A Parallel Implementation of Content-Based Image Retrieval: Final Project Report

2008/12/18

Chunsheng Fang
University of Cincinnati
fangcg@email.uc.edu

Ryan Anderson
University of Cincinnati
andersr9@email.uc.edu

Abstract

Content-based image retrieval has many applications but remains a computationally intensive task. This is mainly due to the large size of an image database required for practical use. Our project aims to examine existing CBIR implementations and improve upon them using a parallel computing approach. Both parallelized offline feature extraction and online query process are implemented on the Beowulf cluster in Univ. of Cincinnati. Optimization and evaluation are also performed.

1. Introduction

The area of interest we have chosen is that of Content-Based Image retrieval (CBIR) [2]. **An online demo for CBIR image search engine has already been developed by the authors in Univ. of Cincinnati, 2008 [4].** CBIR deals with comparing similarities between images based on the content of those images. The Image content is compared based on a feature vector that is extracted from each image. This feature vector is pre-computed from the type of content to be extracted and compared (colors, textures, etc). Typically an image is queried against a training set or database of pre-computed feature vectors. The feature vector for the query image is computed and compared against the database. The top K stored images in database that are closest to the query images are then returned to the user as best matches. Worth mentioning, Google's image search is text-based, which requires manually labeled images for retrieval. The semantic gap between user retrieval demand and system retrieval results can be minimized by means of CBIR.

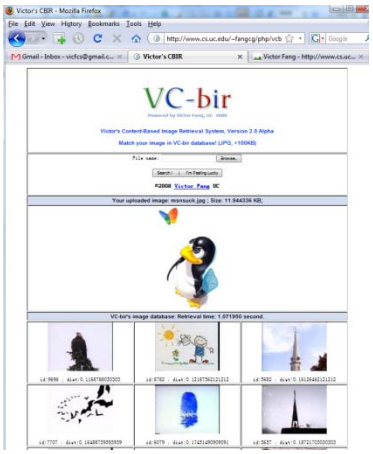

Figure 0, VC-bir image search engine in UC

1.2 Related work

CBIR is very computation-intensive due to several aspects, such as huge image database, offline feature extraction, online image retrieval, etc. Thus it is our hope to improve both the process of training and querying by parallelizing them. Our literature research found a paper by Lu and Yu (2007) [1]. They discuss a very similar approach to what we intend to take. In the

paper, a parallel application was implemented using the Manager-worker model. Both the training and querying used this approach on a large commodity cluster, Beowulf in Univ. of Cincinnati. The feature vectors are based on the color-histogram of the image and Local Binary Pattern (LBP) for texture analysis. C. Cheung, L. Po, K. Wong (2001) discuss a web-based interface for running image queries and presenting their results [3].

1.1 Our Approach

Currently we have a CBIR implementation that allows users to submit images for querying via the web [4]. Code written in PHP generates the feature-vector for the image and queries it against a training set database, which has 10,000 images. Top 10 images with the closest color-histogram match are retrieved in about 0.7 second, which is performed on a single Pentium 4 processor. Our project is to re-write the application to use the University of Cincinnati Beowulf cluster to parallelize both the offline feature extraction, and the online query process. We will then analyze the efficiency and speedup over the sequential method. Since our literature review yielded several papers with similar ideas, we'd also like to expand these by testing various possible performance improvements. Among these are the following:

Even though the offline feature extraction for the whole image database is only generated once, we can investigate improvements here by optimizing the assignment of images based on an even distribution of file size to process.

Since computation time is distributed amongst multiple processors, this opens the possibility of comparing images based on multiple attributes. This can yield a significant improvement over query accuracy in the same time a sequential query might take for one attribute comparison.

To improve the query speed, disk I/O can be minimized by using a listener process on the Beowulf cluster that maintains the feature-vector database in memory.

How to combine multiple image features to refine search results is another important research topic. One recent publication using probability based similarity measure can be found by authors from Univ. of Cincinnati [5]. We'll try to plug this new feature in if time allowed.

Promisingly this problem is highly scalable which means adding a node to Beowulf cluster will not cause any significant increase in memory requirement.

