaded from Github. The library does not depend on any external library (and should be cross platform), you can use `make` to generate the shared object file (further instructions are found on the Github page). Linking against the library would allow you to get similar behavior to the web interface

(minus the imposed limitations). Once linked against the library, you can launch your program as follows:

```
./program -e <expected_output_file> \
    -i <input_file_1>,<input_file_2> -o <output_file> -t <type>
```

The `<expected_output_file>` and `<input_file_n>` are the input and output files provided in the dataset. The `<output_file>` is the location you'd like to place the output from your program. The `<type>` is the output file type: `vector`, `matrix`, or `image`. If an MP does not expect an input or output, then pass `none` as the parameter.

In case of issues or suggestions with the library, please report them through the issue tracker on Github.