Sixth International Conference on Information Science and Technology
Dalian, China; May 6-8, 2016

The On-line Viewing System of the Distributed Gigapixel Image

Nan Mu, Hongzhao Chen, and Guijin Wang
Department of Electronic Engineering
Tsinghua University
Beijing, China
Email: mun13@mails.tsinghua.edu.cn

Houyong Feng and Feng Chen
Department of Automation
Tsinghua University
Beijing, China
Email: fenghy0505@163.com

***Abstract*—The observation of the gigapixel image is the current research hotspot and how to view the image in short time is a big challenge. So in this paper we present an on-line viewing system of the distributed gigapixel image. We construct the MPI programming framework and use the MapReduce data processing model to stitch the subgraphs distributed in the different servers. Based on the size and location of the region of interest (ROI), only the relevant subgraphs will be transmitted and processed. Utilizing our on-line viewing system, users can observe the ROI of the gigapixel image in short time.**

Keywords—image stitching; gigapixel image; MapReduce; on-line viewing system

I. INTRODUCTION

At present, the imaging systems of wide Field of View, high resolution cameras [1] have been developed so quickly that we need a reliable viewing system to observe the gigapixel image. Further more, for some observation samples like the dynamic living microscopic image [2], we have the need for on-line viewing while the imaging system is working. However, the gigapixel image is so huge that how to transmit and process the image quickly is a big problem. At the same time, there is almost no suitable screen to view the whole gigapixel image.

Due to the need for the gigapixel image viewing system, the professor Johannes Kopf designs a system to observe the image [3]. They store the gigapixel image as a pyramid of 256^2pixel tiles and the huge image contains 9 levels of the pyramid. After the image storing, they can view the gigapixel image in different scales and positions, just like using the electronic maps [4]. But this method is off-line and they must generate the huge gigapixel image at first. Further, the professor D.R.Golish and D.J.Brady complete a gigapixel camera to observe the natural scenery [5], [6], but their viewing system is also off-line. They generate the gigapixel image and then store the data in servers. After these operations they can view the image in different locations and scales.

So our contributions in this paper are that we propose an on-line viewing system of the distributed gigapixel image. In our system, we don't need to generate the final huge gigapixel image. And due to the huge data of the image and the limited size of the screen, our design ideas of the system are that each time we just need observe the screen-size image block and only the relevant subgraphs will be transmitted and processed, so we can significantly reduce the transmission data size. As we know, the common network bandwidth can satisfy the data size of the usual screen resolution quite well. In our experimental environment, utilizing the on-line viewing system users can observe the region of interest of the gigapixel image in less than half second.

II. METHOD

Our viewing system consists of the following three parts, which is shown in the Fig. 1: the first part is the ROI location information, and in this part the screen displays the ROI of the gigapixel image. When observers want to see new ROI of the image, they can click and drag the image. After the dragging operation, the display platform automatically figures out the location of new ROI in the large image, and then the display platform delivers the location information to the second part. The second part is the MapReduce module, which target is to generate the appropriate image blocks. This module gets the ROI position information and then just collects the small part of the subgraphs that are relevant to the new ROI. Then we use map and reduce operation to process these subgraphs and we can generate four image blocks. So we collect these blocks and cut the image region which we want to observe. Finally, the ROI display part receives the image block and users can observe the partial image they are interested in.

The following three parts will describe our on-line viewing system in detail.

A. ROI location information

The huge gigapixel image is formed by many subgraphs which are distributed in the different servers. And we design a display software written in C# to show the region of interest. When the users are interested in other parts of the image, they can click and drag the image in the screen using the mouse. At the same time, the software will record the coordinate (x, y), which is the upper-left vertex coordinate of the ROI in the huge image. What is more, the software need record the size of ROI (height, width) that is equal to the screen size. Then we create an array to save the location information (x, y, height, width). Now we obtain the ROI location information and the software will send the information array to the image stitching module by socket correspondence.