2. Application Analysis

The overall system architecture consists of two major parts: the offline image feature extraction (training), and the online image retrieval (query) process. We adopt a Manager/Worker model for both the training and query processes. During profile runs, 99% of computation time resided in the areas of code that computed the feature vector from an image file (training part), and that computed the smallest K distances of one feature vector against a list of other feature vectors (query part). These are the specific areas we chose parallelize.

2.1 Offline feature extraction

The process of creating an image feature database requires a large amount of file I/O. The image sizes can vary greatly, causing an unbalanced workload amongst worker processes. However, since this task is only executed once per set of images, and has no bearing on the online image query, it is not critical to load-balance the image files to worker processes. Thus the Manager process is passed a path to the directory containing the image files. It builds a list of each file in the directory of type "jpeg". It then sends a predetermined number of tasks to each Worker process. Each Worker will then open and process each image within the range and location provided by the Master. When the Worker finishes its assigned task, it will send its list of extracted feature vectors back to the Manager. The Manager then stores the received feature vectors into a primary vector array and assigns a new range of images to the Worker. If there are no images left to process, it tells the Worker to terminate. Each feature vector returned is tagged with the associated image id so that in future the corresponding image can be retrieved based on the best matched feature vector. When all images have been processed and all Workers have been terminated, the Manager writes its array of computed feature vectors to the database file.

It should be noted that the offline training procedure is expected to be highly time intensive for very large image databases (on the order of millions of images). The training process is, however, highly scalable. The Worker memory requirements are determined strictly by the image size and the vector package size. The manager memory requirements can be easily mitigated at a slight performance decrease by having it append the primary vector array to the database file once it reaches a certain size.

The majority of computation is spent in the image feature extraction function which is O(nm) where n and m are the width and height of the image. Thus the computation time is directly related to the image size.

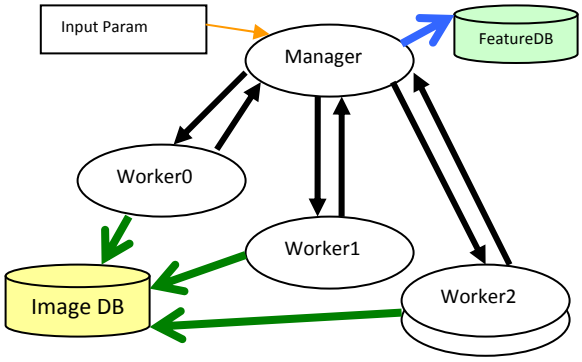

Figure 1. Task/channel graph for Offline feature extraction algorithm.

2.2 Online image retrieval

When the user submits a query image to the CBIR image search engine UI, the Manager first computes its feature vector (called the query vector) and broadcasts it to each Worker. The Manager then opens the image database, and sends a predefined block of vectors from the image database to any available Worker. Each Worker process receives a block of vectors from the database sent by the Manager. The Worker then calls a Distance function that takes the block of vectors and the query vector as input parameters. The Distance function returns an array of key-values pairs consisting of the image ID and its distance to the query image. Once the distance array is computed, it is then sorted using the built-in C-function "qsort" (O(nlogn))and the top K smallest distances are

sent back to the Manager. The Manager maintains its own array of the smallest K distances. Once it receives a set of distances from a Worker, it concatenates its distance array with the Worker's array, sorts based on the smallest distance, and takes the first K smallest distances. The Distance function is where most of the computation time is spent during the query. This is due in large part to the distance computation against each vector plus the sorting of the distance array.

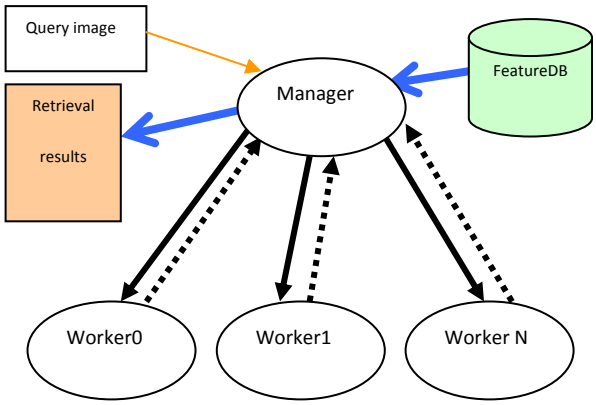

Figure 2. Task/channel graph for online image retrieval algorithm.

