Block Downsampling Using Modal Values Assignment Report

Professor: Sebastian Seung
Author: Yu Chen
Date: Jan-21-2016
email: yu.chen@yale.edu

Content

Introduction

This problem is from Professor Sebastian Seung. Detailed problem description is in the same folder as this report, named **DownsamplingAssignmentRevised.pdf**. I implemented the problem in C++. Code explanation and analysis will be presented in this report.

System Requirement

The compiler I used is g++ 4.8.4, the operating system is Ubuntu 14.8.

I used the boost multi_array library from zi_lib folder, but for some reason when I tried to use the zi_lib concurrency code, it will pop up with so many errors. I guess that comes from system settings or compilers. I have been trying to solve the problem, but under time pressure, I finally gave up using zi_lib multi-thread programming library, and use the C++ standard pthread library. It's great to see your student Aleksandar Zlateski used that library in paper "A Design and Implementation of an Efficient, Parallel Watershed Algorithm for Affinity Graphs". It's amazing that the program can process 90GB image efficiently!

How to Run the Code

In Linux environment, copy paste the following command in a terminal

```
$ mkdir myworkspace
$ cd myworkspace
$ git clone https://github.com/AlbertYuChen/block_down_sample_.git
$ make
$ ./block_down_sample.exe
```

The output is the list of original image and output image, it shows the coordinate of each pixel and pixel value.

```
====== original img =======
0,0,0,0,0,0 : 1
0,0,0,0,0,1 : 1
0,0,0,0,1,0 : 0
0,0,0,0,1,1 : 1
0,0,0,1,0,0 : 2
...
3,1,3,7,1,1 : 0
====== end original img ======
```

```
====== output img =======
0,0,0,0,0,0 : 1
0,0,0,1,0,0 : 2
0,0,0,2,0,0 : 0
...
1,0,1,3,0,0 : 1
====== end output img =====
```

I have put 4 examples in the main function, you can choose which one you want, then make clean, compile and run again.

Potential Problems

It has problem to compile my code on OS X(compiler g++ Apple LLVM version 6.0) and Ubuntu 12(compiler gcc version 4.6.3). I know it's quite annoying to fix the system error, so if you have such problem, I will let you to SSH to my Ubuntu machine and test the code.

Code Analysis

I write C++ code that outputs all l-down samplings of the original image, and the program is using multithreaded algorithm on a multicore processor.

Class **Block_Down_Sample** is designed to deal with boost multi_array original input and will yield output sampled image. The class is declared in **Block_Down_Sample.hpp** and the functions are implemented in **Block_Down_Sample.cpp.** Since the class need template parameters, both *.cpp and *.hpp files should be included in the main function.

User Interface

Four user functions are provided to work out masked image with input image. A simple example is the following:

```
Block_Down_Sample<int, 6> B(A, D, B_length);
B.print_original_img();
B.cal_masked_img(16);
B.print_output_img();
```

Here, class initialization requires two template parameters, one is the data type of the input image, the other one is the dimension of the image.

A: input original image, the data type is boost array.
D: is the dimension size array, which tells you how many entries in each dimension.
B_length: is the block size, used to scanning the original image to get output image.
B.print_original_img(): will print the input original image.
B.cal_masked_img(16): this function will run the down sampling.
B.print_output_img(): will print the output image.

Algorithm

My algorithm will divide the original image into blocks, and each block will be replaced by the most common number(or one of the most common numbers) within that block. The down sampling problem can be highly parallelized, because there's no data dependency between blocks calculation.

One thread will be scheduled to calculate the most common number in one block. The thread will scan along lower dimension axis then to higher dimension axis. For example, if the image is a 2-D image, and the horizontal axis represent the first dimension, and the vertical axis represents the second dimension, within the block, the thread will scan row by row from top to find out the most occurrence. The reason for such scanning direction is based on the following observation: When I initialize the boost array, using:

```
std::fill( A.origin(), A.origin() + 16, 1 );
```

The function will feed one entry then shift the pointer to the next one, and it walks through the image from lower dimension to higher dimension. Computer cache always load next chunk of data ahead of the current reading location, so if we read data alone the RAM addresses, we can take great advantage of cache mechanism. If we read data skipping a large shift position in RAM, the data may not be cached, so that CPU will load new chunk of data from RAM to cache. My observation tells me how the boost array arrange the image on RAM, and that's why I process the block from lower to higher direction. With limited time, I didn't have a chance to look into the boost multi array code to better understand how it arrange memory, but it will be helpful to improve memory performance.