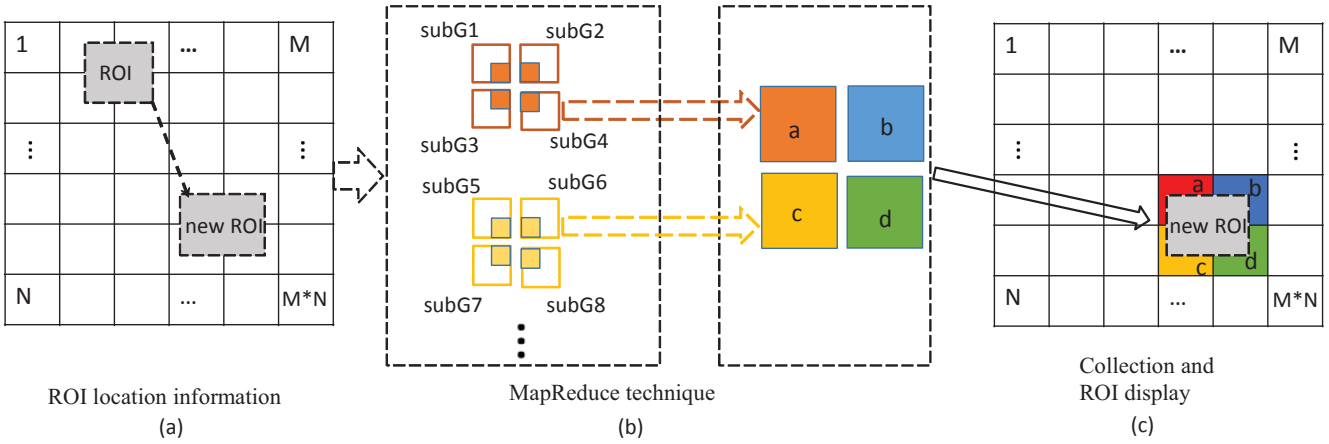


Fig. 1. Flow chart of our on-line viewing system of the distributed gigapixel image. (a) This is the ROI location module, and in this module servers can get the location information of the new ROI. (b) This is the MapReduce technique module, and in this module servers can generate four image blocks a, b, c, d. (c) This is the collection and ROI display module, and in this module display platform receives the image blocks and then shows the new ROI for observers.

When the image stitching module receives the ROI location information, we can know the coordinate and size of the partial image which is going to be displayed in the gigapixel image. So the next step is to find the relevant subgraphs which overlap slightly and stitch these subgraphs into a region of interest. In order to achieve this goal, we adopt the MapReduce technique. The concept MapReduce is promoted by Google [7].

B. MapReduce technique

In this part, we divide the huge gigapixel image into M*N blocks, and each block is relevant to one or more subgraphs. So we can know the ROI is contained by which image block group. Now we only need generate the small four blocks a, b, c and d, so we must get the information that each small block is relevant to which subgraphs.

Then we will generate one maptable matching to one small block. As we know, our calibration module generates the calibration maptable, which is the matchup between each subgraph and the global huge image. The calibration maptable is the group of 121 small maptables, But for our on-line viewing system, this original calibration maptable is not matched for our M*N small blocks. Fortunately, we can generate our maptable from the calibration maptable: each small block is a part of the global image, so for each pixel of the image block, we find the matching local subgraph pixels through the calibration maptable [8], [9]. Because one global pixel is relevant to one or more pixels of the local image, our maptable is shown in Fig. 2. Generating this maptable will take a long time, but as we know, the maptable can be processed in advance before running our system, so this procedure will not bring side effect.

Now we focus on how to generate the four small blocks a, b, c, d. Based on the distributed data in the different servers, we use the MapReduce paralleling technique [7]. This model is made up of two parts: map and reduce. And the map and reduce are independent so the whole parallel framework can work very well.

For our map operation, as shown in the Fig. 3, the input key is an array consisting of the subgraph Number, the local

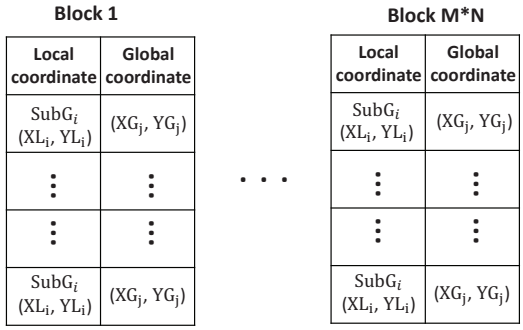


Fig. 2. Our own maptable consists of M*N maptables and each of them is relevant to one small image block. The SubG is the No. of the subgraph and the (XL,YL) is the floating point value.