Note that online image retrieval is highly time-sensitive. Typically the user has patience span of only 2 to 5 seconds for retrieval results. More importantly a practical image database has millions of images to be retrieved every time. Promisingly this problem is highly scalable which means adding a node to Beowulf cluster will not cause any significant increase in memory requirement.

In both parts of the system, the Manager needs to keep all workers as balanced as possible, which turns out to be a NP-hard problem, called bin packing problem in the case of an arbitrary package size. For the purposes of this project we use images which are all relatively the same size and use a fixed vector package size. This also allows us to keep each feature vector a fixed size without significant loss of feature integrity. The task assignment problem degenerates to a polynomial algorithm which can be solved using round robin method and will be implemented outside the scope of this project.

3. Optimization Approach

There a many ways to approach optimization from the perspective of parallelizing the CBIR problem. We will discuss some of the possible areas that can be optimized and which methods we chose.

3.1 Disc I/O and Network Latency

Both parts for CBIR (training & query) require a significant amount of file I/O. The training part requires the opening of each image file and the eventual writing of a database file to hold the image feature vectors. Each worker must know the name and path of the all the image files it must process. At first glance, this could be achieved by the Manager Process scanning a directory of all images files, building a list of filenames, and sending a portion of that list to each worker. We felt this could be avoided and would improve network communication times between Manager and Work if we could minimize the amount of data the Workers needed to process files. Our solution was to assign each image file a sequential unique ID number and set that as the file name. Thus, the Manager only has to be aware of how many total files exit and can assign a set of images to a Worker by sending it two integers consisting of the range of images to be processed. The

Worker will be passed the path of the image files through the command line and will only have to covert the image ID into a path variable to open the file.

The query process also presents possible optimizations with Disk I/O and network transmission. Since the feature vector database can be very large and must have segments sent to each Worker Process, it would greatly save network communication to have the database file split into multiple smaller files, and each worker read its own file. This, however, poses two problems: The database file must be entirely read and split each time the number of processors changes (in order for there to be a one-to-one mapping of vector files to Workers); This also means that there will be as many disk reads and there are Worker Processes. Whereas these reads are done in parallel, they are all reading from the same disk. Moreover, in our environment the files reside on an NFS server. Thus there would not only be multiple reads from the same disk, but each would entail network latency anyway. With this in mind, we have the Manager Process read the entire vector database into memory and send blocks/packages of vectors to each Worker. We are able to vary the block size for each run and anticipated that the optimal block size would be equal to the number of feature vectors divided by the number of Worker Processes.

$$blocksize = \frac{total\_database\_vectors}{total\_worker\_processes}$$

Equation 0. Optimal block size

This, however, turned out not to be the case.

3.2 Clusters vs. Cores

The Beowulf cluster we tested on consisted of 16 servers, each containing 4 processors for a total of 64 nodes. Our initial assumption was that network latency between any pair of nodes would be equal. This assumption must be true for Equation 0 to hold. What we found was that network latency varied greatly between local nodes communicating on the same machine and nodes communicating on different machines. Two test programs were written to test and confirm this theory. As expected, the test results showed that communication between nodes on the same machine took less time since information is sent through the CPU bus instead of the network. This means that we cannot base the optimal block size on Equation 0 since we could do better by increasing the block size for the Workers running on the same machine as the Manager and keeping a relatively low block size for Workers on other machines. An optimization that leverages this information is beyond the scope of this project and will be pursued in future research. For the scope of this project, the greatest speedup for the query was measured while using 8 processors and a block size of 100 vectors.

4. Results and Design Analysis

4.1 Benchmark for training stage

We benchmarked the performance of training part of the program. We extracted the color histogram feature vectors for a test bed of 9907 images of similar size.

As shown in Figure 4, the more Worker processes used, the less time is required to finish the feature vector extraction for the entire test bed. When 18 processors are used the shortest offline feature extraction time is achieved (about 2 seconds). In this experiment, we use a block size of 100.

