BLOCK DOWNSAMPLING USING MODAL VALUES

Problem. Consider a d-dimensional array of size $2^{L_1} \times 2^{L_2} \times \cdots \times 2^{L_d}$, which will be called the “original image.” The $l-$downsampled image of size $2^{L_1-l} \times 2^{L_2-l} \times \cdots \times 2^{L_d-l}$ is defined by dividing the original image into blocks of size $2^l \times 2^l \times \cdots \times 2^l$, and replacing each block by a single pixel that is a most common value in that block, i.e., a mode of pixels in that block. If there is a tie, any of the modes is considered correct. Assume that the pixel values are unsigned integers. Write C++ code that outputs all l-downsamplings of the original image, where $l = 1, \ldots, \min\{L_1, \ldots, L_d\}$. The code should be multithreaded for maximum speed with a multicore processor. Make efficient use of RAM, which is large enough to contain the image and intermediate results.

Example 1. The $d = 2$, $L_1 = 2$, $L_2 = 3$ image

```
11111111
12121212
11222222
12222222
```

yields the 1-downsampled image:

```
1111
1222
```

and the 2-downsampled image:

```
12
```

Note that `11` is the wrong answer because mode-downsampling is based on blocks of the original image, rather than the previous downsampled image. The first value (1) of the 2-downsampled image is the mode of this block of the original image:

```
1111
1212
1122
1222
```

The second value (2) is the mode of this block of the original image:

```
1111
1212
2222
2222
```

Example 2. This case illustrates that any of the multiple modes is considered correct if there are ties for the most frequent value. The image

00000000
10101010
11001100
11101110
00001111
10101111
11101100
11001101

yields the 1-downsampled image

0000
1010
0011
1010

the 2-downsampled image

00
01

and the 3-downsampled image

1

Although 0 is the mode of the 2-downsampled image, the correct answer for the 3-downsampled image is 1, the mode of the original 8×8 image.

1. Implementation

You can either use C style arrays, or Boost.MultiArray (`http://www.boost.org/doc/libs/1_45_0/libs/multi_array/doc/index.html`). The boost library is provided with *zi_lib* (`https://github.com/zlateski/zi_lib`), which we use in the lab (subdirectory: `https://github.com/zlateski/zi_lib/tree/master/external/include`).

For multithreading, you can use the *zi_lib* concurrency library (`https://github.com/zlateski/zi_lib/tree/master/zi/concurrency`). Examples of usage are given here: `https://github.com/zlateski/zi_lib/tree/master/zi/concurrency/test`. The library is header only, so you don't need to compile anything. Just git-clone it, and include its path (and path of the external boost libraries). Also, if you are using Linux, don't forget to link against pthread and rt:

```
g++ your_file.cpp -Ipath/to/zi_lib -Ipath/to/zi_lib/external/include
-lpthread -lrt -o your_binary
```

2. Efficiency

Your code should use the fastest possible parallel algorithm. Answer the following questions about computational complexity. Let N be the number of pixels in the original image, n the number of unique pixel values in the original image, and M the number of parallel threads.

(1) How does the execution time of the fastest parallel algorithm scale with N, n, and M?
(2) How does memory usage scale with N, n, and M?

3. Style

The style of your code will be evaluated, including aspects such as

- organization into functions
- object-oriented design
- memory management