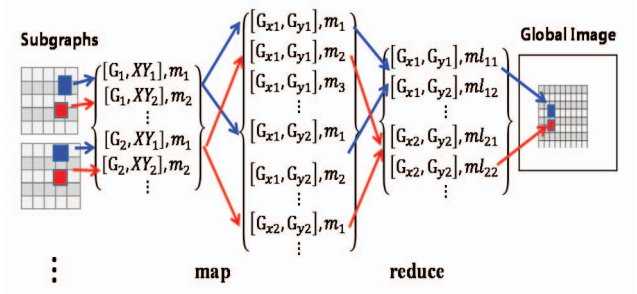


Fig. 3. The MapReduce technique for our on-line viewing system. The G is the No. of the subgraph. The XY is local coordinate which is floating point value and the (G_x , G_y) is the global coordinate. The m is the pixel value and the ml is the averaging pixel value.

pixel coordinate(Xlocal, Ylocal), and the input value is the gray value of the subgraph pixel. So the input pairs are ([subgraph_number, Xlocal, Ylocal], [local pixel value]), and the key/value pairs of the map output are ([Xglobal, Yglobal], [global pixel value]). The projection of a given pixel in a given subgraph is independent of any other pixel in any other subgraph. To get the output, the map function should contain our maptable to build the relationship between local and global coordinate system. But the maptable only has the

position transformation relationship, how to compute the global pixel value is also a problem. As we know, the Xlocal and Ylocal are both floating point values and the Xglobal and Yglobal are both integer values, because the calibration result cannot just right obtain the projection between the integer coordinate values. So we need use the bilinear interpolation of the backward mapping to compute the pixel value of the coordinate (Xlocal, Ylocal), and this value is the global pixel value that we want to get.

Finally, after the map operation, we get a list of key/value pairs ([Xglobal, Yglobal], [global pixel value]), but due to the overlap of the adjacent subgraphs, the output keys [Xglobal, Yglobal] may be the same. To solve this problem, we design a reduce function to merge the overlap region [10]. For the same keys [Xglobal, Yglobal], we compute the weighted average value of all the relevant global pixel value:

$$ml_i = \frac{\sum_{i=1}^{n} t_i m_i}{\sum_{i=1}^{n} t_i} \quad (1)$$

In the equation (1), the ml_i is the average pixel value and the t_i is the corresponding weights. The t_i is generated by the calibration module [8], [9], and the function of t_i is mainly to eliminate the uneven brightness of the different subgraphs.

Just as we described in the first step, the gigapixel image is divided into M*N blocks, and each block operates the reduce function independently in their own processes. And also, only the relevant subgraphs will be transmitted, which is the first transmission module. Our system just need generate the a, b, c, d small blocks, and we use four processes to compute the reduce results.

C. Collection and ROI display

After the MapReduce operation, we get these 4 small blocks, and the users' ROI is the part of the group of them. So we need send the block b, c, d to the server which the block a is generated in, and this is the second transmission module. Then we combine the blocks into a new image, and we cut this new image to get the final ROI. And also we convent the ROI image to a byte stream, then we send the byte stream to the display software by socket method, which is the third transmission module, and the screen will show observers what they want to see. If the users click and drag the image again, the preceding steps will generate the new ROI for them.

III. Results

A. Experimental environment

Our input image is the group of 121 subgraphs which are microscopic Ovarian cancer image, as shown in the Fig. 4. The resolution of 100 subgraphs is 4000*4000, and others are 4000*3000, 3000*4000 or 3000*3000. And the subgraph is an 8-bit grayscale image. Every two adjacent subgraphs have 25% overlap, so the stitched huge image has one billion and ninety million pixels. And also, these 121 subgraphs are distributed in the 9 servers, so each of the 5 servers has 13 subgraphs and each of the other 4 servers has 14 subgraphs.

Our programming framework has these 9 servers for the paralleling cluster, and each server has dual 4-core Intel i7 4790 processors. For MPI [11], each server can create specified number of processes to satisfy the map and reduce operation. Concretely, the number of the map processes should be equal to the number of the subgraphs, so how many subgraphs one server has, this server will create corresponding processes to do the map operation. The number of the reduce processes is based on how many small blocks the huge image is divided into. At the same time, the small block size should match with the screen size, because if the block is too small, we must create lots of reduce processes which will beyond the performance of servers and if the block is too large, the amount of transmission data is so big that it will limit the on-line effect [12], [13]. Our screen size is 1920*1080, so we set the block size as 2000*1100. Based on this, we will get 363 small blocks, so each of the

