Chapter 6

Large-Scale Image Processing

In this chapter we present a scheme to process very large images that do not fit in the memory of a single computer. Such images are divided into blocks. Because of the spatial locality properties of wavelets these blocks can be processed quasi independently, allowing efficient parallel programming. The blocking is almost completely transparent to the user. Our implementation is based on the software library WAILI we developed before.

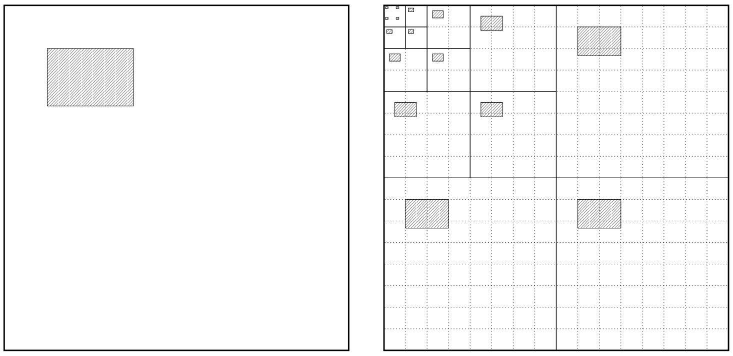
6.1 Introduction

In application areas like Geographical Information Systems (GIS), image sizes are measured in gigabytes (GB) and even terabytes (TB). For example, an aerial color image of Belgium, where each pixel corresponds to one square meter, consumes 90 GB of storage. It is obvious that such images need to be stored and handled in a compressed form. Wavelets can provide for this since wavelet-based techniques offer better compression rates than other techniques. Moreover, many image processing operations can be done in the wavelet domain. Some of them (e.g. noise reduction and edge enhancement) can even be done better in the wavelet domain than in the spatial domain.

We describe an extension of the software library WAILI to block-based processing, which allows for a parallel implementation [55].

6.2 Very Large Images and Tiling

We define an image to be *very large* if it does not fit in the memory of the computer system that is processing it. Note that by this definition, whether an image is very large or not depends not only on the image but also on the computer system. Examples of very large images are:



(a) (b)

Figure 6.1: Locality in the (a) *space* and (b) *wavelet domain*. The dotted lines in (b) indicate block boundaries.

- An aerial color picture of Belgium, digitized at 1 pixel per square meter (90 GB) on a high-powered workstation,
- An aerial picture of a city on a set top box or kiosk system.

6.2.1 Tiling

In order to process very large images, they have to be divided in blocks (tiling). For this division only the spatial domain is important, and since wavelets are localized in the spatial domain, one needs only a limited subset of the input image to calculate a subset of the wavelet coefficients. The same is true for the inverse transform. From Figure 6.1, it is clear that only a fraction of the blocks in the right part of the figure are needed to reconstruct the rectangle in the left part.

Since not all blocks have to be present in main memory at the same time, this allows for the successful processing of very large images on small uni-processor machines with limited memory. Moreover, the processing of the blocks can be distributed among multiple machines and/or CPUs.

Within WAILI, each subband is divided separately in blocks. Blocks are square and block sizes must be powers of two (except near the borders of the image). Because small blocks cause more bookkeeping overhead, it is recommended to use the same block size at different resolution levels.

Since we want to process both complete images and subimages that are part of a large image, the upper left corner of a (sub)image is not always located at the origin $(0,0)$, but may have arbitrary coordinates (see Figure 6.2). To facilitate the wavelet transforms, the internal boundaries of the blocks must always be aligned to the block size, i.e. internal blocks must start at coordinates that are multiples of