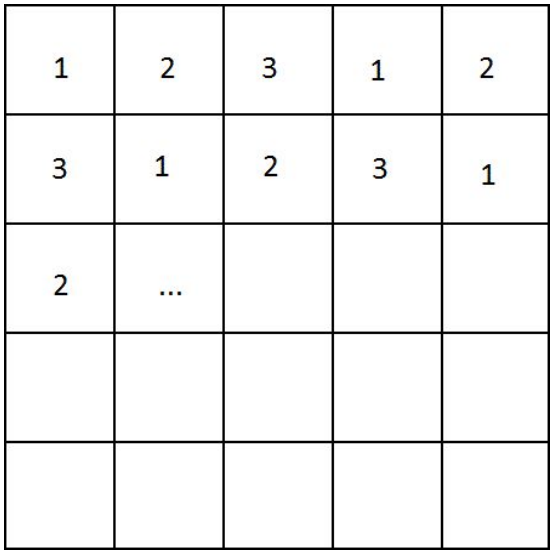

Figure 1. Example of 3 threads arrangement on a 2-D image

If it totally has M threads, those threads also will be arranged from lower dimension to higher dimension to process blocks. Figure 1 is an example of how it arranges 3 threads for a 2-D image. Thread "1" will finish the first block on the top left first, then skip 2 blocks and go to the fourth position. Thread "2" will start from top second position and move on to the 5th one. Those threads will recursively finish all blocks on their ways, without waiting for other threads. The program will yield the output image when all threads finish their jobs.

Test Examples

I put four examples in the **main.cpp**, you can choose which example you want by selecting the number

```
#define EXAMPLE 4
```

EXAMPLE 1: This is the repeat of the first example in the problem sheet.
EXAMPLE 2: This is the repeat of the second example in the problem sheet, with block length 4.
EXAMPLE 3: This is similar to the second example in the problem sheet. But the data type is double, and the block size is 2.
EXAMPLE 4: This is test of high dimensional image with 6 dimensions, and the block size is 2.

Welcome to use your original image to test my code.

Efficiency Analysis

Let N be the number of pixels in the original image, n be the number of unique pixel values in the original image, M be the number of parallel threads, B be the number of blocks totally.

1. How does the execution time of the fastest parallel algorithm scale with N, n, and M?
   Threads will work independently, so if B > M, then total N pixels will be parallelly scanned by M threads, so the running time complexity is *O(N/M)*. If B < M, then the running time complexity is *O(N/B)*, because only B threads will be effective.

2. How does memory usage scale with N, n, and M?
   In the searching most common number function, it will use std::map function. Worst case is there are n unique values in every block, and the map need to keep n slots. So the total memory usage is *O(N + M*n)*

Discussion

In my algorithm, one block will be only handled by only one thread. If the block is very huge, multi-thread programming won't take too much advantage. So in this case, probably more than one thread can be used in most common search within one block, but data dependency will be introduced. Also the synchronization between threads will take extra time. If the block is very small, the overhead will be significant, which may include thread scheduling, map function setting up and etc. So in this case, probably one thread can take care of more than one blocks to avoid more overhead.

In my current algorithm, although there are no data dependency between threads, once one thread get finished, it will wait for other threads to "join". Probably I can push rested threads to continue work, so that to make sure at any time, the program will be parallelized at the same level. But block signs will be used to mark which one has been done. Furthermore, without testing the discrepancy between threads, pushing rested threads may be redundant.

Why I am Interested in Neuron Level Neuroscience(not relevant to this report)

My current work is using fMRI and EEG data to understand why childhood absence epilepsy cause loss of consciousness. I use the fMRI data to figure out the brain network first, and find how seizures affect normal network pattern, which may result in loss of consciousness. But the resolution of both fMRI and EEG are very low. The fMRI data voxel is 4mm x 4 mm x 4 mm, in that cube, it may contain millions of neurons, and it's so far away from deeper understanding of the linkage between how human think and how neuronal network perform. People have been using many methods to figure out how neurons connect to each other and how signals pass through the network. But given limited technology, the whole brain map is still a mystery. Your research is trying to reconstruct the 3-D structure of neurons, which is so amazing, I have seen your TED talk on YouTube before. That would be a great opportunity to join in your lab and do research on the most fabulous machine in the world.