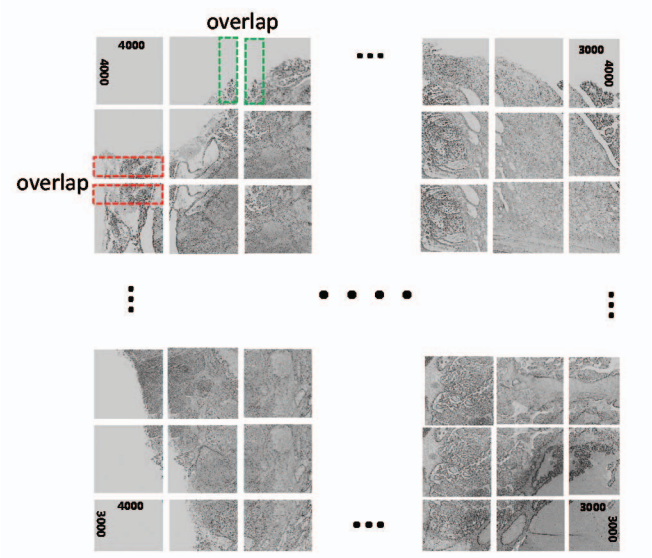


Fig. 4. The 121 subgraphs of the ovarian cancer and every two adjacent subgraphs have 25% overlap.

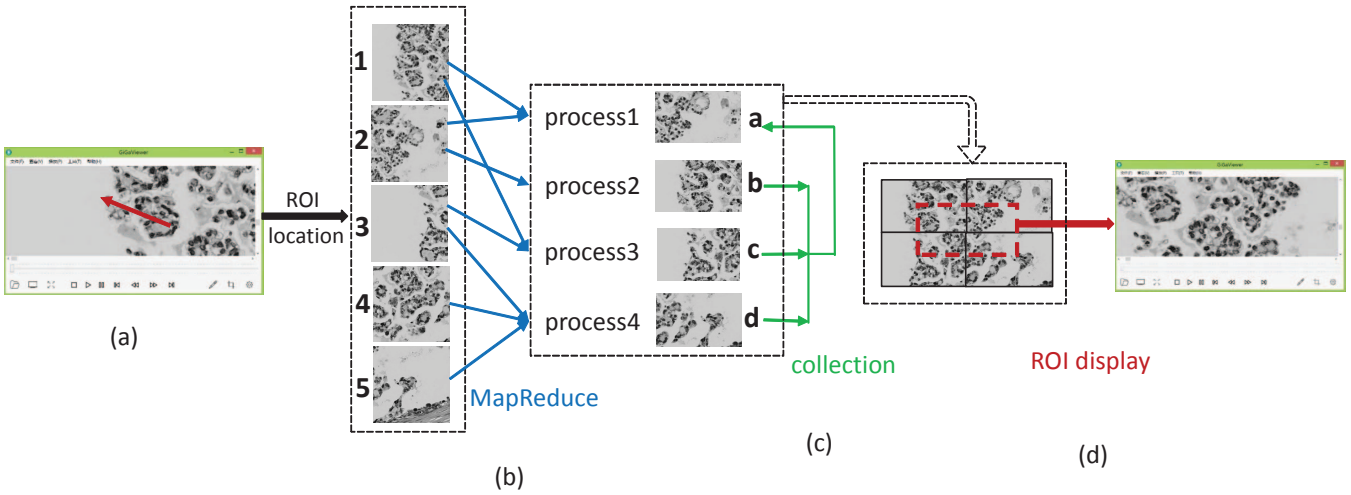
servers should create 40 or 41 reduce processes.

B. Experiment results and analysis

The experiment result of the data transmission time is shown in the table 1. The total transmission time is from the following three modules. The first is the MapReduce module, and the Fig. 5(b)-5(c) describe the procedure of generating the small blocks a, b, c, d. In the figure we can know the pixels of block a are from subgraph 1 and 2, block b from subgraph 2, block c from subgraph 1,3, block d from

Transmission module	MapReduce	collection	ROI display	**total**	
Average transmission data size (Mbit)		18.42	35.22	16.58	70.22
Average transmission time (second)		0.115	0.220	0.104	0.439

TABLE I . The Result of the Data Transmission Time



subgraph 3,4 and 5. So these 4 servers need different subgraphs from other servers, and the average transmission data size is 18.42Mbit. Our available network bandwidth is about 160Mbps so the first transmission time is about 0.115s. Now we generate these 4 small blocks and we need gather these blocks to the process 1. This is the second transmission and the average data size is 35.22Mbit which costs about 0.220 second. When we get the four blocks in one server, we cut these blocks and generate the final ROI. Then we should send the ROI to the display software and this is the third transmission with the data size of 16.58Mbit, which costs about 0.104 second. After these three times data transmission, the users can get the ROI display which they want to observe successfully as shown in the Fig. 5. So utilizing our viewing system, when the multi-microcameras gigapixel imager is photographing the dynamic micro samples, observers can see the cellular image in less than half second.