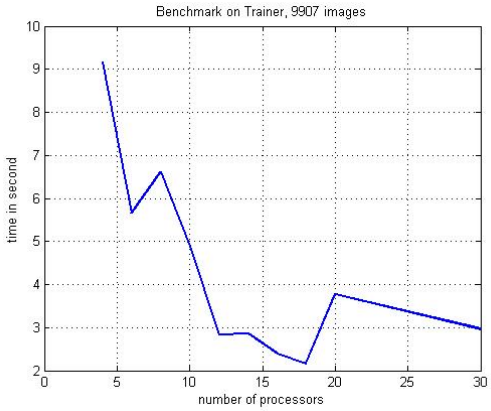

Figure 4. benchmark for training stage.

4.2 Benchmark for online query stage

Because online the query stage is very time-sensitive, we design several experiments to benchmark its performance.

As previously stated, this implementation has only the manager processor performing I/O access.

At the beginning, we tested some basic parameters as follows:

1. Network latency , Lambda = 1e-5;
2. Time spent on finding top 10 image ID from *blocksize = 100* of feature vectors, Chi = 5e-5;
3. Time spent on sending *blocksize = 100* of feature vectors to a worker, it's not easy to estimate because some nodes are internally connected and no need to use Ethernet. So when the worker and manage are in the same machine, the transmission rate is approximately 800MBps, otherwise 100MBps. Therefore, the transmission time is 0.7e-5 and 5e-5 second, respectively.

Interestingly, these times are in the same scale of 1e-5 second. And Chi is larger than the other parameters, which means that not much time is spent as overhead.

Note that *blocksize* is a critical parameter that needs to be tuned, we tested *blocksize* = 100 and 700 respectively, as shown in Figure 5, 6.

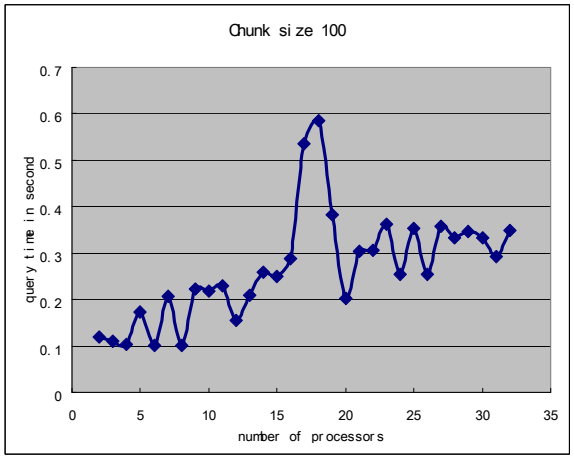

Figure 5. Online query performance when block size is 100.

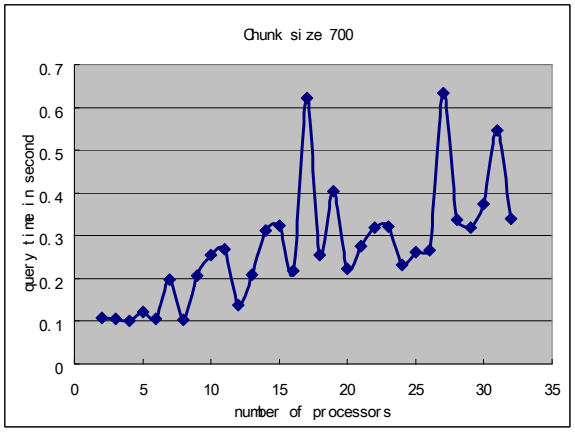

Figure 6. Online query performance when block size is 700.

There are some interesting findings when comparing these 2 figures:

1. When *blocksize* = 100 , as the growing of number of processors, the curve is smoother than *blocksize* = 700; Because smaller blocksize takes less time for the worker to finish, so it will not fluctuate that much.

2. Both of them achieve their best performance when *number of processors* are between 4 and 8, which is about 0.1 second. It's because we have 4 processors in each machine, which means that when the manager has 3 "local" workers, they can all be allocated vectors using the system bus instead of Ethernet. However, as the scale of the image database or the number of processors increase, this effect becomes less apparent (hence the increased query time for runs with more than 8 processors).

Finally, we ran a test case where we set the block size based on Equation 0 (e.g. when we have n processors, then each share is 9907/n-1). In this case each Worker only needs to perform one iteration (which will be perfectly balanced in terms of feature vector assignment). On a side node, this case could leverage one more process by not using the Manager-Worker paradigm and simply assigning all processes an equal number of vectors.

The result is illustrated as Figure 7.