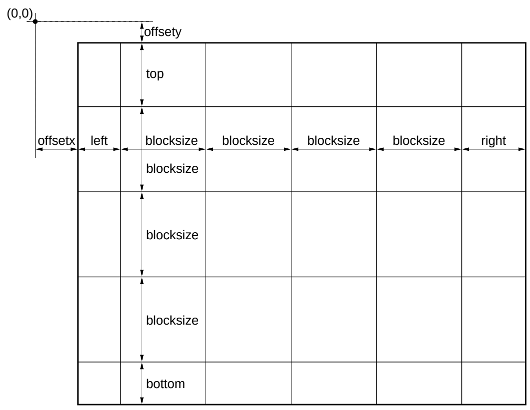


Figure 6.2: Division of a subband in blocks. The margins (*top*, *bottom*, *left* and *right*) are chosen such that the coordinates of the block boundaries are always multiples of the block size.

the block size. Hence, not only the blocks at the right and bottom borders of the image may be rectangular and smaller than the block size, but also at the left and upper borders.

The whole wavelet decomposition is divided in blocks like shown in Figure 6.3. Within each channel (color images consist of multiple channels), the tree forms a "Mallat" ordering (cfr. Figure 5.3). A block within an image is thus uniquely defined by 5 indices: *channel* , resolution *level*, *subband*, *column* and *row*.

To implement wavelet transforms on tiled images, one has to consider the following steps:

Splitting step (lazy wavelet transform) To split a block in "even" and "odd" coefficients, in both the horizontal and vertical directions, one needs that block only. So this step is trivial to parallelize.

Lifting steps To calculate the lifting operation on a border coefficient, some coefficients from an adjacent block are needed. Thus in a parallel implementation there must be communication of border coefficients (exchange of "overlap regions", with a width depending on the wavelet transform). Since in a primal lifting step the "even" blocks (i.e. the blocks with the even coefficients) are updated by the "odd" blocks, and vice versa in a dual lifting step, these updates can be done independently in each block, except for the communication just mentioned. Hence, this step can be parallelized easily. The parallel efficiency grows with increasing block size due to the "perimeter

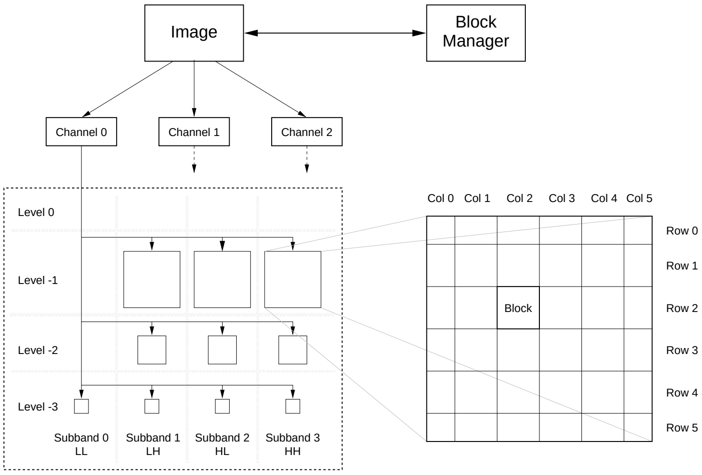


Figure 6.3: Subdivision of an *image* into *channels*, *levels*, *subbands* and *blocks*.

effect".

6.2.2 Block Management

We started from the software library WAILI that we developed before and added support for very large images that are divided in blocks. This is nearly completely transparent to the user: all traditional operations can be performed, just like on small non-blocked images (see Appendix B).

The "block manager" is responsible for managing the blocks of which a very large image is composed. To access a part of the (transformed) image, the block manager is asked for blocks and retrieves them, e.g. from disk or via a network. If a block is modified, it has to be updated on the storage device. Blocks are passed to the compression module to remove redundancy among the wavelet coefficients within each block. The block manager for sequential processing also keeps a cache of recently used blocks to speed up processing (for parallel processing we assume all blocks are simply distributed across all processing nodes). An overview of the complete system is given in Figure 6.4.

To obtain a parallel version, we only need to adapt the block manager and introduce communication of the border coefficients during the lifting steps.