For the time performance analysis, in our paralleling environment, the map and reduce computation consume just 0.009 second, and the transmission time is totally about 0.439 second, so the major time cost is from the data sending and receiving. In the future, we will use the pipeline technique which will improve the resource utilization. And also we will use the faster network bandwidth, then the time of data processing and transmission will be further reduced, which can satisfy the higher speed of gigapixel image observation.

Fig. 5. The experiment results flow diagram. When users drag and click on the display platform, they can observe the region of interest of the gigapixel image in less than half second. (a) This is the display software when the users are clicking and dragging on the screen. (b) This is the MapReduce module. The relevant image blocks have been generated in this module(c) This is the collection module. (d) This is the ROI display module.

IV. Conclusion

This paper has presented an on-line viewing system of the distributed gigapixel image. The system is used for the imaging system of the wide Field of View, high resolution cameras. In this system, we don't need to generate the final huge gigapixel image. For the viewing system, we construct the MPI programming framework and use the MapReduce data processing model. Through MPI, we divide the huge image-stitching work into many small jobs. Further more, each time we see the screen-size image and we only need to stitch 4 or less small blocks into the small part of the gigapixel image. In our experiment environment, the time of a single viewing operation is from two parts: the three times data transmission which cost about 0.44 second and the MapReduce procedure which costs just about 0.009 second. If we use the pipeline technique and the faster network bandwidth, the time will be further reduced. Utilizing our system, while the wide Field of View, high resolution cameras are photographing the gigapixel image, we can observe the region of interest in short time.

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant No. 61327902.

References

[1] D. J. Brady, M. E. Gehm, R. A. Stack, D. L. Marks, D. S. Kittle and D. R. Golish, "Multiscale gigapixel photography," *Nature*, vol. 486, no. 7403, pp. 386-389, 2012.

[2] L. Cassimeris, N. K. Pryer and E. D. Salmon, "Real-time observations of microtubule dynamic instability in living cells," *The Journal of Cell Biology*, vol. 107, no. 6, pp. 2223-2231, 1988.

[3] J. Kopf, M. Uyttendaele, O. Deussen and M. F. Cohen, "Capturing and viewing gigapixel images," *ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[4] R. Szeliski, "Creating full view panoramic image mosaics and environment maps," *Proceedings of the 24th annual conference on Computer graphics and interactive techniques,* pp. 251-258, 1997.

[5] D. J. Brady and N. Hagen, "Multiscale lens design," *Optics Express*, vol. 17, no.13, pp. 10659-10674, 2009.

[6] D. R. Golish, E. M. Vera, K. J. Kelly, Q. Gong, P. A. Jansen and J. M. Hughes, "Development of a scalable image formation pipeline for multiscale gigapixel photography," *Optics Express*, vol. 20, no. 20, pp. 22048-22062, 2012.

[7] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 55, no.1 pp. 107-113, 2008.

[8] D. S. Kittle, D. L. Marks, H. S. Son, J. Kim and D. J. Brady, "A testbed for wide-field, high-resolution, gigapixel-class cameras," *The Review of scientific instruments*, vol. 84, no.5, 2013.

[9] Z. Zhang, "A flexible new technique for camera calibration", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[10] C. Liao, G. Wang and B. He, "Real time aerial video stitching via sensor refinement and priority scan," *IEICE TRANSACTIONS on Information and Systems*, vol. 95, no. 8, pp. 2146-2149, 2012.

[11] W. Gropp, E. Lusk, N. Doss and A. Skjellum. "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, no.6, pp. 789-828, 1996.

[12] Q. Miao, G. Wang, C. Shi, X. Lin and Z. Ruan. "A new framework for on-line object tracking based on SURF," *Pattern Recognition Letters*, vol. 32, no. 13, pp. 1564-1571, 2011.

[13] Q. Miao, G. Wang and X. Lin, "Kernel-based on-line object tracking combining both local description and global representation," *IEICE TRANSACTIONS on Information and Systems*, vol. 95, no. 1, pp. 2146-2149, 2012.