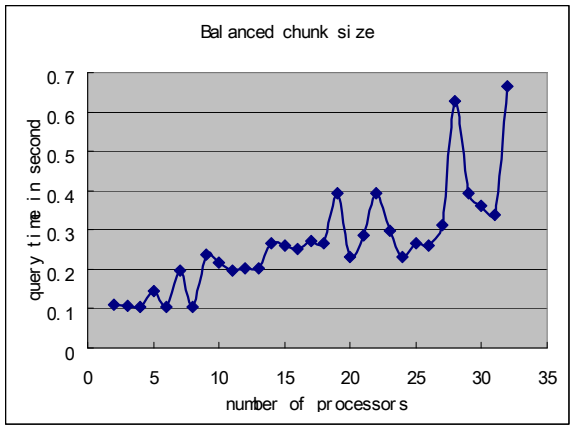

Figure 7. Online query performance when block size is balanced.

We can see that it achieves the best performance when using 4-8 processors. Besides, for this image DB size, this method looks better than has preset *blocksize*. However if we have a larger DB, then it will not hold.

Compared to the sequential version of this CBIR problem which can be seen on website [4], usually it takes about 0.1229 seconds to finish the whole procedure.

Thus the speedup can be easily computed as

Speedup = 0.1229/0.1011 = 1.21

When using 4-8 processors. This is not as high as we hoped. However, we plan to test this on much larger feature database sizes and expect that more significant speedup will begin to show for databases with over 100,000 feature vectors.

5. Program interfaces

One of the challenges we encountered while creating a suitable interface is that the UC Beowulf cluster doesn't allow a direct connection from any IP outside of the UC campus network.

However, the CS department has provided a cyberspace for each student, which can be use as a "front-end" of our system. By running some PHP codes on that server, we can make a bridge to Beowulf cluster, to enable parallel image retrieval performance.

The program interface is shown in Figure 3.

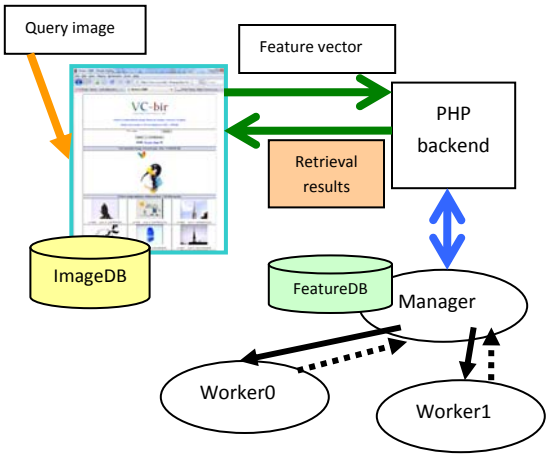

Figure 3. program interface

6. Conclusion and future work

In this project, we implement a parallelized version of content based image retrieval system.

By applying the manager-worker model, we successfully make this problem scalable. Theoretical analysis and experiment results are compared for the time-sensitive online query stage. Optimization on tuning the parameters such as *blocksize*, number of workers, is also performed and discussed.

The maximum speedup is approximately 1.2, using 4-8 processors, cutting the retrieval time a small margin. This however, is very promising when looking towards larger databases.

Future work can be performed in the following directions:

1. Scale up to larger image database;
2. Using more than 1 image feature, like [5].
3. Implementing and testing some of the discussed optimizations.

References

[1] Y Lu, W. Yu, et al. *Study of Content-Based Image Retrieval Using Parallel Computing Technique,* Asian Technology Information Program, 2007, 187~191

[2] Wikipedia, *Content-Based Image Retrieval*, http://en.wikipedia.org/wiki/CBIR

[3] C. Cheung, L. Po, K. Wong. *Web-based Beowulf-Class Parallel Computing on Image Database Indexing and Retrieval System*, Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2001, 457-460

[4] C Fang. *Victor's Content-Based Image Retrieval system*, Univ. of Cincinnati, 2008.

http://www.cs.uc.edu/~fangcg/php/vcbir.php

[5] C. Fang, S. Visa, M. Ionescu, A. Ralescu, ***Experiments on Probability based Similarity Measures Applied to Image Similarity***, Int'l Conf. for Pattern Recognition, Tampa, FL, Dec 2008.