All exercises must be done by yourself. You may discuss questions and potential solutions with your classmates, but you may not look at their code. If in doubt, ask the instructor.

Acknowledge all sources you found useful.

Your code should compute the correct results.

Partial credit is available, so attempt all exercises.

Submit your answers as a PDF file.

Exercise 1

Parallelize the supplied `reduction.cpp` using the tree reduction technique discussed in class using threads (either C++ threads or pthreads).

For ten million numbers (`-n 10000000`), report the time for obtaining the sum using 1, 2, 4, 8, 16, and 32 threads from a run on BlueHive.

Your code should compile and run with the supplied `Makefile` on BlueHive 2.5. Make the required modifications to use either C++ threads or pthreads.

Exercise 2

Write a program that performs 2-D convolution on an image. The convolution matrix can be in 3x3 or 5x5 sizes. The image can be upto 4096x4096x24bpp (i.e. 3x8 bits per pixel).

Your program should accept a PPM image and produce an output in the PPM format. One advantage of PPM is that the file is a text format, so you can see its internals. One disadvantage is that the files are quite large.

I have enclosed a reference program `2dconv.cpp` that performs this task. See `README-2dconv.txt` for information on how to run this program.

Your goal is to speedup this program using autovectorization and multiple threads (either C++ threads or pthreads).

You can reuse parts of this program, but the program was written to be very slow so feel free to change the internals as you see fit (and it is highly recommended you do!).

Report the performance of the program as the time required to perform the convolution. Ignore file loading and writing time. If you transform data before or after convolving, report that time separately.

All performance numbers reported should be from BlueHive. Again, your code should compile and run using the enclosed Makefile modified to support pthreads/C++ threads.

1. Parallelize the code to use threads, name this `2dconv_threads.cpp`. Report the times for your program using 1, 2, 4, 8, 16 and 32 threads for each of the convolution masks supplied (edge1, edge2, sharpen). The masks id3 and id5 are identity masks, they don't change the image, and are useful for checking correctness using `diff` as noted in `README-2dconv.txt`.

2. Use autovectorize on `2dconv_threads.cpp` using `-O3` and `-mavx2` (compile it to `2dconv_vec1`). Report the times for 1, 2, 4, 8, 16, and 32 threads again, as in the previous question.

3. Modify `2dconv_threads.cpp` (saving it as `2dconv_vec2.cpp`), so that pixels at the borders are processed in a separate loop while non-border pixels are processed in a separate loop. You should be able to get rid of the conditional in the pixel processing loop that way. Now autovectorize again and report the times for 1, 2, 4, 8, 16 and 32 threads.

END.