6.2.3 Implementation in WAILI

Since the basic block of a wavelet decomposition hierarchy is a subband (subband), tiled images are implemented by deriving two subclasses of the NTChannel class

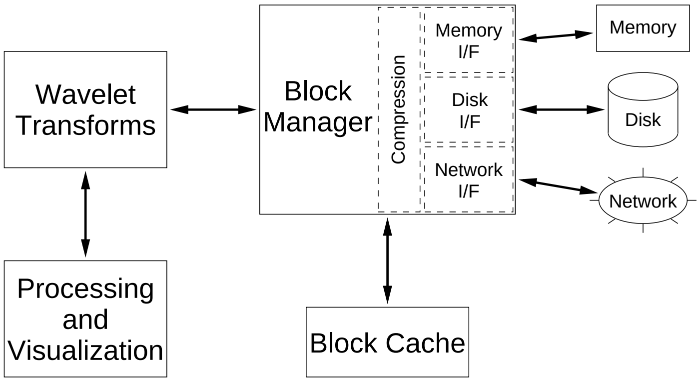


Figure 6.4: Design of the large-scale image processing system.

(see Section 5.2.4):

NTChannelR A regular subband, consisting of one block only.

NTChannelB A blocked subband, consisting of several blocks.

With each image, a *BlockMgr* object is associated, which takes care of managing the individual blocks. The blocks of an image can be stored in main memory, in files on disk, . . . , depending on the type of the BlockMgr.

By adding support for tiled images to WAILI, the code size increased from ca. 12500 to ca. 21000 lines of C++ code.

6.3 Image Processing on Tiled Images

Many image processing operations can be performed on wavelet-transformed images. If large images are divided in blocks, it is feasible that we can still perform those image processing operations on the tiled images.

6.3.1 Block-Based Denoising

WAILI uses soft-thresholding to reduce additive, stationary noise. The threshold is selected at each transform level and for each subband separately, using generalized cross validation [57]. Since this is an asymptotical method, it only performs well for sufficiently large subbands.

For wavelet-transformed images that are divided into blocks, the denoising technique is applied independently on each block of each subband. Because only the high-pass subbands are processed and blocks belonging to low-pass subbands are left alone, it does not become visible where the borders between the blocks are located.

Visual inspection by experts in the field showed that one gets better results, compared to the case where complete subbands are denoised. This is explained by the threshold now being more adapted to the local properties of the image. Thus denoising is not only easily parallelizable, it also yields a better image quality.

6.3.2 Image Compression

Due to their decorrelating properties, wavelets are well suited for the compression of images and signals. If a sufficiently smooth image is transformed, most high-pass coefficients will be zero or very small.

Compression algorithms based on wavelets often use a variant of "zerotree encoding" (see Section 4.5.4). However, for very large images, each zerotree contains coefficients from a multitude of blocks. Thus zerotree encoding will cause a block access pattern that is not feasible and the application of zerotree encoding is limited. A possible solution is to use a two-level encoding strategy:

1. For the lowest resolution levels, zerotree encoding can still be used because the subbands at these levels consist of only one block.

2. For higher resolution levels, a block-based encoding can be used. At these levels the subbands contain several blocks. If the block size is sufficiently large, each block will contain enough coefficients to be well compressible. Note that these subbands are high-pass subbands, with many small or zero coefficients and (typically) only a few large coefficients. Since all blocks can be compressed independently, this step is trivially parallelizable.

6.3.3 Crop and Merge

WAILI already provided crop and merge operations on wavelet-transformed images. After we had added support for very large images, these crop and merge operations could be performed on tiled images, too. Since the tiling of subbands is completely transparent and because only a limited subset of all blocks are required, crop and merge are performed in an efficient way.

6.3.4 A GIS Application

In GIS applications, a very large aerial image is usually created by combining multiple small aerial pictures. A wavelet-transformed tiled version of the very large image can be created using the following recipe:

1. Create an empty very large image, containing no data (i.e. a grey image).

2. For all aerial pictures i
   (a) Load aerial picture i into memory.
   (b) Transform aerial picture i, taking care of the absolute coordinates and resolution of the picture.
   (c) Paste the wavelet-transformed aerial picture i in the very large image.

After these steps, one has a wavelet-transformed image that is exactly the same as if one had first glued together all aerial images, and transformed the result. However, our method requires much less memory.

Thanks to the multi-resolution properties of the wavelet transform, one can now select an arbitrary rectangular subimage of the large image, at any dyadic resolution level, and cut it out of the large image, for display or print. If an intermediate resolution is required, the subimage must be scaled. One can even use the above techniques to "walk" through the image (panning), and zoom in or out depending on the degree of interest.

6.4 Transform Results

To get an impression of the sequential performance of our wavelet transform code, we transformed an image of 3584×3584 pixels, using a block size of 128×128 pixels, for a total of 28×28 blocks. We used the CDF $(2,2)$ biorthogonal wavelets. All calculations were done on 16 bit integer data on a single processor machine (Intel Pentium II, 350 MHz) with sufficient memory to avoid paging. Note that after 2 transform levels, the subband size is no longer divisible by the block size and one has to take care of rectangular blocks.

Each level of the CDF $(2,2)$ wavelet transform of n blocks consists of 5 steps:

- Lazy wavelet transform in both the horizontal and vertical directions. In order to maintain the block size, 4 blocks are jointly transformed into 4 new blocks. In total n blocks are transformed.
- Lifting steps on the columns of the image: a dual lifting step ($n/2$ blocks) and a primal lifting step ($n/2$ blocks).
- Lifting steps on the rows of the image: a dual lifting step ($n/2$ blocks) and a primal lifting step ($n/2$ blocks).

Timings are shown in Table 6.1. An additional overhead of about 0.6 seconds was caused by the supply of blocks by the block manager. However, in a parallel implementation we can assume that the blocks are distributed over all processors, so this overhead can be spread over the processors too.

In a parallel implementation, the lazy wavelet transform requires no communication if the 4 neighboring blocks are transformed by the same processor. If we

Transform Level	Lazy Transform # Calls	Lazy Transform Time	Lifting Steps # Calls	Lifting Steps Time	Total Time	Cumulative Total Time
1	784	0.26	1568	5.59	5.85	5.85
2	196	0.02	392	1.34	1.22	7.07
3	49	0.01	128	0.33	0.34	7.41

Table 6.1: Timings for the wavelet transform of an image of size 3584×3584. Each call corresponds to the processing of one elementary block. All timings are in seconds.

would use $14 \times 14 = 196$ CPUs, the lazy wavelet transform for the first level could be calculated in 1.33 ms and the calculation time for the 4 lifting steps would be 28.5 ms. The lifting steps require communication of the border coefficients. For the CDF $(2,2)$ transform, in each lifting step 2 strips of 128 coefficients must be exchanged between processors. On current parallel systems (e.g. IBM SP2), the communication cost for such short messages is dominated by the startup time ($\mathcal{O}(50\,\mu\text{s})$), which is negligible compared to the calculation cost.

However, the amount of data to be transformed is divided by 4 at each level. Hence, at the second level, we can keep the block size equal to 128, but then only one fourth of the CPUs can be kept busy. Thus the execution time for the second level will also be approx. 30 ms. A similar reasoning holds for the other levels, but in our example rectangular blocks will appear at level 3, causing an additional overhead.

We can conclude that a parallel implementation can be efficient if the number of transform steps is small, and the number of processors is not too large, relative to the image size.

6.5 Conclusions

We have described a strategy to perform wavelet-based image compression and processing on very large images. The images are split in blocks, which can be handled nearly independently.

The (integer) wavelet transform is implemented using the lifting scheme, which can be parallelized easily and efficiently, since only communication of coefficients at the borders of the blocks is required.

The method is implemented as an extension to the software library WAILI. Blocks are accessed and handled under supervision of a block manager. Although the software has not been parallelized yet, the limited communication and the existing block manager allow an easy and efficient parallelization.

The division in blocks also has a positive influence on the performance of our denoising algorithm. Because denoising works independently on the blocks, it adapts better to the local properties of the image.